

COMMUNICATIONS

CACM.ACM.ORG OF THE

ACM

03/2016 VOL.59 NO.03

Software Obfuscation

A Roundtable Discussion
on Google's Move to SDN

NLP Is Breaking Out

An Interview with
John Hennessy

Lessons from
20 Years of MINIX

Association for
Computing Machinery





SCIENCE OF SECURITY

Eligibility: Any paper that exhibits **outstanding contribution** to cybersecurity science and published from **Jan. 1, 2015 - Dec. 31, 2015**

Entries judged on scientific merit, the strength and significance of the work reported, and the degree to which the papers exemplify how to perform and report cybersecurity scientific research.

**3rd Annual
Best Scientific
Cybersecurity
Winning Paper**

*"Additive and Multiplicative
Notions of Leakage and
Their Capacities"*

Nominations close March 31, 2016.

**Submit nominations at
sos-vo.org/papercompetition**

Hosted by the Research Directorate, National Security Agency

4TH ANNUAL
**Best Scientific
Cybersecurity
Paper
Competition**

CAREERS at the NATIONAL SECURITY AGENCY

EXTRAORDINARY WORK

Inside our walls, you will find the most extraordinary people doing the most extraordinary work. Not just finite field theory, quantum computing or RF engineering. Not just discrete mathematics or graph analytics.

It's all of these and more, rolled into an organization that leads the world in signals intelligence and information assurance.

Inside our walls you will find extraordinary people, doing extraordinary work, for an extraordinary cause: the safety and security of the United States of America.

U.S. citizenship is required. NSA is an Equal Opportunity Employer.



APPLY TODAY



NSA

WHERE INTELLIGENCE GOES TO WORK®



Search NSA to Download



www.IntelligenceCareers.gov/NSA

ICMI 2016

The 18th International Conference on Multimodal Interaction

November 12-16, 2016

Tokyo, Japan

<http://icmi.acm.org/2016>

Important Dates

Grand Challenge proposal	January 15th, 2016
Workshop proposal	February 20th, 2016
Paper submission	May 6th, 2016
Doctoral consortium submission	July 15th, 2016
Demo proposals	August 10th, 2016
Main Conference	November 13-15, 2016
Challenges/Doctoral consortium	November 12th, 2016
Workshops	November 16th, 2016

- ✓ Affective Computing and interaction
- ✓ Cognitive modeling and multimodal interaction
- ✓ Gesture, touch and haptics
- ✓ Healthcare, assistive technologies
- ✓ Human communication dynamics
- ✓ Human-robot/agent multimodal interaction
- ✓ Interaction with smart environment
- ✓ Machine learning for multimodal interaction
- ✓ Mobile multimodal systems
- ✓ Multimodal behavior generation
- ✓ Multimodal datasets and validation
- ✓ Multimodal dialogue modeling
- ✓ Multimodal fusion and representation
- ✓ Multimodal interactive applications
- ✓ Speech behaviors in social interaction
- ✓ System components and multimodal platforms
- ✓ Visual behaviors in social interaction
- ✓ Virtual/augmented reality and multimodal interaction

Organizing Committee

General Chairs

Yukiko Nakano (Seikei University, Japan)
Elisabeth Andre (Augsburg University, Germany)
Toyooki Nishida (Kyoto University, Japan)

Program Chairs

Louis-Philippe Morency (Carnegie Mellon University, USA)
Carlos Busso (University of Texas in Dallas, USA)
Catherine Pelachaud (CNRS-LTCl, Telecom-ParisTech, France)

Local Advisor

Kenji Mase (Nagoya University, Japan)

Workshop Chairs

Julien Epps (The University of New South Wales, Australia)
Gabriel Skantze (KTH, Sweden)

Demo Chairs

Ronald Poppe (Utrecht University, Netherlands)
Ryo Ishii (NTT, Japan)

Multimodal Grand Challenge Chairs

Hatice Gunes (Queen Mary University of London, UK)
Mohammad Soleymani (University of Geneva, Switzerland)

Doctoral Consortium Chairs

Dirk Heylen (University of Twente, Netherlands)
Samer Al Moubayed (Disney Research, USA)

Publication Chair

Albert Ali Salah (Bogazici University, Turkey)
Stefan Kopp (Bielefeld University, Germany)

Publicity Chairs

Oya Aran (Idiap Research Institute, Switzerland)
Dan Bohus (Microsoft Research, USA)

Sponsorship Chairs

Takatsugu Hirayama (Nagoya University, Japan)
Ryo Yonetani (The University of Tokyo, Japan)
Emily Mower Provost (University of Michigan, USA)

Local Organization Chairs

Hung-Hsuan Huang (Ritsumeikan University, Japan)
Shogo Okada (Tokyo Institute of Technology, Japan)

Finance Chair

Hung-Hsuan Huang (Ritsumeikan University, Japan)

Web Chair

Yutaka Takase (Seikei University, Japan)



Departments

- 5 **The Strength of Encryption**
By Eugene Spafford
-
- 7 **Cerf's Up**
Computer Science in the Curriculum
By Vinton G. Cerf
-
- 8 **Letters to the Editor**
ACM Moral Imperatives vs. Lethal Autonomous Weapons
-
- 10 **BLOG@CACM**
The Value of Ada
Valerie Barr considers the continuing attraction of the woman considered the first computer programmer.
-
- 29 **Calendar**
-
- 117 **Careers**

Last Byte

- 120 **Q&A**
A Graphics and Hypertext Innovator
Andries van Dam on interfaces, interaction, and why he still teaches undergraduates.
By Leah Hoffmann

News

- 13 **Deep or Shallow, NLP Is Breaking Out**
Neural net advances improve computers' language ability in many fields.
By Gregory Goth
-
- 17 **Rich Data, Poor Fields**
Diverse technologies help farmers produce food in resource-poor areas.
By Tom Geller
-
- 19 **When Computers Stand in the Schoolhouse Door**
Classification algorithms can lead to biased decisions, so researchers are trying to identify such biases and root them out.
By Neil Savage
-
- 22 **Peter Naur: 1928–2016**
Peter Naur, a Danish computer scientist and 2005 recipient of the ACM A.M. Turing Award, died recently after a brief illness.

Viewpoints

- 24 **Legally Speaking**
New Exemptions to Anti-Circumvention Rules
Allowing some reverse engineering of technical measures for non-infringing purposes.
By Pamela Samuelson
-
- 27 **Computing Ethics**
The Question of Information Justice
Information justice is both a business concern and a moral question.
By Jeffrey Johnson

Viewpoints (continued)

- 30 **The Profession of IT**
Fifty Years of Operating Systems
A recent celebration of 50 years of operating system research yields lessons for all professionals in designing offers for their clients.
By Peter J. Denning
-
- 33 **Broadening Participation**
The Need for Research in Broadening Participation
In addition to alliances created for broadening participation in computing, research is required to better utilize the knowledge they have produced.
By Tiffany Barnes and George K. Thiruvathukal
-
- 35 **Viewpoint**
Riding and Thriving on the API Hype Cycle
Guidelines for the enterprise.
By Maja Vukovic et al.
-
- 38 **Viewpoint**
Paper Presentation at Conferences: Time for a Reset
Seeking an improved paper presentation process.
By H.V. Jagadish
-
- 40 **Interview**
An Interview with Stanford University President John Hennessy
Stanford University President John Hennessy discusses his academic and industry experiences in Silicon Valley with UC Berkeley CS Professor David Patterson.
By David Patterson



Watch the authors discuss their work in this exclusive *Communications* video.
<http://cacm.acm.org/videos/an-interview-with-stanford-university-president-john-hennessy>

For the full-length video, please visit <https://vimeo.com/146145543>

Practice

46 **A Purpose-Built Global Network: Google's Move to SDN**

A discussion with Amin Vahdat, David Clark, and Jennifer Rexford.

55 **The Paradox of Autonomy and Recognition**

Thoughts on trust and merit in software team culture.

By Kate Matsudaira

58 **Automation Should Be Like Iron Man, Not Ultron**

The "Leftover Principle" requires increasingly more highly skilled humans.

By Tom Limoncelli



Articles' development led by **acmqueue**
queue.acm.org

**About the Cover:**

For almost as long as there has been software, there have been challenges over how to secure it. So many cryptographic schemes have been applied over the decades. Boaz Barak attests software obfuscation may in fact become the cryptographer's "master tool" enabling security and privacy applications heretofore out of reach.

Contributed Articles

62 **Repeatability in Computer Systems Research**

To encourage repeatable research, fund repeatability engineering and reward commitments to sharing research artifacts.

By Christian Collberg and Todd A. Proebsting

70 **Lessons Learned from 30 Years of MINIX**

MINIX shows even an operating system can be made to be self-healing.

By Andrew S. Tanenbaum



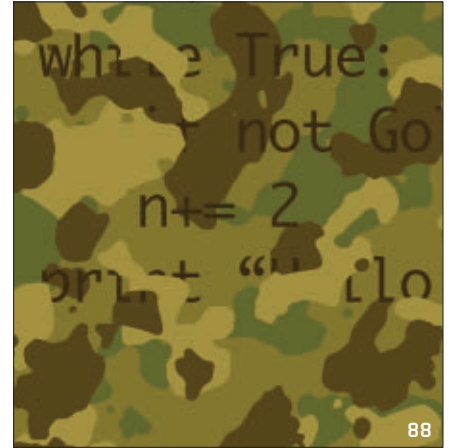
Watch the author discuss his work in this exclusive *Communications* video.
<http://cacm.acm.org/videos/lessons-learned-from-30-years-of-minix>

79 **A Lightweight Methodology for Rapid Ontology Engineering**

UPON Lite focuses on users, typically domain experts without ontology expertise, minimizing the role of ontology engineers.

By Antonio De Nicola and Michele Missikoff

Review Articles

88 **Hopes, Fears, and Software Obfuscation**

What does it mean to be secure?

By Boaz Barak



Watch the author discuss his work in this exclusive *Communications* video.
<http://cacm.acm.org/videos/hopes-fears-and-software-obfuscation>

Research Highlights

98 **Technical Perspective**
STACKing Up Undefined Behaviors
By John Regehr99 **A Differential Approach to Undefined Behavior Detection**
By Xi Wang, Nikolai Zeldovich, M. Frans Kaashoek, and Armando Solar-Lezama107 **Technical Perspective**
Taming the Name Game
By David Forsyth108 **Learning to Name Objects**
By Vicente Ordonez, Wei Liu, Jia Deng, Yejin Choi, Alexander C. Berg, and Tamara L. Berg



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.

Executive Director and CEO

Bobby Schnabel

Deputy Executive Director and COO

Patricia Ryan

Director, Office of Information Systems

Wayne Graves

Director, Office of Financial Services

Darren Ramdin

Director, Office of SIG Services

Donna Cappel

Director, Office of Publications

Bernard Rous

Director, Office of Group Publishing

Scott E. Delman

ACM COUNCIL

President

Alexander L. Wolf

Vice-President

Vicki L. Hanson

Secretary/Treasurer

Erik Altman

Past President

Vinton G. Cerf

Chair, SGB Board

Patrick Madden

Co-Chairs, Publications Board

Jack Davidson and Joseph Konstan

Members-at-Large

Eric Allman; Ricardo Baeza-Yates;

Cherri Pancake; Radia Perlman;

Mary Lou Soffa; Eugene Spafford;

Per Stenström

SGB Council Representatives

Paul Beame; Jenna Neefe Matthews;

Barbara Boucher Owens

BOARD CHAIRS

Education Board

Mehran Sahami and Jane Chu Prey

Practitioners Board

George Neville-Niel

REGIONAL COUNCIL CHAIRS

ACM Europe Council

Dame Professor Wendy Hall

ACM India Council

Srinivas Padmanabhuni

ACM China Council

Jianguang Sun

PUBLICATIONS BOARD

Co-Chairs

Jack Davidson; Joseph Konstan

Board Members

Ronald F. Boisvert; Anne Condon;

Nikil Dutt; Roch Guerrin; Carol Hutchins;

Yannis Ioannidis; Catherine McGeoch;

M. Tamer Ozsu; Mary Lou Soffa; Alex Wade;

Keith Webster

ACM U.S. Public Policy Office

Renee Dopplick, Director

1828 L Street, N.W., Suite 800

Washington, DC 20036 USA

T (202) 659-9711; F (202) 667-1066

Computer Science Teachers Association

Mark R. Nelson, Executive Director

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.

STAFF

DIRECTOR OF GROUP PUBLISHING

Scott E. Delman

cacm-publisher@cacm.acm.org

Executive Editor

Diane Crawford

Managing Editor

Thomas E. Lambert

Senior Editor

Andrew Rosenbloom

Senior Editor/News

Larry Fisher

Web Editor

David Roman

Rights and Permissions

Deborah Cotton

Art Director

Andrij Borys

Associate Art Director

Margaret Gray

Assistant Art Director

Mia Angelica Balaquiot

Designer

Iwona Usakiewicz

Production Manager

Lynn D'Addesio

Director of Media Sales

Jennifer Ruzicka

Publications Assistant

Juliet Chance

Columnists

David Anderson; Phillip G. Armour;

Michael Cusumano; Peter J. Denning;

Mark Guzdial; Thomas Haigh;

Leah Hoffmann; Mari Sako;

Pamela Samuelson; Marshall Van Alstyne

CONTACT POINTS

Copyright permission

permissions@cacm.acm.org

Calendar items

calendar@cacm.acm.org

Change of address

acmhq@acm.org

Letters to the Editor

letters@cacm.acm.org

WEBSITE

<http://cacm.acm.org>

AUTHOR GUIDELINES

<http://cacm.acm.org/>

ACM ADVERTISING DEPARTMENT

2 Penn Plaza, Suite 701, New York, NY

10121-0701

T (212) 626-0686

F (212) 869-0481

Director of Media Sales

Jennifer Ruzicka

jen.ruzicka@hq.acm.org

Media Kit acmm mediasales@acm.org

Association for Computing Machinery (ACM)

2 Penn Plaza, Suite 701

New York, NY 10121-0701 USA

T (212) 869-7440; F (212) 869-0481

EDITORIAL BOARD

EDITOR-IN-CHIEF

Moshe Y. Vardi

vic@cacm.acm.org

NEWS

Co-Chairs

William Pulleyblank and Marc Snir

Board Members

Mei Kobayashi; Kurt Mehloth;

Michael Mitzenmacher; Rajeev Rastogi

VIEWPOINTS

Co-Chairs

Tim Finin; Susanne E. Hambrusch;

John Leslie King

Board Members

William Aspray; Stefan Bechtold;

Michael L. Best; Judith Bishop;

Stuart I. Feldman; Peter Freeman;

Mark Guzdial; Rachelle Hollander;

Richard Ladner; Carl Landwehr;

Carlos Jose Pereira de Lucena;

Beng Chin Ooi; Loren Terveen;

Marshall Van Alstyne; Jeannette Wing



PRACTICE

Co-Chair

Stephen Bourne

Board Members

Eric Allman; Peter Bailis; Terry Coatta;

Stuart Feldman; Benjamin Fried;

Pat Hanrahan; Tom Killalea; Tom Limoncelli;

Kate Matsudaira; Marshall Kirk McKusick;

George Neville-Niel; Theo Schlossnagle;

Jim Waldo

The Practice section of the CACM

Editorial Board also serves as

the Editorial Board of [ACM PRACTICE](http://www.acm.org/journals/practice)

CONTRIBUTED ARTICLES

Co-Chairs

Andrew Chien and James Larus

Board Members

William Aiello; Robert Austin; Elisa Bertino;

Gilles Brassard; Kim Bruce; Alan Bundy;

Peter Buneman; Peter Druschel;

Carlo Ghezzi; Carl Gutwin; Gal A. Kaminka;

James Larus; Igor Markov; Gail C. Murphy;

Bernhard Nebel; Lionel M. Ni; Kenton O'Hara;

Sriram Rajamani; Marie-Christine Rousset;

Avi Rubin; Krishan Sabnani;

Ron Shamir; Yoav Shoham; Larry Snyder;

Michael Vitale; Wolfgang Wahlster;

Hannes Werthner; Reinhard Wilhelm

RESEARCH HIGHLIGHTS

Co-Chairs

Azer Bestavros and Gregory Morrisett

Board Members

Martin Abadi; Amr El Abbadi; Sanjeev Arora;

Nina Balcan; Dan Boneh; Andrei Broder;

Doug Burger; Stuart K. Card; Jeff Chase;

Jon Crowcroft; Sandhya Dwaekadas;

Matt Dwyer; Alon Halevy; Norm Jouppi;

Andrew B. Kahng; Sven Koenig; Xavier Leroy;

Steve Marschner; Kobbi Nissim;

Steve Seitz; Guy Steele, Jr.; David Wagner;

Margaret H. Wright; Andreas Zeller

WEB

Chair

James Landay

Board Members

Marti Hearst; Jason I. Hong;

Jeff Johnson; Wendy E. MacKay

ACM Copyright Notice

Copyright © 2016 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

For other copying of articles that carry a code at the bottom of the first or last page or screen display, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center; www.copyright.com.

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 acmhq@acm.org.

COMMUNICATIONS OF THE ACM

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

POSTMASTER

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

Printed in the U.S.A.



Association for
Computing Machinery



The Strength of Encryption

SOME OF THE most perplexing and frustrating experiences that technologists have are with politics and social policy. There are issues that have overwhelming data and scientific analyses to support a position, but value systems based on economics, religion, and/or misinformation are relied upon instead—usually to the consternation of the scientists and engineers. Examples abound, from issues such as the anthropogenic contributions to climate change, the safety of childhood inoculations, and the nature of evolution. Amazing to most of us, there are even those who are certain the Earth is flat! Furthermore, to hold some of these positions requires also believing that scientists are either ignorant or corrupt.

Computing is not immune to these conflicts. One that is currently playing out involves encryption, and what (if anything) should be done to regulate it. Some officials involved in law enforcement and in government are concerned about the potential impact of encryption and wish to restrict how and where it can be used. Many computing professionals have a different set of views, and stress that restrictions to weaken encryption will be much more harmful than helpful.

Conflicts over encryption are not new, with historical examples stretching back many centuries. What has made it a more pressing issue in recent decades is the strength of encryption used with computers, and the immediacy and scope of digital communication. Some 20 years ago, the U.S. had an active controversy over the role of allowing encryption in commercial products. Elements of law enforcement were concerned about the potential for criminals—particularly child pornographers and drug traffickers—to hide evidence of their activities from authorized investigations. Some in the national intelligence community were also worried that export of strong encryption technology might harm national intelligence capabilities.

Efforts by technologists and civil libertarians (including ACM's U.S. Public

Policy Committee) helped shape the discussion in the U.S., as did an extensive study by the National Academies. The outcome was a Presidential decision to not interfere with use of encryption, with some limits on the strength of exported technologies. There was considerable grumbling by some in the law enforcement community, but the decision proved to be sound—in the decades since then we have not been overrun by criminals using encryption (although some exist). Meanwhile, organizations around the globe have had the benefit of strong encryption to protect their information resources.

The evolution of the technology we use regularly has incorporated stronger, built-in encryption. This is especially the case in personal devices such as smartphones and tablets, and in systems supported by some ISPs. The encryption that is present is there to protect the user community from information theft and abuse. However, these same mechanisms may prevent law enforcement from accessing information during their authorized investigations. Globally, we have seen increasing instances of sophisticated crime involving computer-based resources. Terrorism is effective when it induces fear, and unfortunately, recent terrorist events (and political opportunism around them) have generated heightened public concern. In response, law enforcement officials in several countries have felt greater urgency to investigate and forestall any new such activity. To accomplish this, they wish to be able to intercept and monitor communications of suspects, and to be able to capture and analyze their stored data. As such, they are seeking to mandate products incorporating some authorized encryption circumvention technique, colloquially called a “backdoor.”

Here is where the conflict with technologists comes about. Those of us who have studied encryption know that inclusions of “backdoors” weaken encryption schemes, and do not know of any practical way of enabling any such circumvention in a manner that is itself sufficiently robust. Having any sort of al-

ternate key mechanism often makes the encryption weaker. It would also highlight the holders of that key as targets to attack, as well as enable insider abuse. Furthermore, once compromised everyone would be endangered—and there is little doubt such a scheme would be compromised or leaked eventually. Unfortunately, some policymakers, perhaps conditioned by TV shows and movies with unrealistic portrayals of computing, do not believe the warnings. A few national governments, such as the Netherlands, have taken the position that encryption should not be weakened. However, others—including the U.S. and the U.K.—appear to be on paths toward legislating weakening of commercially available encryption sold within their borders.

Ironically, the results of the “crypto wars” of 20 years ago means anyone who really wants strong encryption can obtain it and layer it on their regular platforms (superencryption). The Daesh (ISIL) already has crypto applications they provide to some of their operatives that do exactly that. Thus, any restrictions will only weaken the protections for the rest of us against criminal activities, economic espionage, and overly intrusive governments. Once lost, it may take a long time to regain the privacy and security afforded by strong encryption.

What will it take to resolve this conflict? To start, it would help if all sides accepted that their counterparts are neither fundamentally venal nor oblivious to the issues involved. There are genuine concerns all around, but education and exploration of issues is required. ACM, as the preeminent computing association globally, has the potential to have a strong voice in mediating this discussion. As ACM members, we should seek to help clarify the issues with our political representatives in such a way as to define a workable way forward ... and that won't need to be revisited in another two decades. 

Eugene H. Spafford (spaf@acm.org) is an at-large member of ACM Council.

Copyright held by author.



ACM Books



MORGAN & CLAYPOOL
PUBLISHERS

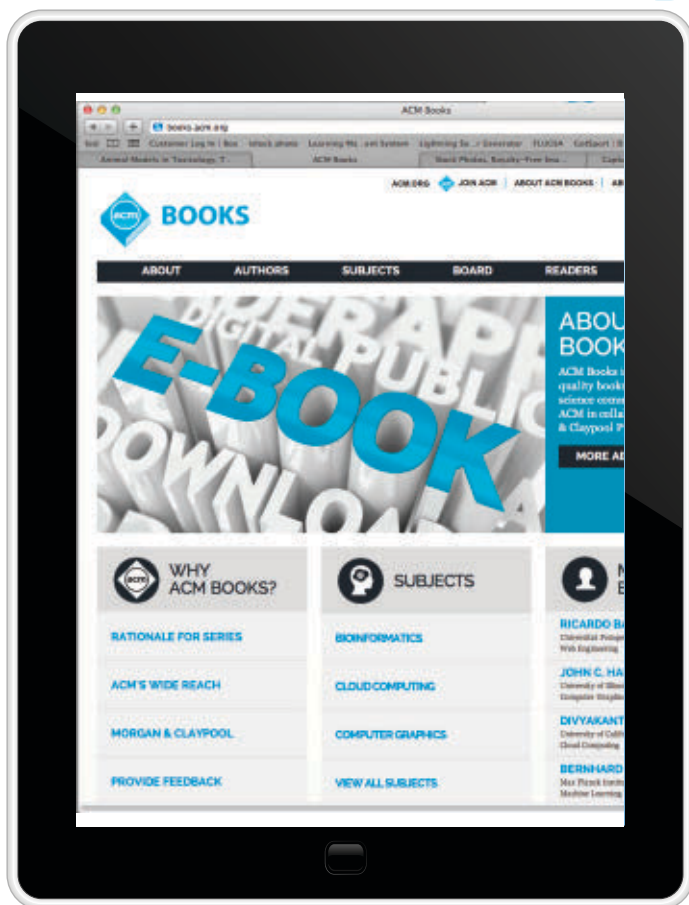
Publish your next book in the ACM Digital Library

ACM Books is a new series of advanced level books for the computer science community, published by ACM in collaboration with Morgan & Claypool Publishers.

I'm pleased that ACM Books is directed by a volunteer organization headed by a dynamic, informed, energetic, visionary Editor-in-Chief (Tamer Özsu), working closely with a forward-looking publisher (Morgan and Claypool).

—Richard Snodgrass, University of Arizona

books.acm.org **ACM Books**



- ◆ will include books from across the entire spectrum of computer science subject matter and will appeal to computing practitioners, researchers, educators, and students.
- ◆ will publish graduate level texts; research monographs/overviews of established and emerging fields; practitioner-level professional books; and books devoted to the history and social impact of computing.
- ◆ will be quickly and attractively published as ebooks and print volumes at affordable prices, and widely distributed in both print and digital formats through booksellers and to libraries and individual ACM members via the ACM Digital Library platform.
- ◆ is led by EIC M. Tamer Özsu, University of Waterloo, and a distinguished editorial board representing most areas of CS.

Proposals and inquiries welcome!

Contact: **M. Tamer Özsu**, Editor in Chief
booksubmissions@acm.org



Association for
Computing Machinery

Advancing Computing as a Science & Profession



DOI:10.1145/2889282

Vinton G. Cerf

Computer Science in the Curriculum

As I write this, I have just come off a conference call with White House representatives from the Office of Science and Technology Policy including

Megan Smith, the U.S. Chief Technology Officer, Thomas Kalil, the Deputy Director for Technology and Innovation, among many others. Highlighted was President Obama's "Computer Science for All" program^a that is proposing \$4B to the states and \$100M directly to expand K-12 computer science in the U.S. by training teachers, providing access to instructional materials, and building regional partnerships. The National Science Foundation (NSF^b) and the Corporation for National and Community Service (CNCS^c) will provide an immediate \$135M into the program. The earlier CS10K^d program provided computer science principles and exploring computer science curriculum materials to introduce the subject to students who might not have otherwise considered it. Topics include human-computer interaction, problem solving, Web design, programming, computing and data analysis, and robotics.

Computer science is in significant measure all about analyzing problems, breaking them down into manageable parts, finding solutions, and integrating the results. The skills needed for this kind of thinking apply to more than computer programming. They offer a kind of disciplined mind-set that is applicable to a broad range of design and

implementation problems. These skills are helpful in engineering, scientific research, business, and even politics! Even if a student does not go on to a career in computer science or a related subject, these skills are likely to prove useful in any endeavor in which analytical thinking is valuable.

Establishing and bolstering CS education for school-age children is a challenge felt globally. In the U.S., progress has been painfully slow. What is very gratifying about this announcement is that ACM has long had an aspiration to make computer science a normal part of school curriculum in the U.S., especially in K-12 (in fact, the April issue will feature a news story detailing ACM's involvement in the CS for All program). It is hoped this effort, with the funding that enables it, will make computer science as acceptable as physics, chemistry, and biology for satisfying curricular requirements for high school graduation. Moreover, the appearance of computer science in the elementary and middle school curriculum may attract a much broader range of students to careers involving computing. It is my belief that many potential computer science and electrical engineering students are lost to these disciplines because they have not had the opportunity to explore them early enough in their school years to discover their power and attraction.

With such a program in place, ACM's members have new opportunities to engage in the K-12 space to convey their interest and fascination with the many facets of computer science. When this is combined with the so-called maker culture,^e it seems to me

a powerful recipe for progress emerges. Students learning by doing, sharing their work, and learning from each other creates a force to be reckoned with. One of the drivers of the maker culture is 3D printing.^f Like the evolution of the World Wide Web and the webmasters whose pages we all enjoy, the 3D printing world offers the opportunity for inventors and designers to share their designs, to learn from one another, and to advance the state of the art through collaborative processes.

Anyone who has watched children as young as three years old playing with tablets and even smartphones must appreciate the newest generation(s) are steeped in software and comfortable with its use in a way earlier generations were not. It is sometimes said "Technology is what you *didn't* grow up with." If you grew up with it, it is just there. No big deal. For someone who began life with a three-party telephone and no television, let alone computers, I find myself thinking I should have a T-shirt made reading "Don't look back, there is a three-year-old gaining on you!"

However you read the tea leaves, the new CS for All program is a major step forward for all of us who believe that exposure to computer science and its way of thinking is in everyone's best interest. □

^e https://en.wikipedia.org/wiki/Maker_culture

^f https://en.wikipedia.org/wiki/3D_printing

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

Copyright held by author.

^a <https://www.whitehouse.gov/blog/2016/01/30/computer-science-all>

^b <https://www.nsf.gov>

^c <http://www.nationalservice.gov/programs/amer-corps/computer-science-teachers-amer-corps>

^d <https://cs10kcommunity.org/> and <https://dl.acm.org/citation.cfm?id=1953193>

ACM Moral Imperatives vs. Lethal Autonomous Weapons

MOSHE Y. VARDI'S editor's letter "On Lethal Autonomous Weapons" (Dec. 2015) said artificial intelligence is already found in a wide variety of military applications, the concept of "autonomy" is vague, and it is near impossible to determine the cause of lethal actions on the battlefield. It described as "fundamentally vague" Stephen Goose's ethical line in his Point side of the Point/Counterpoint debate "The Case for Banning Killer Robots" in the same issue. I concur with Vardi that the issue of a ban on such technology is important for the computing research community but think the answer to his philosophical logjam is readily available in the "ACM Code of Ethics and Professional Conduct" (<http://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct>), particularly its first two "moral imperatives"—"Contribute to society and human well-being" and "Avoid harm to others." I encourage all ACM members to read or re-read them and consider if they themselves should be working on lethal autonomous weapons or even on any kind of weapon.

Ronald Arkin's Counterpoint was optimistic regarding robots' ability to "... exceed human moral performance ...," writing that a ban on autonomous weapons "... ignores the moral imperative to use technology to reduce the atrocities and mistakes that human warfighters make." This analysis involved two main problems. First, Arkin tacitly assumed autonomous weapons will be used only by benevolent forces, and the "moral performance" of such weapons is incorruptible by those deploying them. The falsity of these assumptions is itself a strong argument for banning such weapons in the first place. Second, the reasons he cited in favor of weaponized autonomous robots are equally valid for a simpler and more sensible proposal—autonomous safeguards on human-controlled weapons systems.

What Arkin did not say was why the

world even needs weaponized robots that are autonomous. To do so, I suggest he first conduct a survey among the core stakeholder group he identified—civilian victims of war crimes—to find out what they think.

Bjarte M. Østvold, Oslo, Norway

Author's Response:

The desire to eliminate war is an old one, but war is unlikely to disappear in the near future. "Just War" theory postulates that war, while terrible, is not always the worst option. As much as we may wish it, information technology will not get an exemption from military applications.

Moshe Y. Vardi, Editor-in-Chief

Technological Superiority Lowers the Barrier to Waging War

I am writing to express dismay at the argument by Ronald Arkin in his Counterpoint in the Point/Counterpoint section "The Case for Banning Killer Robots" (Dec. 2015) on the proposed ban on lethal autonomous weapons systems. Arkin's piece was replete with high-minded moral concern for the "... *status quo with respect to innocent civilian casualties* ..." [italics in original], the depressing history of human behavior on the battlefield, and, of course, for "... our young men and women in the battlespace ... placed into situations where no human has ever been designed to function." There was an incongruity in Arkin's position only imperfectly disguised by these sentiments. While deploring the "... regular commission of atrocities ...," in warfare, there was nowhere in Arkin's Counterpoint (nor, to my knowledge, anywhere in his extensive writings) any corresponding statement deploring the actions of the U.S. President and his advisors, who, in 2003, through reliance on the technological superiority they commanded, placed U.S. armed forces in the situations that gave us, helter-skelter, the images of tens of thousands of inno-

cent civilian casualties, many thousands of men and women combatants returning home mutilated or psychologically damaged, and the horrors of Abu Ghraib military prison.

Is it still surprising that an enemy subject to the "magic" of advanced weapons technology would resort to the brutal minimalist measures of asymmetric warfare, and combatants who see their comrades maimed and killed by these means sometimes resort to the behavior Arkin deplores?

In the face of clear evidence that technological superiority lowers the barrier to waging war, Arkin proposed the technologist's dream—weapons systems engineered with an ethical governor to "... outperform humans with respect to international humanitarian law (IHL) in warfare (that is, be more humane) ..." Perfect! Lower the barrier to war even further, reducing consideration of harm and loss to one's own armed forces—at the same time representing it as a gentleman's war, waged at the highest ethical level.

Above all, I reject Arkin's use of the word "humane" in this context. My old dictionary in two volumes¹ gives this definition:

Humane—"Having or showing the feelings befitting a man, esp. with respect to other human beings or to the lower animals; characterized by tenderness and compassion for the suffering or distressed."

Those, like Arkin, who speak of "ethical governors" implemented in software, or of robots behaving more "humanely" than humans are engaging in a form of semantic sleight of hand the ultimate consequence of which is to debase the deep meaning of words and reduce human feeling, compassion, and judgment to nothing more than the result of a computation. Far from fulfilling, as Arkin wrote, "... our responsibility as scientists to look for effective ways to reduce man's inhumanity to man through technology ...," this is a mockery and a betrayal of our humanity.

Reference

1. Emery, H.G. and Brewster, H.K., Eds. *The New Century Dictionary of the English Language*. D. Appleton-Century Company, New York, 1927.

William M. Fleischman, Villanova, PA

Author's Response:

*While Fleischman questions my motive, I contend it is based solely on the right to life being lost by civilians in current battlefield situations. His *jus ad bellum* argument, lowering the threshold of warfare, is common and deserves to be addressed. The lowering of the threshold of warfare holds for the development of any asymmetric warfare technology—robotics is just one—that provides a one-sided advantage, as one might see in, say, cyberwarfare. Yes, it could encourage adventurism. The solution then is to stop all research into advanced military technology. If Fleischman can make this happen I would admire him for it. But in the meantime we must protect civilians better than we do, and technology can, must, and should be applied toward this end.*

Ronald C. Arkin, Atlanta, GA

Braces Considered Loopy

The “naked braces” discussion, beginning with A. Frank Ackerman’s letter to the editor “Ban ‘Naked’ Braces!” (Oct. 2015), perhaps misses the forest for the trees, as a major reason for deeply nested expressions is the inability of most programming languages to handle arrays without looping. This shortcoming further compounds itself by contributing to the verbosity of the boilerplate required for such looping (and multi-conditional) constructs.

Jamie Hale’s proposed solution in his letter to the editor “Hold the Braces and Simplify Your Code” (Jan. 2016)—including “... small and minimally nested blocks ...”—to the issue first raised by Ackerman pointed in a good direction but may remain lost in the forest of intrinsically scalar languages. Small blocks of code are good, but, in most languages, doing so merely results in a plethora of small blocks, pushing the complexity to a higher level without necessarily reducing it.

A more functional, array-based way of looking at problems can, however, reduce that apparent complexity by treating collections of objects en masse at a higher level.

Given most programmers’ lack of familiarity with array-oriented programming, it is difficult for anyone, including me, to provide a widely comprehensible pseudocode example of what I mean by this, but consider the following attempt, based on the problem of invoking different code based on transaction-

size breakpoints (where “transactions” is a vector of transaction sizes)

```
Perform ( smallT(), largeT() )
BasedOn transactions > 1000
```

Ignoring the length discrepancy between the number of functions provided and the ostensible shape of the Boolean condition on which their selection is based, such a construct could easily be extended to additional breakpoints with something like this

```
Perform ( smallT(), largeT(),
veryLargeT() )
BasedOn transactions > ( 1e3, 1e5 )
```

For anyone interested in wrestling with a specific example of an array-based functional notation guiding my thoughts on this example, see <http://code.jsfiddle.com/wiki/Vocabulary/atdot>.

Devon McCormick, New York, NY

Programming By Any Other Name

Thomas Haigh’s and Mark Priestley’s Viewpoint “Where Code Comes From: Architectures of Automatic Control from Babbage to Algol” (Jan. 2016) focused on the words “code” and “programming” and how they came to be defined as they are today. However, it also mentioned other types of programming from the days before those words took their current meaning, without acknowledging they were exactly the same in the minds of those scientists and “card jockeys” who diagrammed analog computers or charted the progress of a job on the “data processing” floor and wired the plugboards of the unit record equipment on that floor. If no scholar has in fact published a looking-backward article on the plugboard wiring of those machines from the modern programmer’s perspective, someone should. If you have never wired a plugboard, I urge you to try it. Teach yourself to sense a pulse and make something useful happen or debug a problem when someone dislodges a cable. Once you understand the machine, you will find you step immediately into programming mode, whereby the cable is the code, the plugboard the subroutine, and the floor the program. Drawing flow diagrams was, once upon a time, what programming was about, no matter what the target environment happened to be.

The only programmer I ever met who coded a significant production

program on a UNIVAC SSII 80 (circa 1963) computer and saw it run successfully on its first shot, was an old plugboard master. He flowcharted the program the way he learned to flowchart a machine-room job. The “concept” of programming was nothing new to him.

Ben Schwartz, Byram Township, NJ

Decidability Does Not Presuppose Gödel-Completeness

Contrary to what Philip Wadler suggested in his otherwise interesting and informative article “Propositions as Types” (Dec. 2015, page 76, middle column, third paragraph), the algorithmic decidability of an axiomatically defined theory, say, T (such as Set Theory, as in Hilbert’s concern) does not presuppose the negation- or “Gödel-” completeness (not to be confused with the semantic completeness) of T . First, negation-completeness does not imply algorithmic decidability without further ado, and second, negation-incompleteness does not imply algorithmic undecidability. The negation-completeness of T does indeed imply the algorithmic decidability of T if the set of axioms of T is algorithmically decidable and T is consistent (recall Recursion Theory), and there are theories that are negation-incomplete but algorithmically decidable (such as temporal-logical theories), respectively.¹

Reference

1. Kramer, S. Logic of negation-complete interactive proofs (formal theory of epistemic deciders). *Electronic Notes in Theoretical Computer Science* 300, 21 (Jan. 2014), 47–70, section 1.1.1.

Simon Kramer, Lausanne, Switzerland

Author's Response:

Thank you to Simon Kramer for clarifying the relation between completeness and decidability. The word “presupposes” has two meanings: “require as a precondition of possibility or coherence” and “tacitly assume at the beginning of a line of argument or course of action that something is the case.” Kramer presupposes I mean the former, when in fact I mean the latter; my apologies for any confusion. The logics in question are consistent and have algorithmically decidable axioms and inference rules, so completeness indeed implies decidability.

Philip Wadler, Edinburgh, Scotland

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.



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

DOI:10.1145/2874307

<http://cacm.acm.org/blogs/blog-cacm>

The Value of Ada

Valerie Barr considers the continuing attraction of the woman considered the first computer programmer.



Valerie Barr
"Why is Ada Lovelace Still the Woman That Young (and Not So Young) Women Look To?"

<http://bit.ly/1jT0tSp>
December 10, 2015

I was lucky enough to represent ACM and ACM-W at the recent Ada Lovelace Symposium at Oxford University, celebrating her 200th birthday. I participated in a panel "Enchantress of Abstraction and Bride of Science: can women scientists escape being icons, role-models and heroines." Following are my remarks:

Why? Why do so many current organizations and events identify with and recognize Ada Lovelace? We are well into the 21st century; Ada was born 200 years ago. Why do so many women today seem to look to her as a model and icon? How is it that this woman, who lived her life in the 1800s, can be so important today to women in computing, especially when, by and large, people know very little about the detail and depth of her accomplishments?

Thomas Haigh and Mark Priestley discuss Ada in their September 2015 piece in *Communications* entitled "In-

novators Assemble: Ada Lovelace, Walter Isaacson, and the Super-heroines of Computing" (<http://bit.ly/1MRkFeo>). While they make a number of good points in their piece, there are a number of problems as well. I'll digress long enough to comment on only one of the problems.

They state that "most areas of science and engineering are gradually becoming more balanced in their gender representation." This is a problematic statement for two reasons.

First, they do not place that comment geographically in the world, though based on the rest of the piece I assume they are talking about the U.S.

Second, assuming a U.S. focus, they do not account for changing demographics in the U.S. Today, almost 60% of college graduates in the U.S. are women. This skews any by-discipline view of gender representation.

The only accurate way to gauge relative balance is to look separately at women's degrees and men's degrees; that perspective shows us that we have a long way to go. Overall, in the U.S., 11% of women's degrees are earned in the STEM disciplines, while 24% of men's degrees are earned in the same fields. Only biology has true gender balance with 7% of women's degrees and 7% of men's degrees

earned in the field. And, since I'm sure you are wondering, less than 1% of women's degrees are earned in CS, while 5% of men's degrees are.

Returning to Ada, Haigh and Priestley argue, "The superhero narrative is not ... the best way to understand history." They argue for the historian's responsibility to "provide accurate and nuanced stories" and further that history will "ultimately prove more inspiring and more relevant than superhero stories." They make a compelling case, one I agree with, that we need to give more airtime to the many, many women who were involved in the development of computing as a technology and a field. They close by saying, "Superhero stories have little time for ordinary humans, who exist only to be endangered or rescued. Reducing the story of women in computing to the heroics of a handful of magical individuals draws attention away from real human experience and counterproductively suggests that only those with super-human abilities need apply."

So how do we make sense of this? How are we to understand the iconic nature of Ada as a figure for women in computing? And, frankly, why would anyone bother resurrecting a figure from such a different era?

Where I think Haigh and Priestley go wrong is at the outset, in the title, where they cast Ada as a superheroine. I would argue part of the value of Ada, the reason why she plays an important role, is that she actually is not seen as a superhero, she is not seen as being magical in some way.

I do believe, however, that part of her appeal is precisely because she is not of the modern world, because she comes from a different era, a different educational system, a completely different moment in time. This means today's young women are not dissuaded by her story because they know their life has not been and could not be like hers, so they feel no expectation they have to be exactly like Ada in order to succeed in computing.

Despite the historical differences, there is something very relatable about her for today's women. Her parents had some real problems (that might be the polite way of putting it), she did not have educational access equal to that of men with comparable intellect, and she was micromanaged day to day. Wow, that's the story of many, many women around the world today!

At the same time, she was in many ways able to ignore the script society wanted to write for her, or maybe she managed to just be somewhat unaware of it. She did what she wanted to do, engaged in the intellectual pursuits that clearly drove her and excited her, and seemingly went about her business. That is something well worth emulating!

Imagine for a moment, what if Ada were alive today? How would she measure up relative to some of today's female "superheroes" in tech? If we put Ada Lovelace on stage at the Grace Hopper Celebration of Women in Computing (<http://ghc.anitaborg.org/>), what would she talk about? I suspect she'd be up there, like roboticist Manuela Veloso recently was, talking about her latest technical work and giving credit to her graduate students; not like Sheryl Sandberg, whose big take-away message was "before you go to sleep every night, write down three things that you did well today." If we limit ourselves to those figures who are most hyped in the press today, is there anyone better than Ada to serve as a role model for today's female computer science student?

"If we limit ourselves to those figures who are most hyped in the press today, is there anyone better than Ada to serve as a role model for today's female computer science student?"

As Haigh and Priestley argue, we do have to do a much better job laying out who the key women in computing have been, and who they are today. Until then, we have a great gap, and that gap can actually dissuade women from coming into the field. Leaders, prominent figures, superheroes stand on the shoulders of lots of people below them, but if people hear only about the superheroes, then they will be dissuaded from even trying.

My eye was caught recently by an online listing of top 10 women in tech. I thought: "great, I can post this to the ACM-W Facebook page." I started reading the list and my next thought was, "why would I post this?" Given the lack of detail presented, there did not seem to be a single "regular" person on the list. Everyone on it was young and already worth millions or billions, was a founder or high-level executive in a major company like Lens Technology in Hong Kong, BET365 in the U.K., Epic Healthcare Software, Facebook, or YouTube. I do not mean to take away from the accomplishments of the women who lead these and other companies, but let us not pretend they got there on their own. Most often they had an extraordinary level of help and mentoring and coaching that is made invisible, and that makes it hard for them to be effective role models, because most people do not have access to the kinds of help they had.

A young woman sitting in her classroom today, or banging her head against a recalcitrant bug in her as-

signment, or working hard to get the next product release ready on time, is not likely to be motivated by the story of Marissa Mayer from Yahoo or Judy Faulkner from Epic. She might be motivated by the story of Margaret Hamilton (<http://bit.ly/1OLmmLw>), who developed the onboard flight software for the Apollo space program (<http://bit.ly/1Sy3Xrv>), or Sue Black (<http://bit.ly/1Pe73vj>), who did not follow the typical route into the field but got her Ph.D. at age 39 and is today a rock-star advocate for women in computing and a champion of Britain's role in the history of computing, or Dame Shirley (<http://bit.ly/1RfSHku>) and the women who worked with her.

While those stories are kept quiet, there is a dearth of role models for the majority of women who are in and might enter computing. In this context, Ada continues to serve very effectively as inspiration and as icon.

Reader's comment:

Two things occur to me; one personal and one historical.

First, for whatever reason, the idea of a role model has not (that I am conscious of) played a major part in my life. All along, there have been people I admired and wished to be more like, but never did I discover someone who made me say, "There, that is what I want to do, that is who I want to be." I was interested in subjects, areas, more than in persons, so naturally I'm unsure how much role models matter for others. My remarks may need to be discounted a good deal, because I am a white male, which is likely to have affected my view of what I could do. Still, as long as women can see that there IS a computing field and women are already working in it, just how much do particular role models count? I am not saying they do not; I am suggesting this is not the only thing we need to be concerned with.

Second, did Ada Lovelace need a role model in order to do what she did? I think, without knowing very much about her yet, that she did not. Does this make her a good example? I believe it does, but, as you are more or less saying through this essay, it depends on how you take her.

—John Branch

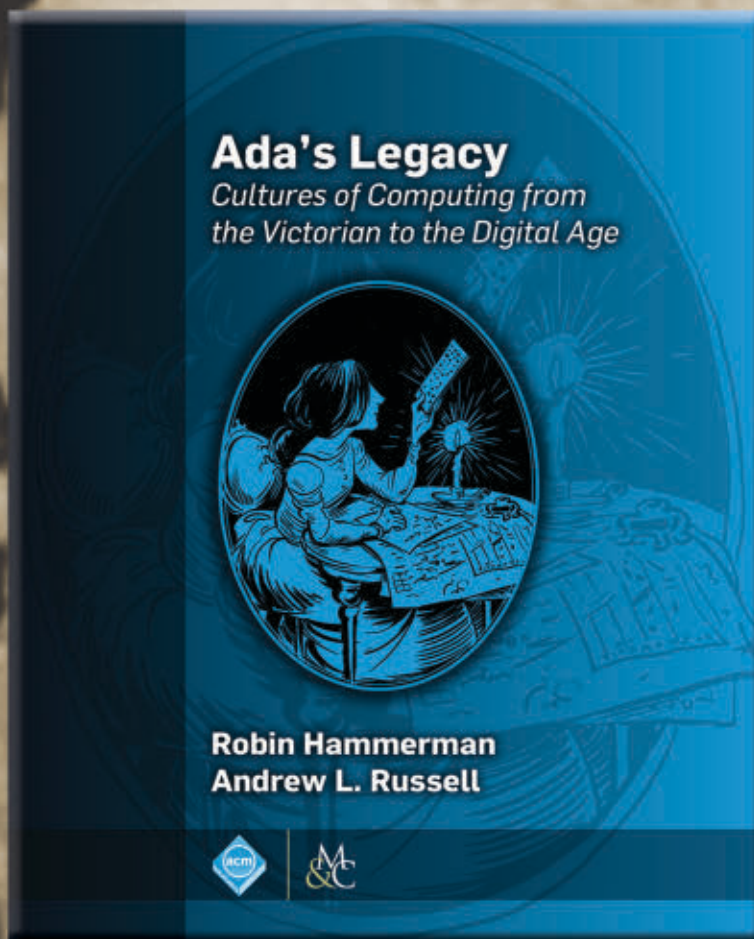
Valerie Barr is a professor at Union College, Schenectady, NY.

Copyright held by author.

Robin Hammerman and Andrew L. Russell

Ada's Legacy

Cultures of Computing from the
Victorian to the Digital Age



INSPIRING MINDS FOR 200 YEARS

Ada's Legacy illustrates the depth and diversity of writers, things, and makers who have been inspired by Ada Lovelace, the English mathematician and writer.

The volume commemorates the bicentennial of Ada's birth in December 1815, celebrating her many achievements as well as the impact of her work which reverberated widely since the late 19th century. This is a unique contribution to a resurgence in Lovelace scholarship, thanks to the expanding influence of women in science, technology, engineering and mathematics.

ACM Books is a new series of high quality books for the computer science community, published by the Association for Computing Machinery with Morgan & Claypool Publishers.



ISBN: 978-1-97000-148-8 DOI: 10.1145/2809523

<http://books.acm.org>

<http://www.morganclaypoolpublishers.com/acm>

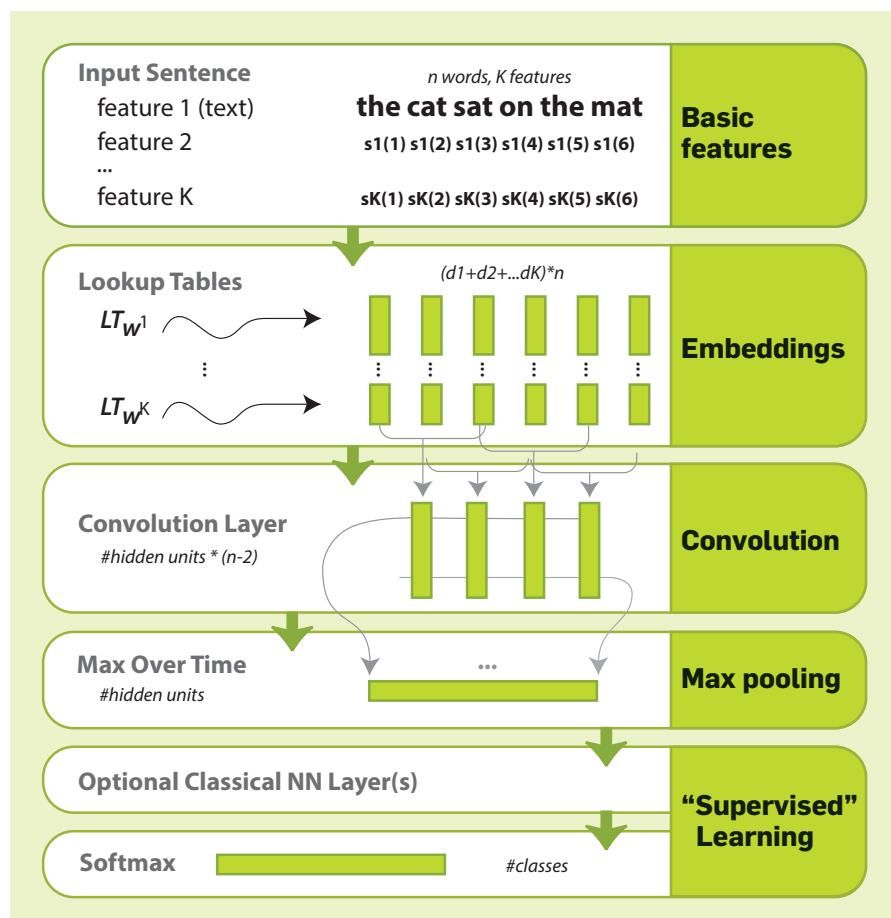
Deep or Shallow, NLP Is Breaking Out

Neural net advances improve computers' language ability in many fields.

ONE OF THE featured speakers at the inaugural Text By The Bay conference, held in San Francisco in April 2015, drew laughter when describing a neural network question-answering model that could beat human players in a trivia game.

While such performance by computers is fairly well known to the general public, thanks to IBM's Watson cognitive computer, the speaker, natural language processing (NLP) researcher Richard Socher, said, the neural network model he described "was built by one grad student using deep learning" rather than by a large team with the resources of a global corporation behind them.

Socher, now CEO of machine learning developer MetaMind, did not intend his remarks to be construed as a comparison of Watson to the academic model he and his colleagues built. As an illustration of the new technical and cultural landscape around NLP, however, the laughter Socher's comment drew was an acknowledgment that basic and applied research in language processing is no longer the exclusive province of those with either deep pockets or strictly academic intentions.



General Deep Architecture for NLP. Source: Collobert & Weston, Deep Learning for Natural Language Processing, 2009 Conference on Neural Information Processing Systems.

Indeed, new tools and new techniques—particularly open source technologies such as Google’s word2vec neural text processing tool—combined with steady increases in computing power, have broadened the potential for natural language processing far beyond the research lab or supercomputer. In domains as varied as finding pertinent news for a company’s potential investors to making hyper-personalized recommendations for online shopping to making music recommendations on streaming radio services, NLP is enabling everyday human-computer interaction in an ever-increasing range of venues. In the process, some of these advances are not only redefining what computers and humans can accomplish together, but also the very concept of what deep learning is.

Vectors Deep or Wide

One approach to natural language processing that has gained enormous traction in the past several years is representing words as vectors—that is, each word is given a series of scores that position it in an arbitrary space. This principle was explained by deep learning pioneer Geoffrey Hinton at a recent presentation to the Royal Society, the U.K. national academy of science. Hinton, a distinguished researcher for Google and distinguished professor emeritus at the University of Toronto, said, “The first thing you do with a word symbol is you convert it to a word vector. And you learn to do that, you learn for each word how to turn a symbol into a vector, say, 300 components, and after you’ve done learning, you’ll discover the vector for Tuesday is very similar to the vector for Wednesday.”

The result, Hinton said, is that given enough data, a language model can then generalize: for any plausible sentence with Tuesday in it, there’s a similar plausible sentence with Wednesday in it. More broadly, words with similar vector scores can be used to classify and cluster concepts. Companies using vector-based NLP technologies in production analyze concepts as varied as documents referring to a business’s financial activity or fashion customers’ reviews of a piece of clothing to try to help predict what type of customer will gravitate toward a certain style, much more quickly than could active human curation alone.

NLP is enabling everyday human-computer interaction in an ever-increasing range of venues.

In a recent interview with *Communications*, Hinton said his own research on word vectors goes back to the mid-1980s, when he, David Rumelhart, and Ronald Williams published work in *Nature* that demonstrated family relationships as vectors. “The vectors were only six components long because computers were very small then, but it took a long time for it to catch on,” Hinton said.

The concept has indeed caught on, and as explained by the Google team that recently released the open source TensorFlow machine learning system, using the vector principle in NLP helps to address problems caused by methods that treat words as discrete atomic symbols. That discrete classification leads to data sparsity, and usually means more data may be needed to successfully train statistical models. Using vector representations can overcome some of these obstacles.

In introducing their explanation, the TensorFlow team cited image processing as a field that already used vectors of raw pixel intensities: that field is also one of the foremost examples of using “deep” neural networks, networks of multiple layers that learn from each other as data is passed between them, to improve accuracy and performance.

As vectors became more popular in NLP research, so too did the principles of deep learning within the field. However, orthodox deep learning approaches that may be very suitable for raw pixel intensities can prove problematic for text; as explained by data scientist Will Stanton in a presentation prepared for the 2015 machine learning “Ski Hackathon,” each hidden layer and each feature means more parameters to train, and hu-

man-generated text has a near-infinite number of features and data.

In 2013, however, a research team from Google led by Tomas Mikolov published word2vec, a three-layer model (input, hidden layer, and output layer) that vastly improved the speed of what had been the contemporary state of the art—by making the neural network shallower but wider.

“Shallow models can be trained using a bigger, wider, net on much more data, which can pay off in some cases much more than training a deeper net on a small subset of the training data, due to time constraints—the training can be very expensive,” Mikolov, now a research scientist at Facebook, said. For example, he said, one of the first well-known examples of a vectorized neural network contained 50 dimensions; that is, just 50 neurons were used.

“It took two months to train this model on approximately 600 million words,” he said. “In my papers, I analyzed the shallow net’s performance and on some tasks, going to 200–300 dimensionality helps—that is, the model is wider, and more precise; also, using the shallow model and an efficient implementation, I could train the word vectors with word2vec on a 100-billion-word dataset in hours.

“If you pre-train the vectors—convert words into distributed continuous vectors that capture in some sense the semantics of the words—on Wikipedia, that is several billions of words you just trained the model on. The resulting vectors are not good by themselves for anything concrete. Then, when you pick a task of, say, sentiment analysis, you can build a classifier that will take these pre-trained vectors as its input, instead of just the raw words, and perform classification. This is because labeled examples are much more expensive to obtain than the unlabeled ones. The resulting classifier can work much better when it is based on the pre-trained word features.”

Word2vec relies on two algorithms, one a “continuous bag of words,” a model trained to predict a missing word in a sentence based on the surrounding context; the other deemed “skip-gram,” which uses each current word as an input to a log-linear classifier to predict words within a certain range before and after that current word.

While Mikolov's flattening of the neural network concept appears on the surface to be a significant break from other approaches to NLP, Yoav Goldberg and Omer Levy, researchers at Bar-Ilan University in Ramat-Gan, Israel, have concluded much of the technique's power comes from tuning algorithmic elements such as dynamically sized context windows. Goldberg and Levy call those elements hyperparameters.

"The scientific community was comparing two implementations of the same idea, with one implementation—word2vec—consistently outperforming another, the 'traditional' distributional methods from the 90's," Levy said. "However, the community did not realize that these two implementations were in fact related, and attributed the difference in performance to something inherent in the algorithm."

"We showed that these two implementations are mathematically related, and that the main difference between them was this collection of 'hyperparameters'. Our controlled experiments showed that these hyperparameters are the main cause of improved performance, and not the count/predict nature of the different implementations."

Other researchers have released vectorization technologies with similar aims to word2vec's. For example, in 2014, Socher, then at Stanford University, and colleagues Jeffrey Pennington and Christopher D. Manning released Global Vectors for Word Representation (GloVe). The difference between GloVe and word2vec was summarized by Radim Rehurek, director of machine learning consultancy RaRe technologies, in a recent blog post:

"Basically, where GloVe precomputes the large word x word co-occurrence matrix in memory and then quickly factorizes it, word2vec sweeps through the sentences in an online fashion, handling each co-occurrence separately," Rehurek, who created the open source modeling toolkit gensim and optimized it for word2vec, wrote. "So, there is a trade-off between taking more memory (GloVe) vs. taking longer to train (word2vec)."

Machine learning specialists in industry have already taken to using general-purpose tools such as GloVe and

word2vec, but are not getting caught up in comparisons.

"There have definitely been some arguments about the kinds of results that have been presented, like the accuracy of GloVe vs. word2vec," said Samiur Rahman, senior machine learning engineer at MatterMark, a company specializing in document search for business news. "And then Levy and Goldberg have their own vectors too, but they're all essentially pretty good for general-purpose vectors. So instead of spending time trying to figure out which of the three works best for you, I would recommend—and it's worked well for us—choose the one that has the best production implementation right now, that fits easily into your workflow, and also figure out which one has better tools to train on data you have."

Rahman and others maintain word2vec, while very useful in initializing domain-specific NLP, complements but does not supplant other models. "We used word2vec to construct document vectors, because we didn't have a lot of labeled examples of what were funding articles and what weren't," Rahman said, adding that once a given model within a narrow domain has enough training data, a Naive Bayes-based model works well, with less computational complexity.

What's + Next + NLP = ?

Just as "Hello, World" may be the best-known general programming introductory example, Mikolov, who was then at Microsoft Research, also introduced what fast became a benchmark equation in natural language processing at the 2013 proceedings of the North American Association for Computational Linguistics, the *king-man+woman=queen* analogy, in which the computer solved the equation spontaneously.

"What was really fascinating about it was that nobody trained the computer to solve these analogies," Levy said. "It was a by-product of an unsupervised learning scheme. Word2vec shows it, but also a previous model of Mikolov's shows it as well. So you would train the computer to do language modeling, for example, or to complete the sentence and you would get vectors that exhibit word similarity like 'debate and discussion,' or 'dog and cat,' but nobody told

ACM Member News

RESOLVING MATH, CS ISSUES WITH DISTRIBUTED COMPUTING



Mathematics and problem-solving are two of Faith Ellen's lifelong passions.

A professor of computer science (CS) at the University of Toronto, Ellen has ample opportunity to use mathematical and problem-solving skills in her research involving distributed data structures and the theory of distributed computing. "I've always liked the fact that there were right and wrong answers to questions and problems," she says.

Ellen recalls an early affinity for CS, taking courses in theory and learning to program in high school. She received her bachelor's degree with honors in mathematics and computer science at the University of Waterloo, where she also received her master's degree in CS on formal CS programming languages; "I thought they were beautiful." She earned her doctorate from the University of California, Berkeley with a dissertation on "lower bounds for cycle detection and parallel prefix sums."

Her current research involves distributed computing and proving lower bounds on "the complexity of concrete problems to understand how the parameters of various models affect their computational power."

Among her proudest accomplishments was co-authoring "Impossibility Results for Distributed Computing" with Hagit Attiya, which was published as a book in 2014. The pair surveyed results from various distributed computing models proving tasks to be impossible, either outright or within given resource bounds. "Lower bounds are fun. I like to prove that it's impossible to do something faster or that it's impossible to do something without using lots of storage space."

—Laura DiDio

it anything about analogies, and the fact these analogies emerged spontaneously was amazing. I think it's the only case where we use 'magic' in a science publication because it looked like magic."

It was not magic, of course, but the principle behind it allows the concept of vectorization to be made very clear to those far outside the NLP and machine learning communities. As data scientist Chris Moody explained, also at the 2015 Text By The Bay conference, the gender-indicating vectors for king and queen will be the same length and angle as those for woman and man, aunt and uncle, and daughter and son; in fact, any conceptual group at all, such as different languages' words for the same animal, or the relationship of countries and their capital cities, can be shown to have similar properties that can be represented by similar vectors—a very understandable universality.

"That's the most exciting thing, lighting up that spark," he told *Communications*. "When people say, 'oh, you mean computers understand text? Even at a rudimentary level? What can we do with that?' And then I think follows an explosion of ideas."

Moody works for online fashion merchant Stitch Fix, which uses analysis of detailed customer feedback in tandem with human stylists' judgments to supply its clients with highly personalized apparel. The Stitch Fix experience, Moody said, is not like typical online shopping.

"Amazon sells about 30% of their things through personalized recommendations—'People like you bought this'—and Netflix sells or rents out 70% of their viewings through those kinds of recommendations. But we sell everything through this personalized service. The website is very minimal. There's no searching, no inventory, no way for you to say 'I want to buy item 32.' There is no fallback; we have to get this right. So for us, being on the leading edge of NLP is a critical differentiating factor."

The combination of the company's catalog and user feedback—for example, a certain garment's catalog number and the word "pregnant" and words that also denote pregnancy or some sort of early-child-rearing status, located near each other in the Stitch

"Most of our reasoning is by analogy; it's not logical reasoning."

Fix algorithm's vector space—can help guide a stylist to supply a customer with a garment similar in style to the original, but cut for maternity wear. What is more, he said, the word2vec algorithm as used by Stitch Fix in production is not used on text.

"In our boxes we ship five items. and you can use word2vec here and say 'given this item in that box, can you predict the other items?'" Moody said. "Word2vec doesn't care if it's a word or not a word, it's just another token and you have to make that token similar to the other tokens. It's almost like using the backbone of the word2vec algorithm to look inside someone's closet and saying 'these things are very similar because they would all appear in the same sort of closet together.'"

In fact, he said, the company is starting to use analogical principles to go beyond text and synthesize the images of imagined new items from images of existing pieces of clothing—a process he said was "starting to get toward this hint of creativity. So if you think of these word vectors like *king-man+woman=queen*, we're now exploring spaces between those data points, and that's what we're calling creativity—things that have never been seen before, but are really just somewhere in between all those other observations."

How quickly that sort of creativity may lead to breakthroughs for machine learning and artificial intelligence is clearly an open question, but it bears mulling, given an observation about the basis of human reasoning from Hinton.

"Most of our reasoning is by analogy; it's not logical reasoning," he said. "The early AI guys thought we had to use logic as a model and so they couldn't cope with reasoning by analogy. The honest ones, like Allen New-

ell, realized that reasoning by analogy was a huge problem for them, but they weren't willing to say that reasoning by analogy is the core kind of reasoning we do, and logic is just a sort of superficial thing on top of it that happens much later." 

Further Reading

Levy, O. and Goldberg, Y., Linguistic Regularities in Sparse and Explicit Word Representations. *Proceedings of the 18th Conference on Computational Natural Language Learning*, 2014. <http://bit.ly/10XBici>

Mikolov, T., Chen, K., Corrado, G., and Dean, J., Efficient Estimation of Word Representations in Vector Space. *Proceedings of Workshop at International Conference on Learning Representations*, 2013. <http://arxiv.org/abs/1301.3781>

Goldberg, Y., and Levy, O., word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method, *arXiv* 2014. <http://arxiv.org/abs/1402.3722>

Pennington, J., Socher, R., and Manning, C., GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. <http://nlp.stanford.edu/projects/glove/>

Moody, C., A Word is Worth a Thousand Vectors. *MultiThreaded, StitchFix*, 11 March 2015. <http://bit.ly/1NL35xz>

Iyyer, M., Boyd-Graber, J., Claudino, L., Socher, R., and Daume III, H., A Neural Network For Factoid Question Answering Over Paragraphs. *Proceedings of EMNLP 2014* <https://cs.umd.edu/~miyyer/qblearn/>

Video resources

Hinton, G., Deep Learning. *Royal Society keynote, recorded 22 May 2015*. <https://www.youtube.com/watch?v=IcOMKXAw5VA>

Socher, R., Deep Learning for Natural Language Processing. *Text By The Bay 2015*. <https://www.youtube.com/watch?v=tdLmf8t4oqM>

Bob Dylan and IBM Watson on Language, advertisement, 5 October 2015. <https://www.youtube.com/watch?v=pwh1INne97Q>

Gregory Goth is an Oakville, CT-based writer who specializes in science and technology.

© 2016 ACM 0001-0782/16/03 \$15.00

Rich Data, Poor Fields

Diverse technologies help farmers produce food in resource-poor areas.

IN A WORLD with more mobile phones than flush toilets, digital devices are now standard equipment among even the world's poorest and most remote people. Farmers in these areas are getting tools for their devices that help deliver water, nutrients, and medicine to plants as needed; test for crop diseases and malnourishment; and survey their soil for future planning. In some cases, these emerging apps are the biggest new technologies resource-poor farms have seen in hundreds of years.

That is not very surprising to Rajiv “Raj” Khosla, professor of Precision Agriculture at the College of Agricultural Sciences of Colorado State University. “What we’re finding is that many small-scale farmers in resource-poor environments are still farming in the 1500s. They’re looking for leapfrog technologies,” he said.

Founder and past president of the International Society of Precision Agriculture, Khosla described how data and modern techniques can multiply yields through small changes. “I was involved in a study in the northwestern part of India where a farmer had two acres of land,” he said. “They divided it into two halves. They precision-leveled one side [using GPS] and they did conventional leveling of the other [using a wooden plank and oxen]. On the precision-leveled side, they also sampled the soil for balanced nutrition, and instead of flooding the entire field when water was available, they only flooded it every other time, so using 50% less water. Same crop, same variety, same size, and guess what? On the “conventional” side, the farmer produced 800 kilos of wheat; on the other half, they harvested 2,250 kg.”

Precision agriculture also guides farm machinery based on satellite images, GPS information, or local surveys. John Nowatzki, agricultural machine systems specialist at North



A prototype of a portable assay device connected to a smartphone; such equipment could help remote farmers diagnose crop diseases without access to a full lab.

Dakota State University, described how precision “side-dressing” of fertilizer benefits farmers over the traditional method of blanketing the entire field. “First, farmers lower their costs by being more efficient in their use of their fertilizer. Second, you could put a no-spray zone in over an ecologically sensitive area, such as a waterway, so it’s an environmentally friendly technology as well.”

From the Soil to the Sky

Yield improvements mean a lot locally to the individual farmer. However, such improvements come from a global effort that incorporates both satellite data and decades of on-the-ground surveys.

One example is soil research, which guides individual farmers on what to grow and how to treat the soil. “Soil surveys can and should be used for agricultural, land development, and

conservation decisions affecting more than a couple of acres” said Dylan Beaudette, a soil scientist for the National Resources Conservation Service (NRCS), which makes U.S. soil conditions available via apps, websites, and a JavaScript Object Notation (JSON) interface. “They’ve gone from loosely coordinated local efforts to nationwide scientific investigations,” he said. “These changes reflect our continuously changing understanding of—and expectations from—soil research.”

The International Soil Reference and Information Centre (ISRIC) in the Netherlands manages similar data worldwide, recently producing soil maps for the Africa Soil Information Service (AfSIS). ISRIC’s data for this project was predictive, applying a “random forest data mining algorithm on the input data to achieve 250-meter resolution.

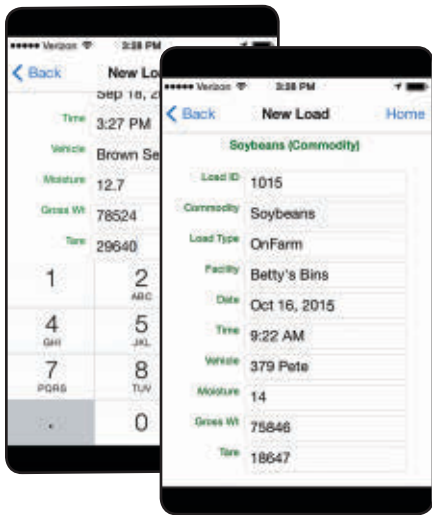
“We discovered that machine learning is well suited for soil data,” ISRIC senior researcher Tomislav Hengl said. “Most likely this is because we model the soil variable as a function of climate, relief, geology, and vegetation dynamics. In most cases, this relationship is nonlinear, so when you use machine learning techniques, you have a better chance to discover the nonlinear relationships.”

The Plant Itself

Once a plant pushes its way above the soil, farmers can scan it using hyperspectral imaging, which examines its appearance for disease and malnutrition under several specific wavelengths of light. According to Simon Holland, CEO of agricultural technology company Barefoot Lightning, the technique works well from a variety of vantage points. “You’ve got your satellites looking from a great distance, with maybe one- to three-meter resolution on the ground,” he said, “and you’ve got a drone with similar equipment doing nutrient analysis and looking for two or three disease signals. Then on the

ground, there are devices that can recognize a disease because the chlorophyll signal's gone off track. But they're still doing a point analysis, not a full image analysis of a leaf or any other structure."

For a closer look, Holland's company has partnered with the University of Manchester's e-Agri Sensors Centre to develop a handheld hyperspectral imager. Besides providing diagnoses unavailable to cameras far above the crops, such devices could also be far less expensive. Centre director Bruce Grieve explained, "Cameras in an aircraft or satellite are high-cost because the only



MapShot Inc.'s free Harvest Entry mobile app, which gives farmers a way to record harvest data.

light available to them is sunlight. You need a sensitive detector because the system filters out 99.9% of the energy to keep only the very narrow bands that you're trying to get," he said. "But if you go close up, you can do narrow-band illumination using LEDs, and read the signals with a standard CMOS detector that might be in a mobile phone."

Those images are compared against disease libraries held in the phone for a diagnosis; then, said Holland, "If we come up with something we don't recognize, we could send a sample to a lab for more-detailed spectra and to do a pathology assessment on the scanner's 'big brother' device. That way, you can also start doing a big-data collection for the disease database."

Even the process of collecting those samples can be challenging in low-resource areas, where transportation and temperature control are harder to come by. One simple solution is

found in the lab of Ratmir Derda, assistant professor in the Department of Chemistry at the University of Alberta: a sort of paper Petri dish. "Even in North America, you can be a half-day's travel from the nearest lab. In the field you don't have incubators, or an aseptic environment, or other things that you'd find in a clinical laboratory. So we're trying to package the sample in a device that's aseptically sealed, that can be easily inoculated with a sample, and is lightweight enough to maintain the desired temperature for transport or growth without much energy."

A somewhat more technical solution comes from the University of Wisconsin, where associate professor of biochemistry Douglas Weibel has developed a "cartridge" containing all the chemistry and fluid-handling capabilities needed to perform in-field analyses, delivering results to a mobile phone via a USB cable connected to a small electronic reader. "To test a cassava crop for five to 10 different viruses, we create a 'cassava chip' in which all of the reagents are pre-loaded on the small cartridge, which is vacuum-sealed in a disposable Mylar bag," he said. "Your phone guides you through all the steps. You open the bag, take the chip out, suck up a small amount of sample, and put the chip in the reader, which heats the assays up for 15 minutes. The reader performs a one-wavelength scan with inexpensive LEDs and a low-cost CMOS camera. It could be multispectral; however, the goal was to keep it simple."

Food and Finance

These technologies are only helpful if low-resource farmers can overcome barriers of cost and location to access them. Weibel hopes eventually to bring his assay readers to developing countries for under \$50, with the price on specific chips below 50 cents. However, he says, that means approaching a larger, higher-margin market first. "We have a two-stage model," said Weibel. "The first stage is commercialization in a place like North America that can drive research and development to create the platform. Then in the second stage, once we have scalability figured out, we adapt the technology to problems in sub-Saharan Africa. Those problems are likely to be quite different from the agricultural challenges here.

Economics also plays an important role. In the U.S., it is possible to eradicate an entire crop to prevent spread, however this strategy would bankrupt farmers in developing countries."

The \$50 cost for the assay reader could be out of reach in these areas, particularly to "smallholder" farmers, Weibel said, "But the beauty is that a technology like this could stimulate a business model whereby someone invests money in buying one of these test systems, and provides the route to distribution by performing tests for farmers at a low cost."

Khosla pointed out farm size would also encourage such a "service provider" model. "I might have 10 farmers come to my field day [in Colorado], each with 2,000–10,000 acres they're farming, for a total of 50,000 acres," he said, "but 50,000 acres in south Asia means thousands of farmers!"

Developing a network of providers to service all those farmers is likely to be at least as big a challenge as perfecting the technology. As the technologies improve—and costs drop—some of the tools will eventually come into individual farmers' hands. Speaking of his hyperspectral imaging device, Grieve said, "We're looking at actually putting it in peoples' pockets, so a farmer who wants to check some feature on a crop can just pop the thing in and see what's actually happening." **C**

Further Reading

Pongnumkul, S., Chaavalit, P., and Surasvadi, N. (2015).

Applications of Smartphone-Based Sensors in Agriculture: A Systematic Review of Research. *Journal of Sensors*, 2015.

Hengl, T., Heuvelink, G. B., Kempen, B., Leenaars, J. G., Walsh, M. G., Shepherd, K. D., and Tondoh, J. E. (2015).

Mapping soil properties of Africa at 250 m resolution: Random forests significantly improve current predictions. *PLoS one*, 10(6), e0125814.

Derda, R., Gitaka, J., Klapperich, C. M., Mace, C. R., Kumar, A. A., Lieberman, M., and Yager, P. (2015).

Enabling the Development and Deployment of Next Generation Point-of-Care Diagnostics, *PLOS Neglected Tropical Diseases*, <http://bit.ly/1PnG40F>. z.

Tom Geller is an Oberlin, Ohio-based writer and documentary producer.

When Computers Stand in the Schoolhouse Door

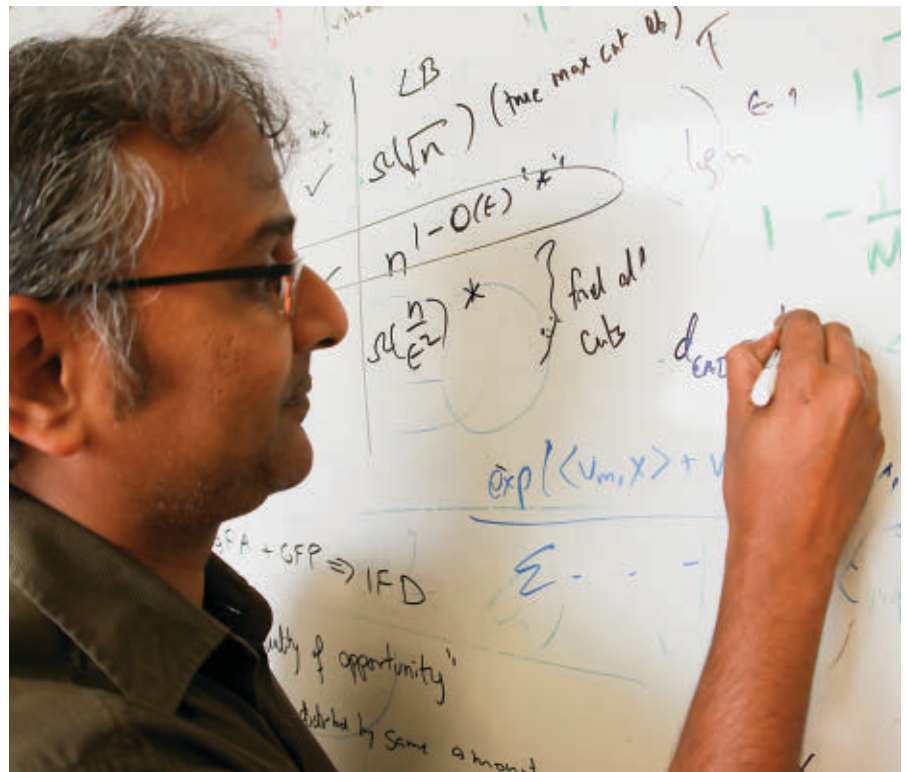
Classification algorithms can lead to biased decisions, so researchers are trying to identify such biases and root them out.

IF YOU HAVE ever searched for hotel rooms online, you have probably had this experience: surf over to another website to read a news story and the page fills up with ads for travel sites, offering deals on hotel rooms in the city you plan to visit. Buy something on Amazon, and ads for similar products will follow you around the Web. The practice of profiling people online means companies get more value from their advertising dollars and users are more likely to see ads that interest them.

The practice has a downside, though, when the profiling is based on sensitive attributes, such as race, sex, or sexual orientation. Algorithms that sort people by such categories risk introducing discrimination, and if they negatively affect a protected group's access to jobs, housing, or credit, they may run afoul of antidiscrimination laws. That is a growing concern as computer programs are increasingly used to help make decisions about who gets a credit card, which résumés lead to job interviews, or whether someone gets into a particular college. Even when the programs do not lead to illegal discrimination, they may still create or reinforce biases.

Computer scientists and public policy experts are beginning to pay more attention to bias in algorithms, to determine where it is showing up and what ought to be done about it. "Certainly there's a pretty hardy conversation that's begun in the research community," says Deirdre Mulligan, co-director of the Center for Law and Technology at the University of California, Berkeley.

One way algorithms may discriminate is in deciding who should see



Suresh Venkatasubramanian of the University of Utah presented a method for finding disparate impact in algorithms last year at the ACM Conference on Knowledge Discovery and Data Mining.

particular job-related ads. Anupam Datta, a professor of computer science at Carnegie Mellon University in Pittsburgh, PA, created AdFisher (<http://bit.ly/1IRhF6P>), a program that simulates browsing behavior and collects information about the ads returned after Google searches. Datta and his colleagues created 1,000 fake users and told Google half of them were men and half were women. Using AdFisher, each of the simulated users visited websites related to employment, then collected data about which ads they were shown subsequently. The tool discovered more ads related to higher-

paying jobs were served to men than were presented to women.

In particular, the top two ads shown to men were for a career-coaching service for people seeking executive positions that paid upward of \$200,000 per year. Google showed that ad 1,852 times to the male group, but only 318 times to the female group. The top two ads shown to women were for a generic job-posting service and an auto dealer. "Ads for employment are a gateway to opportunity," Datta says. "If you don't make [job-seekers] aware of opportunities, you might be reinforcing biases."

The source of the differentiation be-

tween ads is not entirely clear. It could be the advertisers specified groups they wanted to target their ads toward. Waffles Pi Natusch, president of the Barrett Group executive coaching firm, told the *Pittsburgh Post-Gazette* last year the company does not specifically target men, but does seek out clients who have executive-level experience, are older than 45, and already earn more than \$100,000 a year. Datta says there may be some correlation between those preferences and a person's gender.

How much the advertiser was willing to spend on the ad may have played a role as well. Google's algorithm presents advertisers with profiles of users and allows them to bid for placement on pages seen by those users. If a job ad paid the same whether it was targeted toward a male or a female user, but a clothing ad was willing to pay a premium to be seen by a woman, it could be that the job ad got outbid in the women's feed but not the men's feed.

It is also possible, Datta says, that Google's algorithm simply generated more interest for a particular ad from one group. "If they saw more males

"Your perception of the gender balance of an occupation matters not only to how you hire, how you recruit, but it also affects the choice of people who go into the profession."

were clicking on this ad than females, they may have decided to serve more of these ads to male viewers," he says.

Google would not discuss the issue beyond offering the following official statement: "Advertisers can choose to target the audience they want to reach, and we have policies that guide the

type of interest-based ads that are allowed. We provide transparency to users with 'Why This Ad' notices and Ad Settings, as well as the ability to opt out of interest-based ads."

One of the challenges, researchers say, is that many of the datasets and algorithms used for classification tasks are proprietary, making it difficult to pinpoint where exactly the biases may reside. "The starting point of this work was the observation that many important decisions these days are being made inside black boxes," Datta says. "We don't have a very good sense of what types of personal information they're using to make decisions."

Look in the Mirror

Some imbalance may come from users' own biases. Sean Munson, a professor at the University of Washington's Department of Human Centered Design and Engineering in Seattle, WA, looked at the results returned by searches for images representing different jobs. In jobs that were more stereotypically male, there was a higher proportion of men in the search results than there



We're seeking the
best new minds to
encrypt the future
of **cyber security**.

The University of Central Florida's Faculty Cluster Initiative fosters the development of talented, interdisciplinary teams focused on solving today's toughest scientific and societal challenges through teaching and research. We're hiring new faculty members in the area of cyber security and privacy to advance UCF's unique areas of excellence and global impact.

ucf.edu/research/cyber

UCF is an equal opportunity/affirmative action employer.

was in that profession in reality, while women were underrepresented. What is more, when users were asked to rate the quality of the results, they were happier with images where the gender shown matched the stereotype of the occupation—male construction workers or female nurses, say.

Some of the imbalance probably comes from which images are available. “We also play a role when we click on things in image result sets,” Munson says. “My personal belief—and not knowing the Google algorithm—is that it’s just reflecting our own biases back at us.”

While it is unlikely image searches would violate anti-discrimination laws, Munson says skewed results could still have negative consequences. “Your perception of the gender balance of an occupation matters not only to how you hire, how you recruit, but it also affects the choice of people to go into the profession,” he says.

Other algorithms, though, may run afoul of the law. A credit-scoring algorithm that winds up recommending against borrowers based on their race, whether purposefully or not, would be a problem, for instance. Anti-discrimination law uses the concept of adverse (or disparate) impact to avoid having to prove intent to discriminate; if a policy or procedure can be shown to have a disproportionately negative impact on people in a protected class or group, it will be considered discriminatory.

Joseph Domingo-Ferrer and Sara Hajian, computer scientists at Rovira i Virgili University in Tarragona, Spain, have developed a method for preventing such discrimination in data mining applications that might be used to assess credit worthiness.

One obvious approach might be to simply remove any references to race from the training data supplied to a machine learning algorithm, but that can be ineffective. “There might be other attributes which are highly correlated with this one,” Hajian says. For instance, a certain neighborhood may be predominantly black, and if the algorithm winds up tagging anyone from that neighborhood a credit risk, the effect is the same as if it had used race. In addition, transforming the data too much by removing such attributes

“Ethical issues can be integrated with data mining and machine learning without losing a lot of data utility.”

might lead to less-accurate results.

What the researchers do instead is let the algorithm develop its classification rules, then examine its results. Using legal definitions of discrimination and protected classes, they examine the algorithm to see which rules led to the unwanted decisions; they then can decrease the number of records in the data that supports those rules. That way, Hajian says, they can transform the data enough to remove the discriminatory results while still preserving its usefulness.

“Ethical issues can be integrated with data mining and machine learning without losing a lot of data utility,” she says.

At the ACM Conference on Knowledge Discovery and Data Mining in Sydney, Australia in August 2015, Suresh Venkatasubramanian, a computer scientist in the School of Computing at the University of Utah, presented a method for finding disparate impact in algorithms. Like Hajian and Domingo-Ferrer, Venkatasubramanian checks on the classification algorithm by examining its results, which does not require him to look at any proprietary code or data. If he can look at the decisions the algorithm made and use those to accurately infer protected attributes, such as race or sex, in the dataset, that means the algorithm has produced a disparate impact. He also offers a method, similar to the other researchers’, that lets the data be transformed in a minimal way, to eliminate the bias while preserving the utility.

This type of approach only works when there is a clear legal standard to define bias. Other cases, such as image search results, require developing

a societal consensus on whether there is a problem and what should be done about it.

“Some of it is a technological challenge, a computer science challenge, but some of it is a challenge of how these algorithms should operate in a larger social context,” Datta says.

Should a company like Google be liable if it shows ads from job-coaching services more to men than to women, or does that not rise to the level of actual job discrimination? Is the company just a neutral platform delivering information? Should it tweak the algorithm to deliver the ads more proportionally, and what if that causes it to lose money because of lower click-through rates?

“There is a cost to trying to enforce fairness, and someone has to bear that cost,” Datta says.

Not every instance of differentiation is discrimination, says Mulligan. Few people, for instance, object because women’s magazines contain advertisements targeted specifically at women.

“Any system of classification has a bias. That’s actually what makes it useful. It’s curating. It’s how it helps you sort stuff,” Datta says.

“The question becomes, what sort of unfairness do we want to avoid?” ■

Further Reading

- Hajian, S., and Domingo-Ferrer, J. *A Methodology for Direct and Indirect Discrimination Prevention in Data Mining*, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 25, No. 7 (2013)
- Kay, M., Matuszek, C., and Munson, S.A. *Unequal Representation and Gender Stereotypes in Image Search Results for Occupations*, *CHI 2015*, Seoul, Korea (2015)
- Datta, A., Tschantz, M.C., and Datta, A. *Automated Experiments on Ad Privacy Settings*, *Proceedings on Privacy Enhancing Technologies*, 1, (2015)
- Dwork, C., and Mulligan, D. K. *It’s Not Privacy, and it’s Not Fair*, *Stanford Law Review Online*, Vol. 66, No. 35, (2013)
- Venkatasubramanian, S. *Certifying and Removing Disparate Impact*, <https://www.youtube.com/watch?v=4ds9fBDtMmU>

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

© 2016 ACM 0001-0782/16/03 \$15.00



Peter Naur: 1928–2016

Peter Naur, a Danish computer scientist and 2005 recipient of the ACM A.M. Turing Award, died recently after a brief illness.

PETER NAUR, A Danish computer scientist and 2005 recipient of the ACM A.M. Turing Award for fundamental contributions to programming language design and the definition of Algol 60, to compiler design, and to the art and practice of computer programming, died January 3 after a brief illness.

Born in Fredricksberg, Denmark, Naur studied astronomy at the University of Copenhagen, where he received his Ph.D. in that field before going to Kings College Cambridge in the 1950s to conduct research both into astronomy and the emerging field of computer programming. As he told *Computerworld Denmark* in a 2014 interview (<http://bit.ly/1O13v1I>), “I had the great privilege to get to Cambridge in the early 1950s. Here I discovered that calculations of planetary motion that could take several hours, could now be carried out in seconds with a computer.”

As a result, Naur changed his career focus to computer science. From 1959 to 1969, he worked at Denmark’s Regnecentralen computing institute (now known as the Danish Institute of Mathematical Machines), while he was also lecturing at the Niels Bohr Institute and the Technical University of Denmark.

At the Regnecentralen, Naur participated in the development of a programming language that came to be known as Algol 60. He became the main author of the Report on the Algorithmic Language ALGOL 60 (http://web.eecs.umich.edu/~bchandra/courses/papers/Naure_Algol60.pdf) and made the decision, which was controversial at the time, to include recursion.

In 1969, Naur was appointed professor at the Institute of Datalogi at Copenhagen University, a post he held until his retirement in 1999 at

“Computing presents us a form of description ... very useful for describing a great variety of phenomena of this world, but human thinking is not one of them.”

the age of 70. His main areas of inquiry were design, structure, and performance of computer programs and algorithms.

In his book *Computing: A Human Activity* (1992), Naur rejected the formalist view of programming as a branch of mathematics. He did not like being associated with the Backus-Naur Form (a notation technique for context-free grammars attributed to him by Donald Knuth), and said he would prefer it to be called the Backus Normal Form.

Naur disliked the term “computer science,” suggesting it instead be called “datalogy” or “data science” (“datalogy” has been adopted in Denmark and in Sweden as *datalogi*, while “data science” is used to refer to data analysis (as in statistics and databases).

In his 2005 ACM A.M. Turing Award Lecture (the full lecture is available for viewing at http://amturing.acm.org/vp/naur_1024454.cfm), Naur offered a 50-year retrospective of computing vs. human thinking. He concluded, “Computing presents us a form of

description ... very useful for describing a great variety of phenomena of this world, but human thinking is not one of them, the reason being that human thinking basically is a matter of the plasticity of the elements of the nervous system, while computers—Turing machines—have no plastic elements. For describing human thinking one needs a very different, non-digital form, as demonstrated by the Synapse-State Theory” (Naur’s theory documented in a February 2004 paper that defines “mental life” in terms of neural activity).

Google vice president and Chief Internet Evangelist Vint Cerf said, “I did not know Peter well but I was a beneficiary of his work. For many years, I made heavy use of the Backus-Naur Form to describe the syntax of languages and other constructs such as email formats. Peter’s insight, as I see it, was to provide us with a thoroughly general and regular way in which to express the structure of string-based objects. Conversion of this description into parsing programs that allowed applications to process these objects was made straightforward by simplicity of the BNF form.”

Cerf, a former ACM president, added, “In later years, I would meet Peter at the newly created Heidelberg Laureate Forum and enjoyed so much his ideas, especially about the way in which the brain operates. He seemed to be onto some very intriguing notions and I am sorry to learn that he has passed away before he could pursue them more fully.

“The field of computer science benefited greatly from Peter’s contributions and his enthusiasm for the field. I will miss our annual meetings and will remember him with respect and admiration.”





DOI:10.1145/2879643

Pamela Samuelson

Legally Speaking

New Exemptions to Anti-Circumvention Rules

Allowing some reverse engineering of technical measures for non-infringing purposes.

EVERY THREE YEARS, the U.S. Copyright Office opens a new inquiry into whether the law that bars circumvention of technical protection measures (TPMs) used by copyright owners to protect access to their works are impeding lawful uses of the works. Those who propose exemptions to the anti-circumvention rules must offer substantial evidence that TPMs are thwarting lawful uses.

After examining this evidence, the Copyright Office recommends to the Librarian of Congress that he should grant certain specific exemptions for the ensuing three years. A new application will be necessary in the next rule-making cycle to extend the exemption beyond the three-year term.

My July 2015 *Communications Legally Speaking* column offered an overview of the exemptions being sought during the current triennial review. It also explained why I thought some would be granted, some would be narrowed, and some would be denied.

In late October 2015, the Librarian issued a motley set of exemptions

that largely bore out my predictions, although the exemptions granted were more numerous and generous to circumventors (that is, reverse engineers) than in previous triennial reviews.

After offering some general observations about the rule-making process, this column focuses on the anti-circumvention exemptions of greatest interest to computing professionals: those that enable unlocking of all-purpose mobile devices, those that allow jailbreaking that will enable owners of devices to run software that would otherwise not be available on their devices, and those that affect computer security testing.

Three General Observations

First, the Office has now accepted two propositions that years ago might have been almost unthinkable: that, as critics of the anti-circumvention rules have been saying since 1998, TPMs impede many lawful activities and many circumventions of TPMs enable legitimate non-infringing acts.

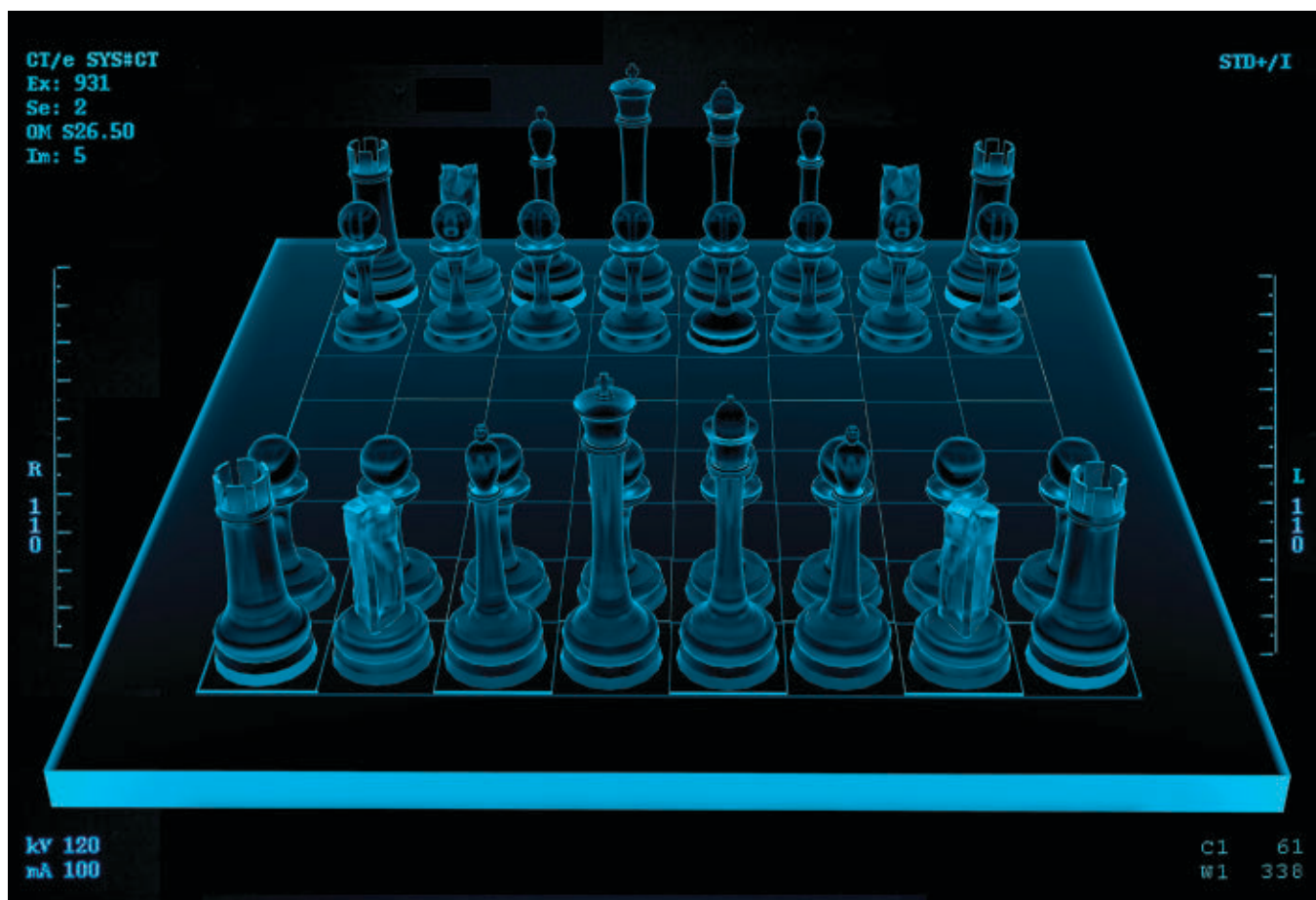
For each exemption granted, the Office had to admit the specific use to

be enabled was either fair use or otherwise privileged under U.S. copyright law. So the Office has developed an official record of lawful uses enabled by reverse engineering of TPMs. This is a step in the right direction.

Second, the Office has recognized the ubiquity of software embedded in a wide range of consumer products, such as automobiles and medical devices, means the anti-circumvention rules now arguably have implications far beyond the anti-piracy purposes that drove adoption of the rules back in 1998.

Yet, rather than saying, as perhaps it should have, the anti-circumvention rules have no application to, say, farmers who want to reverse engineer the software in their tractors to repair or modify them, the Office has implicitly accepted the anti-circumvention rules do apply to these acts. It has, however, provided an exemption that enables some reverse engineering of these vehicles.

Under the new exemption, farmers and other owners of motor vehicles can reverse engineer software to repair



or modify the vehicles themselves, but they cannot hire someone to do it for them. The Office insists new legislation would be required to enable third-party help to fix vehicle software. This substantially limits the utility of the granted exemption.

Third, the Office has suspended some granted exemptions—for reverse engineering of motor vehicles, medical devices, and some security testing—for 12 months to allow the U.S. Department of Transportation (DOT), the Environmental Protection Agency (EPA), and the Food and Drug Administration (FDA) to consider whether such reverse engineering should be permissible given health, safety, or environmental concerns raised by exemption opponents.

This suspension effectively deprives successful applicants of the benefits of the granted exemptions for a full third of their durations. It also subjects them to the risk that if they engage in the acts described in their applications during the first 12 months, they will open themselves to anti-circumvention liability, even though the Office accepted

the lawfulness of the uses in granting the exemption.

In addition, these reverse engineers will now be subject to new rounds of scrutiny by the DOT, EPA, and FDA. And if these agencies object, the anti-circumvention exemptions granted in 2015 may never go into effect.

Unlocking and Jailbreaking Exemptions

Five sets of exemption requests focused on unlocking various types of information technology devices: cellphones, tablet computers, portable mobile connectivity devices, wearable wireless devices, and smart devices. These exemptions were sought to enable users to connect to their preferred wireless providers, to improve the resale value of the devices, and to avoid harm to the environment by encouraging disposal of devices rather than reuse of them.

In past rulemakings, exemptions permitted unlocking only with regard to cellphones. Congress instructed the Librarian to consider in this latest review process whether to include other devices as well.

The Office supported the first four requested exemptions because they enabled non-infringing uses of the devices TPMs were impeding. But the Office limited the scope of the exemptions to “used devices.” This means bypassing the TPMs to connect to a preferred wireless carrier is only exempt if the device had been lawfully acquired and activated to a network. This exemption did not extend to devices embedded in motor vehicles. The Office denied the smart devices exemption request on the ground it was too vague in its definition and scope.

Other exemption requests focused on “jailbreaking,” that is, bypassing TPMs to access operating system software for the purpose of enabling the owner of the device to install and execute software that could otherwise not be run on that OS. (Think of this as allowing users to get apps for their iPhones or iPads from a source other than the Apple App Store.) The exemption also allows removal of unwanted preinstalled software from the device.

The granted exemption was, however, not so broad as to enable jail-

breaking of dedicated devices, such as e-book readers or laptops. Yet, it did extend to all-purpose mobile devices, such as phones and tablets. Also granted was a similar exemption allowing the jailbreaking of smart TVs.

Security Research

The Copyright Office is now on record that the anti-circumvention rules have had a chilling effect on good faith computer security testing. The existing statutory exemption for such testing is, the Office has recognized, unduly narrow. Especially in this day and age when cybersecurity risks are so evident, further breathing space for good faith security testing is much needed.

The Office was not, however, willing to support as broad an exemption for such testing as some computing professionals had sought. The Office pointed out that submissions in support of security testing exemptions focused on testing of consumer-facing products, such as motor vehicles, e-voting technologies, and software-enabled medical devices, not large-scale industrial or governmental systems. The exemption granted was tailored to allow testing of consumer-facing products.

As noted earlier in this column, this exemption was suspended for 12 months so other agencies concerned with these devices could consider what if any further limits should be imposed on security testing.

Yet, the suspension did not apply to e-voting technologies. The Office was persuaded there were no public safety issues posed by this exemption to justify a delay in its implementation. Given the upcoming U.S. presidential election, we should be glad that good faith security researchers will be free to investigate whether some malefactors are tampering with software that might throw that election.

The Office expressed concern that the security testing should be conducted in controlled environments designed to ensure individuals and the public will not be harmed. The FDA insisted on a limitation to the medical device exemption to exclude systems that were being used or could be used by patients. The Office also limited the exemption for circumventing TPMs to get access to patient data being collected by the software.

Most applicants for exemptions got something for their troubles, even if not as extensive an exemption as requested.

Other Exemptions

Very few of the proposed anti-circumvention exemptions were rejected outright, although some were. As predicted in my July 2015 *Communications* column, the proposed exemption to allow backup copying and format shifting of DVD movies fell flat. But most applicants for exemptions got something for their troubles, even if not as extensive an exemption as requested.

As for bypassing TPMs for noncommercial, documentary, or nonprofit educational purposes, for instance, the Office is now willing to say that certain bypassing of TPMs protecting Blu-ray discs and online streaming services, as well as DVDs, should be exempt.

But the Authors Alliance plea for a broad multimedia e-book exemption was denied. Film studies professors are the main beneficiaries of the new exemption. So I, as a law professor, run the risk of anti-circumvention liability if, for example, I make an e-book with clips from movies portraying different versions of James Bond so my students can consider whether Bond should be a copyright-protectable character.

Other granted exemptions included one to allow bypassing TPMs to develop assistive technologies for print-disabled persons to provide access to contents of literary works distributed electronically and another to provide a narrow privilege to provide alternative feedstock for 3D printing.

Also exempt is bypassing TPMs by libraries, museums, and archives to preserve video games when the games'

developers have ceased to provide necessary remote server support. The Office even recognized the legitimacy of a user's interest in being able to continue playing videogames for which outside support had been discontinued.

Conclusion

This synopsis of the 2015 anti-circumvention rule is no substitute for reading the original. The final rule, along with submissions in support of and opposition to exemptions during the triennial review and other relevant materials, can be found at <http://copyright.gov/1201>.

Be forewarned, though, that the final rule is a dense 21 pages long, and like the Copyright Act of 1976, it is not exactly an easy read. For those computing professionals who engage in reverse engineering that involves some bypassing of TPMs for non-infringing purposes, the rule contains mostly good news. Yet, a close read (and possibly some legal advice) may be needed before computing professionals can feel completely safe relying on a granted exemption.

Yet to be addressed in the case law or the policy arena is how strict the courts will or should be in reading the exemptions recently granted. Under a strict reading, only those who have applied for and been granted explicit exemptions are relieved from circumvention liability. Any straying beyond the prescribed borders of the exemptions, even to engage in similar non-infringing activities, may seem dangerous.

However, there is some case law to suggest bypassing TPMs for non-infringing purposes does not violate the anti-circumvention rules, even if there is no applicable exemption. Moreover, lawsuits against non-infringing reverse engineers seem unlikely because courts will be unsympathetic to claims that merely bypassing a TPM to engage in legitimate activities is illegal. Still, the risk averse may understandably be unwilling to offer themselves up to be the test case for this proposition. 

Pamela Samuelson (pam@law.berkeley.edu) is the Richard M. Sherman Distinguished Professor of Law and Information at the University of California, Berkeley.

Copyright held by author.

Computing Ethics

The Question of Information Justice

Information justice is both a business concern and a moral question.

“ONCE THE ROCKETS go up, who cares where they come down; That’s not my department,” says Wernher von Braun.
—Tom Lehrer

In the 1990s, the government of India began a program to digitize and open land records, one rooted in what open data proponents tout as its chief virtue: “The Internet is the public space of the modern world, and through it governments now have the opportunity to better understand the needs of their citizens and citizens may participate more fully in their government. Information becomes more valuable as it is shared, less valuable as it is hoarded. Open data promotes increased civil discourse, improved public welfare, and a more efficient use of public resources.”^a

Digitizing the Record of Rights, Tenants, and Crops (RTC) along with demographic and spatial data was intended to empower citizens against state bureaucracies and corrupt officials through transparency and accountability. Sunshine would be the best disinfectant, securing citizens’ land claims against conflicting re-



cords. In fact, what happened was anything but democratic. The claims of the lowest classes of Indian society were completely excluded from the records, leading to the loss of their historic land tenancies to groups better able to support their land claims within the process defined by the data systems. Far from empowering the least well off, the digitization program reinforced the power of bureaucracies, public officials, and developers.

This case illustrates an underappreciated challenge in data science:

creating systems that promote justice. Data systems, similar to von Braun’s rockets, are too often assumed to be value-neutral representations of fact that produce justice and social welfare as an inevitable by-product of efficiency and openness. Rarely are questions raised about how they affect the position of individuals and groups in society. But data systems both arbitrate among competing claims to material and moral goods and shape how much control one has over one’s life. These are the two classic questions

^a See 8 Principles of Open Government, 2007; <http://bit.ly/1KbMC0I>. For further discussion see the author’s article “From Open Data to Information Justice,” in *Ethics and Information Technology* 14, 4 (Dec. 2014), 263–274; <http://dx.doi.org/10.1007/s10676-014-9351-8> and http://papers.ssrn.com/sol3/cf_dev/AbsByAuth.cfm?per_id=1459381.

of philosophical justice, raising the question of information justice.

Data is a social process, not just a technical one. Data, the structures that store it, and the algorithms that analyze it are assumed to be objective reflections of an underlying reality that are neutral among all outcomes. But data scientists constantly make choices about those systems that reflect both technical and social perspectives. One common validation table for a gender field contains only two values, “Male” and “Female.” But many include an “Unspecified” value as well; Facebook began allowing dozens of different values in 2014. Or the validation table might not exist at all, storing whatever text subjects want to use to describe their gender identities.

A data architect charged with storing such data must choose the specific architecture to be implemented, and there are few truly technical constraints on it; yet practice often depends on adopting one answer around which a system can be designed. The design of the gender field and its associated validation table are thus, in part, social choices. They might be decided based on conscious decisions about gender norms, or based on organizational data standards that are the result of social or political processes. The Utah System of Higher Education, for example, deprecated the “Unspecified” value in 2012 to make reporting comply with data standards for the U.S. Integrated Postsecondary Education Data System. This makes them, as much as any act of the political systems, choices about how society will be organized.

Information systems cannot be neutral with respect to justice. Information justice is a reflection of Kranzberg’s first law: “Technology is neither good nor bad, nor is it neutral.”³ Many questions of information justice arise as a consequence of a problem familiar to data scientists: the quality of data. “Injustice in, injustice out” is a fundamental principle of information systems. Gender is a good example of what I call the “translation regime” of a data system. Because many different data frameworks can represent the same reality, there must be a process of translating reality into a single data state. That incorporates technical constraints but also the social assump-

This is not simply a problem in data architecture that can be overcome by better architecture.

tions and processes of that reality. Together, these structures translate reality into a definitive data state. When the translation regime is built around injustice—when it is built around one group’s prejudices about the world, for instance—the associated data system perpetuates that injustice by embedding it in the processes the data informs while at the same time hiding it behind a veil of technicity.

Translation is not the only way that injustice enters data systems. Data collection processes may bias data in favor of the privileged. The undercount of minority households in the decennial U.S. Census is a consequence of a wide range of barriers to participation, barriers that are not equally distributed. The result is that the process is more likely to capture data about those who are relatively privileged rather than those who work two jobs and are rarely home, who do not live as a single family unit, who move frequently, who do not speak the most common languages in their community, who have learned to be wary of government.⁵ These kinds of collection issues can bias conclusions in favor of the privileged, as when cities conclude that building code violations are as common among the wealthy as among the impoverished because the number of violations reported by residents is equal, not considering the wealthy have far lower thresholds for reporting violations when they see them.¹

This is not simply a problem in data architecture that can be overcome by better architecture. The problem exists because the data architecture is embedded in a wider constellation of problems, models, and actions. As Lehrer’s caricature of the famous rocket scientist suggests, data

scientists cannot be content to say the use of their systems is someone else’s problem: where the rockets are meant to come down determines the specifications of the system. Learning analytics systems becoming increasingly common in higher education are built to address the problem of, as Austin Peay State University’s provost described it, students who “find it difficult to make wise choices.”⁴ But wise choices do not necessarily mean the same thing to the provost as they do to any particular student. Austin Peay expects students to graduate on time; decisions that lead away from that are unwise from the institution’s perspective. Hence a data system that includes the information and models needed to predict students’ course grades but not how much they are challenged. The design specifications and intended uses of a system are key sources of its social elements: data systems only capture what is made legible to them, and that depends on what the data system exists to do.

All of these factors contributed to the failure of RTC digitalization. The RTC was accepted to the exclusion of the kinds of informal and historical knowledge that had long been the basis of land claims in the region. It was stored in a relational database that could not easily query the kinds of unstructured documents that supported other claims. The RTC itself was based on a model of individual ownership that was not the only ownership practice in the region; some land was historically held in common for the community in ways that could not be reflected in the RTC process.⁶ Digital land records maintained in geographic information systems, in spite of being open, became a tool for obscuring the needs of some citizens, for barring participation, and for undermining public welfare.

Information injustice is not a problem of bad data systems, nor are data systems inherently unjust. Robust information systems may just as easily promote as undermine justice. The Map Kibera project used crowdsourced data to identify public services available to residents of slums in Nairobi, Kenya that officials regarded as illegal and thus non-existent.² In this case, data acts as a countervailing power to

oppression by government. Alternative data systems that would have improved the outcomes of the Indian case described here might have digitized more than just the RTC, used a data architecture more friendly to unstructured data, built analytical approaches that did not assume all land was privately owned, or aimed to coherently document and resolve land claims in practice rather than identifying a definitive owner for the purpose of public administration.

There are probably no universally right or wrong choices in information justice, but this does not absolve data architects from considering the justice of their choices and choosing the better over the worse, and when that cannot be done through technical means what is left is an act of politics. A useful solution to information justice is thus to practice information science in a way that makes politics explicit.

One increasingly common way to do this is for information scientists to work with social scientists and philosophers who study technology. There is precedence for this: anthropologists have become frequent and valued collaborators in user experience design. Expertise in the ethical and political aspects of technology can inform the unavoidable choices among social values as opposed to pretending these choices are merely technical specifications.

The same can result from more participatory development processes. If we understand data systems as part of a broader problem-to-intervention nexus, we see the end user is not the person receiving the data report but the one on whom the intervention acts. Just as consulting the data user is now regularly part of business intelligence processes, consulting the people who are subjects of the system should be routine. Their participation is crucial to promoting information justice.

To be sure, justice should be its own reward. But information scientists must be aware of the consequences of information injustice, consequences that go beyond the compliance concerns with which many are already familiar. Student data management firm inBloom provided data storage and aggregation services to primary and secondary schools enabling them to track student progress and success using not only local data but data aggrega-

gated from schools nationwide. Many districts and some entire states adopted inBloom, but the aggregation of such data raised deep concerns about student privacy. After several states backed out of the arrangement because of these concerns, the company ceased operations in 2014.

CEO Iwan Streichenberger attributed inBloom's failure to its "passion" and a need to build public acceptance of its practices—in essence rejecting the legitimacy of the ethical concerns its critics raised.⁷ Whether one accepts the legitimacy of those claims or dismisses them as old-fashioned (as is quite common among information technologists), there is no question that inBloom's business failure was not one of inadequate technology but of inadequate ethical vision. InBloom either failed to appreciate the ethical risks of its technologies and business model or failed to convince the public of new ethical principles that would support them. Either way, information justice has become a business concern as much as a moral one.

But whether a business concern, a moral one, or a political one, the challenge information justice presents is one that can be met. It requires that information scientists work with an eye toward the social, asking critical questions about the goals, assumptions, and values behind decisions that are too easily—but mistakenly—seen as merely technical. **C**

References

1. Big data excerpt: How Mike Flowers revolutionized New York's building inspections. *Slate*, 2013; <http://slate.me/1huqx0p>
2. Donovan, K. Seeing like a slum: Towards open, deliberative development. SSRN Scholarly Paper (Mar. 5, 2013) Social Science Research Network, Rochester, NY; <http://papers.ssrn.com/abstract=2045556>
3. Kranzberg, M. Technology and history: "Kranzberg's Laws." *Technology and Culture* 27, 3 (July 1986), 544–560.
4. Parry, M. College degrees, designed by the numbers. *Chronicle of Higher Education*. (June 18, 2012); <https://chronicle.com/article/College-Degrees-Designed-by/132945/>
5. Prewitt, K. The U.S. decennial census: Politics and political science. *Annual Review of Political Science* 13, 1 (May 2010), 237–254.
6. Raman, B. The rhetoric of transparency and its reality: Transparent territories, opaque power and empowerment. *The Journal of Community Informatics* 8, 2 (Apr. 2012).
7. Singer, N. InBloom student data repository to close. *The New York Times* (Apr. 22, 2014), B2; <http://nyti.ms/1PUYyIq>

Jeffrey Johnson (Jeffrey.Johnson@uvu.edu) is Interim Director of Institutional Effectiveness and Planning at Utah Valley University in Orem, UT.

Copyright held by author.

Calendar of Events

March 1

I3D '16: Symposium on Interactive 3D Graphics and Games, Sponsored: ACM/SIG, Contact: Chris Wyman, Email: chris.wyman@ieee.org

March 2–5

SIGCSE '16: The 47th ACM Technical Symposium on Computing Science Education, Memphis, TN, Sponsored: ACM/SIG, Contact: Jodi L Tims, Email: jltims@bw.edu

March 7–10

IUI '16: 21st International Conference on Intelligent User Interfaces, Sonoma, CA, Co-Sponsored: ACM/SIG, Contact: John O'Donovan, Email: jodmail@gmail.com

March 9–11

CODASPY '16: 5th ACM Conference on Data and Application Security and Privacy, New Orleans, LA, Sponsored: ACM/SIG, Contact: Elisa Bertino, Email: bertino@purdue.edu

March 10–11

TAU '16: ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Santa Rosa, CA, Sponsored: ACM/SIG, Contact: Debjit Sinha, Email: debjitsinha@yahoo.com

March 12–16

PPoPP '16: 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Barcelona, Spain, Sponsored: ACM/SIG, Contact: Rafael Asenjo, Email: asenjo@uma.es

March 13–17

CHIIR '16: Conference on Human Information Interaction and Retrieval, Carrboro, NC, Sponsored: ACM/SIG, Contact: Diane Kelly, Email: dianek@email.unc.edu



The Profession of IT Fifty Years of Operating Systems

A recent celebration of 50 years of operating system research yields lessons for all professionals in designing offers for their clients.

OPERATING SYSTEMS ARE a major enterprise within computing. They are hosted on a billion devices connected to the Internet. They were a \$33 billion global market in 2014. The number of distinct new operating systems each decade is growing, from nine introduced in the 1950s to an estimated 350 introduced in the 2010s.^a

Operating systems became the subject of productive research in late 1950s. In 1967, the leaders of operating systems research organized the SOSP (symposium on operating systems principles), starting a tradition of biannual SOSP conferences that has continued 50 years. The early identification of operating system principles crystallized support in 1971 for operating systems to become part of the computer science core curriculum (see the sidebar).

In October 2015, as part of SOSP-25, we celebrated 50 years of OS history. Ten speakers and a panel discussed the evolution of major segments of OS, focusing on the key insights that were eventually refined into OS principles (see <http://sigops.org/sosp/sosp15/history>). A video record is available in the ACM Digital Library. I write this summary not only because we are all professional users of operating systems, but also because these 50 years of operating

systems research yield important lessons for all computing professionals who design systems for customers.

Timeline

A remarkable feature of our history is that the purposes and functions of an operating system have changed so much, encompassing four stages:

- Batch systems: one job at a time (1950–1960);

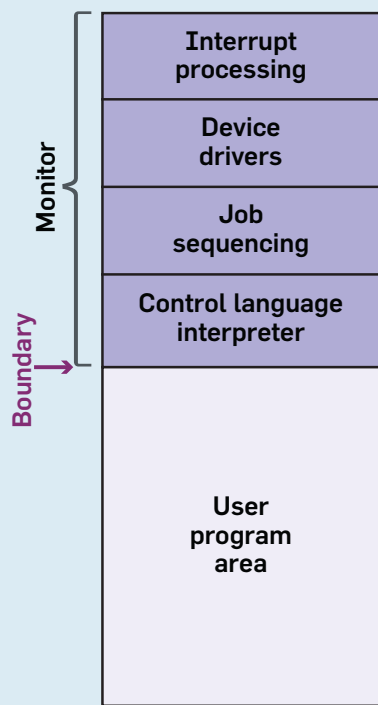
- Interactive systems: many users on multiple systems constantly interacting, communicating, and sharing resources (1960–1975);

- Desktop systems: Immersive personalizable distributed systems to manage work in an office (1975–2005); and

- Cloud-mobile systems: Immersive personalizable systems to manage all aspects of one’s life, work, and social relations (2005 onward)

The accompanying figure depicts a memory layout of an early batch operating system.

Memory layout of an early batch operating system.



The Great Confluence of 1965

The very first operating systems were little more “manual operating procedures” for the first computers in the 1950s. These procedures established a queue of jobs waiting to be executed; an operator put the jobs on the machine one by one and returned output to the requesting users. These procedures were soon automated in the late 1950s; IBM’s 1401 front end to the IBM 709x number crunchers was the best known of commercial “spooling” systems. From that time on, computer system engineers became interested in automating all aspects of computing including in-execution job scheduling, resource allocation, and user interaction, and pre-execution job design, preparation, testing, and debugging. By 1965, their experiments yielded a set of eight principles that became the starting point for a new generation of operating systems:

^a See https://en.wikipedia.org/wiki/Timeline_of_operating_systems

- ▶ Interactive computing (time-sharing)
- ▶ Hierarchical file systems
- ▶ Fault tolerant structures
- ▶ Interrupt systems
- ▶ Automated overlays (virtual memory)
- ▶ Multiprogramming
- ▶ Modular programming
- ▶ Controlled information sharing

The MIT Multics project (<http://multicians.org>) and the IBM System 360 project were the first to bring forth systems with all these characteristics; Multics emphasized interactivity and community, System 360 a complete line of low to high-end machines with a common instruction set. Moreover, Multics used a high-level language (a subset of PL/I) to program the operating system because the designers did not want to tackle a system of such size with assembly language. Developed from 1964 to 1968, these systems had an enormous influence later generations of operating systems.

Dennis Ritchie and Ken Thompson at Bell Labs loved the services available from Multics, but loathed the size and cost. They extracted the best ideas and blended with a few of their own to produce Unix (1971), which was small enough to run on a minicomputer and was written in a new portable language C that was close enough to code to be efficient and high level enough to manage OS program complexity. Unix became a ubiquitous standard in the configuration interfaces of operating systems and in the middleware of the Internet. In 1987, Andy Tanenbaum released Minix, a student-oriented version of Unix. His student, Linus Torvalds, launched Linux from Minix.

OS Principles

By the late 1960s OS engineers believed they had learned a basic set of principles that led to reliable and dependable operating systems. The SOSP institutionalized their search for OS principles. In my own work, I broadened the search for principles to include all computing^{3,4} (see <http://greatprinciples.org>).

I am often asked, “What is an OS (or CS) principle?” A principle is a statement either of a law of computing (Box 1) or of design wisdom for computing (Box 2).

Box 1. Examples of Laws

Semaphore invariant $c(t) = \min(a(t), s(t)+I)$ [5]

Space-time law: memory used = (spacetime per job) × (system throughput)

Mean value equations for throughput and response time in a queueing network

Locality principle

Box 2. Examples of Design Wisdom

Information hiding

Levels or layers of abstraction

Atomic transactions

Virtual machines

Least privilege

Of the many possible candidates for principle statements, which ones are worthy of remembering? Our late colleague Jim Gray proposed a criterion: A principle is great if it is “Cosmic”—it is timeless and incredibly useful. Operating systems contributed nearly one-third of the 41 great principles listed in a 2004 survey (see <http://greatprinciples.org>). The accompanying table gives examples—OS is truly a great contributor to the CS field.

Lessons

As I looked over the expanse of results achieved by the over 10,000 people who participated in OS research over the past 50 years, I saw some lessons that apply to our daily work as professionals.

Even though it seems that research is academic and does not apply to professional work, a closer look at

Both the researcher and professional search for answers. The one pushes the frontier of knowledge, the other makes systems more valuable to customers.

Founding History

My first volunteer position in ACM was editor of the SICTIME newsletter in 1968. SICTIME was the special interest committee on time-sharing, a small group of engineers and architects of experimental time-sharing systems during the 1960s. Jack Dennis (SICTIME) and Walter Kosinski (SICCOMM) organized the first symposium on operating systems principles (SOSP) in 1967 to celebrate the emergence of principles from the experimental systems and to promote research that would clearly articulate and validate future operating system principles. It is significant that they recognized the synergy between operating systems and networks before the ARPANET came online; Larry Roberts presented the ARPANET architecture proposal at the conference. The conference inspired such enthusiasm that the leaders of SICTIME wanted to convert their SIC to a SIG (special interest group); they recruited me to spearhead the conversion. I drafted a charter and bylaws and proposed to rename the group to operating systems because time-sharing was too narrow. The ACM Council approved SIGOPS in 1969 and ACM President Bernard Galler appointed me as the first chair. One of my projects was to organize a second SOSP at Princeton University in 1969. That conference also inspired much enthusiasm, and every two years since then SIGOPS has run SOSP, which evolved into the premier conference on operating systems research. SIGOPS has identified 48 Hall of Fame papers since 1966 that had a significant shaping influence on operating systems (see <http://www.sigops.org/award-hof.html>).

Following these successes, in 1970 Bruce Arden, representing COSINE (computer science in engineering), an NSF-sponsored project of the National Academy of Engineering, asked me to chair a task force to propose an undergraduate core course on operating system principles. A non-math core course was a radical idea at the time, but the existence of so many OS principles gave them confidence it could be done. Our small committee released its recommendation in 1971.¹ Many computer science and engineering departments adopted the course and soon there were several textbooks. I wrote a follow-on paper in 1972 that explained the significance of the paradigm shift of putting systems courses in the CS core.² After that, ACM curriculum committees began to include other systems courses in the core recommendations. The place of OS in the CS core has gone unchallenged for 45 years.

—Peter J. Denning

**A Fistful of Bitcoins:
Characterizing
Payments Among
Men with No Names**

**Security Multiparty
Computations
on Bitcoin**

**Forty Years
of Suffix Trees**

**Does the Use of Color
on Business Dashboards
Affect Decision Making?**

**Multimodal Biometrics
for Enhanced
Mobile Device Security**

Beyond Viral

**Why Logical Clocks
Are Easy**

**More Encryption
Means Less Privacy**

**How SysAdmins
Devalue Themselves**

Plus the latest news
about automating
proofs, mobile-assistive
technologies, and
search engine biases.

Examples of computing principles contributed by operating systems.

Process	A program in execution on a virtual processor. A process can be started, stopped, scheduled, and interacted with. Operating systems and networks comprise many interacting processes that never terminate.
Interactive Computation	Processes can receive inputs or generate outputs at any time—contrasts with the Turing view that processes get all their input at the start and produce all their output at the end. Interactive computations can implement functions that non-interactive computations cannot.
Concurrency controls	To avoid pathologies such as race conditions, buffer overflows, and deadlocks, processes need explicit mechanisms to wait for and receive signals.
Locality	Processes use small subsets of their address spaces for extended periods. Caches and memory managers detect working sets, position them for significantly improved performance, and protect them to prevent thrashing.
Naming and mapping	Objects can be assigned location-independent names; mappers translate names to object physical locations when needed. Hierarchical naming systems (such as directories and URLs) scale to very large name spaces.
Protection and Sharing	The global name space is visible to everyone (for example, the space of all Web URLs). Objects are by default accessible only to their owners. Owners explicitly state who is allowed to read or write their objects.
System Languages	System programming languages yield systems that are well structured, more easily verified, and fault tolerant.
Levels of Abstraction	System software can be simplified and verified by organizing the functions as a hierarchy that can make only downward calls and upward returns.
Virtual machines	A set of related functions can be implemented as a simulation of a machine whose interface is an “instruction set” and whose internal structure and data are hidden.

what actually happens reveals a great deal of overlap. Both the researcher and the professional seek answers to questions. The one aims to push the frontier of knowledge, the other to make a system more valuable to a customer. If we want to find out what it is like to explore a question, our main sources are academic research papers; there are very few written professional case studies. The typical research paper tells a tidy story of an investigation and a conclusion. But the actual investigation is usually untidy, uncertain, and messy. The uncertainty is a natural consequence of numerous contingencies and unpredictable circumstances through which the investigator must navigate. We can never know how a design proposal will be received until we try it and see how people react.

You can see this in the presentations of the speakers at the conference, as they looked back on their struggles to find answers to the questions they asked. They were successful because they allowed themselves to be beginners constantly searching for what works and what does not work: building, tinkering, and experimenting. From this emerged many insights.

The results of their work were almost always systems that others could use and experiment with. After the messy process of learning what worked, they wrote neat stories about what they learned. Before they produced theories, they first produced prototypes and systems.

Professionals do this too. When sitting around the fire spinning yarns of what they did for their customers, they too tell neat stories and graciously spare their clients their struggles with their designs.

References

1. COSINE Task Force 8 report. An Undergraduate Course on Operating Systems Principles. National Academy of Engineering, 1971; <http://denninginstitute.com/pjd/PUBS/cosine-8.pdf>
2. Denning, P. Operating systems principles and undergraduate computer science curricula. In *Proceedings of AFIPS Conference*. 40 (SJCC), 1972, 849–855; <http://denninginstitute.com/pjd/PUBS/OSprinciples.pdf>
3. Denning, P. Great principles of computing. *Commun. ACM* 46, 11 (Nov. 2003), 15–20.
4. Denning, P. and Martell, C. *Great Principles of Computing*. MIT Press, 2015.
5. Habermann, A.N. Synchronization of communicating processes. *Commun. ACM* 15, 3 (Mar. 1972), 171–176.

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, CA, is Editor of *ACM Ubiquity*, and is a past president of ACM. The author's views expressed here are not necessarily those of his employer or the U.S. federal government.

Copyright held by author.

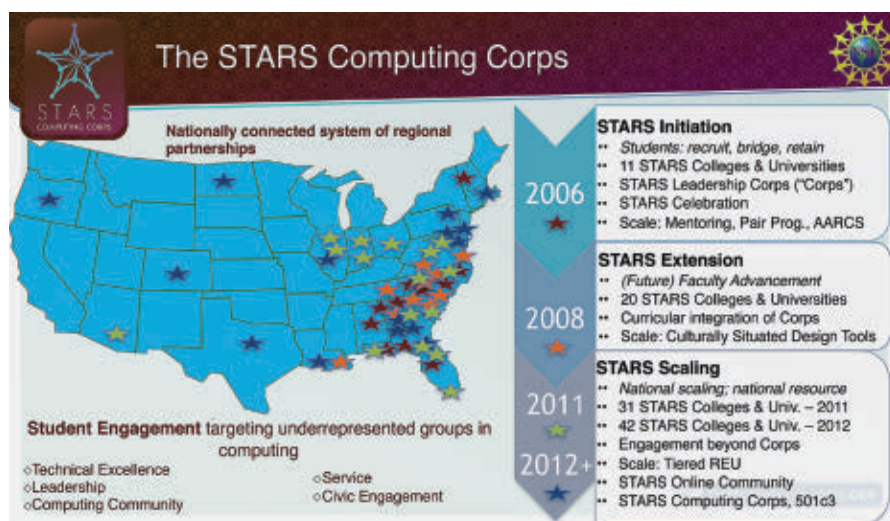
Broadening Participation

The Need for Research in Broadening Participation

In addition to alliances created for broadening participation in computing, research is required to better utilize the knowledge they have produced.

UNDERREPRESENTATION IN COMPUTING is a *global* problem, marked by a disturbing lack of access to computing resources and education among people underrepresented by race, ethnicity, gender, income, disability, and sexual-orientation status. It is urgent that we address this divide between those with and without the knowledge to create computational artifacts or even basic functional literacy. Important alliances for broadening participation (BP) are catalyzing efforts to engage more people in computing, but they are not enough. We need solid research as well.

The U.S. National Science Foundation (NSF) has funded eight current Broadening Participation in Computing Alliances to increase the diversity of computing students in the U.S. These alliances have built networks among diverse computing students and professionals while building capacity to improve the culture of computing. The National Center for Women & Information Technology (NCWIT; <https://www.ncwit.org/>) has mobilized over 600 organizations to recruit, retain, and advance women in computing. The Computing Alliance for Hispanic-Serving Institutions (CAHSI; <http://cahsi.cs.utep.edu/>) has provided mentoring and effective educational practices like peer-led team learning and the Affinity Research Group model to integrate undergraduates into research teams. The Alliance for Access to Computing Careers (Ac-



The STARS Computing Corps is one of eight NSF-funded Broadening Participation in Computing Alliances.

cessComputing; <http://www.washington.edu/accesscomputing/>) promotes inclusive practices and builds community for students with disabilities. The Institute for African American Mentoring in Computer Sciences (IAAMCS; <http://www.iaamcs.org/>) encourages undergraduate students in research, K-12 outreach, and participation in seminars and the Tapia Celebration of Diversity in Computing (<http://tapiac-conference.org/>). Expanding Computing Education Pathways (ECEP; <http://expandingcomputing.cs.umass.edu/>) supports state-level computing education reforms. Into the Loop (<http://explorings.org/>) works to integrate rigorous computing courses in Los Angeles Unified School District high

schools. The Computing Research Association-Women (CRA-W; <http://cra.org/cra-w>) and the Coalition to Diversity Computing (CDC; <http://www.cdc-computing.org/>) support Sustainable Diversity in the Computing Research Pipeline (<http://cra.org/cepp/>), and the Data Buddies project across programs. The STARS Computing Corps (<http://starscomputingcorps.org>) develops university and student leaders to serve local communities through regional partnerships and an annual STARS Celebration conference.

BP Research Needs

Alliances build support for underrepresented groups through extensive, interdisciplinary research at the boundaries

of computing, education, and the social sciences, but such work is underestimated and undervalued, with few publication venues and low disciplinary recognition.³ To leverage the knowledge alliances are building, research is needed to identify what works, to replicate it, and to synthesize findings into a theoretical framework. Computing professionals and researchers must value this BP research, credit those doing this difficult work, and help them advance their careers. This can only happen with the support of a dedicated community that produces peer-reviewed conferences and journals.⁴ The STARS Computing Corps (<http://starscomputingcorps.org>) has helped foster this new professional community.

Since its founding in 2006, STARS has mobilized more than 80 faculty and 2,100 students at 51 colleges and universities to lead projects that broaden participation in computing. STARS has focused on BP interventions, such as student-led outreach projects that have reached over 130,000 K–12 students. At the annual STARS Celebration conference, STARS students and faculty share their work and meet with BP thought leaders. Support, recognition, and a peer-review process were enlisted from professional societies to expand the Celebration conference to include a new conference on BP research.⁴ The IEEE Computer Society Special Technical Community (STC) on Broadening Participation helped sponsor RESPECT'15, (Research on Equity and Sustained Participation in Engineering, Computing, and Technology, held August 13–14, 2015 in Charlotte, NC; <http://respect2015.stcbp.org/>). The IEEE Computer Society's STCBP and the ACM Special Interest Group on Computer Science Education (SIGCSE; <http://www.sigcse.org/>), have helped establish a community that ensures rigorous BP research publications. These publications are available to IEEE and ACM members through the organizations' digital libraries.

Research and the RESPECT Conference

The RESPECT interdisciplinary research conference draws on computer science, education, learning sciences, and the social sciences. It builds the foundation for broadening participa-

We, as computing professionals, have a responsibility to improve computing culture to be more inclusive for everyone.

tion research. The first annual STCBP conference, RESPECT'15, was co-organized by STARS and co-located with the STARS Celebration (<http://www.starscomputingcorps.org>) to leverage and engage the existing activist-oriented STARS community in BP research. The RESPECT and Celebration conferences' joint "RESPECT for Diversity" theme highlighted our common belief that the engagement of diverse people in computing is a matter of equity—all people deserve the opportunity to engage in work that is integral to solving increasingly complex global challenges.

RESPECT'15 research papers showed BP programs are effective, and are more impactful for people from underrepresented groups. Many of these programs engage people from underrepresented groups in BP work—but this could, ironically, limit the diverse perspectives that impact technical innovations. Exploding demand for computing degrees is outpacing growth in computing programs—disproportionately impacting those without access to the "preparatory privilege" of extra experience in science, technology, engineering, math, and computer science.⁵ Zweben and Bizot⁷ found that the percentage of women in undergraduate programs in computing is continuing to decline. Robinson et al.⁶ studied diverse students, finding that subtle racial and gendered discriminations called microaggressions erode the culture and student experience in graduate engineering programs.

We, as computing professionals, have a responsibility to improve computing culture to be more inclusive for everyone. Educators should consider how to enable students to become

more interested in and excited about computing. Professionals should reach out and help others learn computing by connecting through STARS, NCWIT, or other alliances. Parents can encourage their children to learn computing, and lobby schools to get computing in the curriculum. Leaders should consider how to change company culture and practices to create more diverse technical and leadership teams. These actions should be informed by emerging BP research, such as the RESPECT'15 proceedings,¹ and the *Computing in Science & Engineering* special issue on the best of RESPECT.² Those committed to broadening participation are welcome to join the IEEE Computer Society's special technical community on Broadening Participation (<http://stcbp.org>) and to participate in RESPECT'16 and the STARS Celebration in Atlanta, GA, from August 11–13, 2016. 

References

1. Barnes, T. et al. In *Proceedings of the First Annual International Conference on Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT'15)*, Charlotte, NC, (Aug. 13–14, 2015); <http://bit.ly/1JplqRe>.
2. Barnes, T. et al. Special issue on the best of RESPECT 2015. *Computing in Science and Engineering* (2016); <http://bit.ly/1nDIS3L>.
3. Chubin, D.E. and Johnson, R.Y. A program greater than the sum of its parts: The BPC alliances. *Commun. ACM* 54, 3 (Mar. 2011), 35–37; DOI: <http://dx.doi.org/10.1145/1897852.1897866>
4. Dahlberg, T. Why we need an ACM special interest group for broadening participation. *Commun. ACM* 55, 12 (Dec. 2012), 36–38.
5. Margolis, J. et al. *Stuck in the Shallow End: Education, Race, and Computing*. MIT Press, 2008.
6. Robinson, W.H. et al. Racial and gendered experiences that dissuade a career in the professoriate. *Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)* (Aug. 2015), 1–5.
7. Zweben, S.H. and Bizot, B. Representation of women in postsecondary computing 1990–2013: Disciplines, institutional, and individual characteristics matter. In *Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT 2015)*. IEEE, 2015, 1–8.

Tiffany Barnes (tiffany.barnes@gmail.com) is an associate professor of Computer Science at North Carolina State University, Raleigh, NC. She co-chairs the IEEE Computer Special Technical Community on Broadening Participation and serves on executive boards for ACM SIGCSE, EDM, and AIED.

George K. Thiruvathukal (gkt@cs.luc.edu) is Professor of Computer Science at Loyola University Chicago and a Visiting Computer Scientist at Argonne National Laboratory in the Mathematics and Computer Science Division/Argonne Leadership Computing Facility. He co-chairs the Special Technical Community on Broadening Participation.

RESPECT'15 was made possible by NSF funding (CNS-1042372), the STARS Computing Corps, technical co-sponsorship by IEEE Computer Society, and in-cooperation status with ACM SIGCSE. We particularly thank RESPECT local chair and STARS PI Jamie Payton, and RESPECT program chairs Kristy Boyer and Jeff Forbes. The authors thank Jenny Stout, editor of *Computing in Science and Engineering*, and NC State Ph.D. students Thomas Price and Drew Hicks for comments on drafts of this column.

Copyright held by authors.

Viewpoint

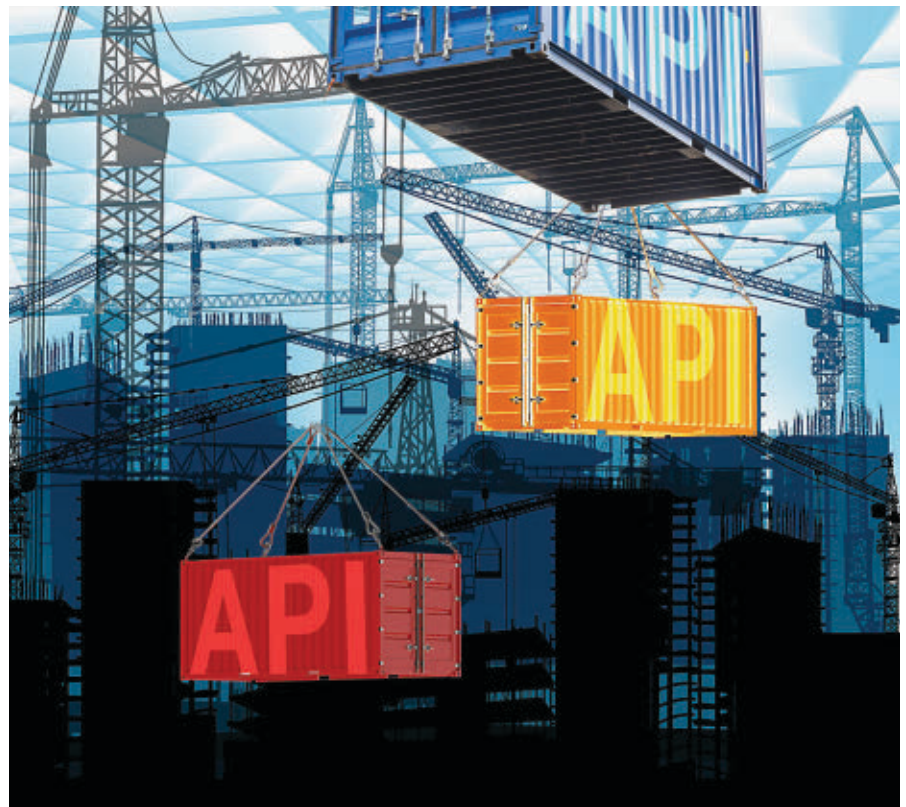
Riding and Thriving on the API Hype Cycle

Guidelines for the enterprise.

APPPLICATION PROGRAMMING INTERFACES (APIs) are, in the simplest term, specifications that govern interoperability between applications and services. Over the years, the API paradigm has evolved from beginnings as purpose-built initiatives to spanning entire application domains.⁸ Driven by the promise of new business opportunities, enterprises are increasingly investigating API ecosystems. In this Viewpoint, we discuss the challenges enterprises face in capitalizing on the potentials of API ecosystems. Is the investment in API ecosystems worth the promise of new profits? From a technical perspective, standardization of APIs and a systematic approach to consumability are critical for a successful foray into API ecosystems.

The Web API Economy: No Longer SOA's Adolescence

When the service-oriented architecture (SOA) concept emerged in the early 2000s, it attracted many companies that saw the benefits of bolstering business-to-business relationships via standard interfaces, often implemented via the simple object access protocol (SOAP). Later, SOA evolved into more Web-friendly technologies, such as REpresentational State Transfer (REST) that greatly simplified reusability, partly because the create, read, update, and delete (CRUD) interface was more approachable to even the most casual developers. While SOA



was largely confined to the enterprise and focused on interoperability, REST APIs brought the power of reuse within reach of individual developers at Internet-scale through consumability.⁷ Developers now enjoyed cheap and easy access to deep computing capabilities and vast amounts of data that were hitherto hidden behind closed enterprises, driving today's surging API ecosystems. Today, organizations are heavily competing in the "API game," rapidly externalizing their business

assets and hoping to vastly monetize locked up data and services.

ProgrammableWeb (see <http://programmableweb.com>), the largest free online API registry, claims API availability exhibited a compounded annual growth rate of approximately 100% from 2005 to 2011. As of April 2015, ProgrammableWeb listed over 13,300 APIs. This trend is not necessarily the result of altruistic nurturing of crowd-based creativity—APIs are huge business enablers! For example, Salesforce

as a leading enterprise customer relationship management (CRM) provider, offers APIs to enable broad proliferation of its CRM capabilities into its customers' own systems. Today, 60% of Salesforce's transactions go through its API, contributing to its 1.3 billion daily transactions and more than \$5 billion in annual revenue.⁸ APIs are primary business drivers not only for the born-on-the-cloud businesses, but are also helping traditional businesses, such as financial services, to reinvent for their survival and prosperity.⁵

To extract value from some business asset, a set of services interconnected by APIs must be established. APIs that are part of an ecosystem are more valuable than when they exist in isolation. To create successful business models around the API economy, it is important to develop an ecosystem of partners and consumers. It is also important to understand technology trends affecting applications that consume APIs. Container-based technologies such as Docker^a are making it possible for developers and service providers to rapidly develop and deploy their services using standardized interfaces. This allows the ecosystem to rapidly evolve while the participants focus on their core competencies. In such an ecosystem, Platform-as-a-Service (PaaS) providers manage the IT service automation using APIs, while Software-as-a-Service (SaaS) providers supply specialized services. Successful business models find a niche in an existing ecosystem or create new ecosystems using APIs.

The greatest risk for enterprises remains a lack of a sound API strategy.⁵ We discuss key challenges that must be overcome to tame inflated API expectations and form a healthy, self-evolving API ecosystem.

API success factors. API ecosystems bring much more attention to API consumability, ease of reuse, and reuse in contexts not originally envisioned by the provider—sometimes referred to as serendipitous reuse.¹⁰ APIs initially generated a lot of hype for enterprises, given the potential of new client bases through the promise of almost accidental reuse of APIs. Yet, the critical

question is how can enterprises design the desirable APIs, for easy reuse, and avoid the investment losses from (re-)design of ineffective APIs and deployment of infrastructure to host and support them? The probability of API success is largely a function of where an organization is in its digital evolution.⁴ On one hand, there exist born-on-the-Web businesses that have developed their core identities around APIs (for example, Twilio^b and Stripe^c). Such companies enjoyed the first-mover advantage and are currently benefiting from huge consumer demand. On the other hand, there are the pre-Web businesses (such as large banks and healthcare institutions), which have only recently started investigating API strategies. They typically have to proceed along the API path more cautiously since the market has become flooded and many consumers already have their preferred APIs. Another factor that seemingly affects API success actually falls counter to the notion of serendipity. Early evidence indicates strong business models bolster successful APIs.² Specifically, the approach of releasing free APIs to judge value doesn't always yield strong API adoption, and may require multiple iterations of APIs. Observations have shown that successful external APIs are first frequently used both internally and by strategic partners.^{3,9} That is, successful APIs are designed according to a use case that has already demonstrated value to a proven business function as well as being of high quality. Hence, throwing the APIs into the wild, without continuously improving

b <http://www.twilio.com>

c <http://www.stripe.com>

The greatest risk for enterprises remains a lack of a sound API strategy.

them, is ripe for failure. Success of APIs depends on the level of digital maturity of the enterprise and the corresponding API adoption models, which vary in complexity and their required investment. For example, some enterprises might just start with API experimentation, before exposing own capabilities or reinventing them. The final, most challenging step is to embrace new business models through APIs.⁴ Therefore, the businesses that want to leverage the power of APIs need reiterate both on APIs and the underlying business models in order to ensure proven value over the time.

Abundance of API choice. API consumers, especially those developing enterprise applications, need to quickly identify APIs that satisfy their functional and non-functional requirements, such as performance capabilities and security properties. In SOA, Universal Description, Discovery and Integration (UDDI) registries^d were meant to assist here, but they had several drawbacks. A UDDI registry was a passive entity meant to be looked up at the time of service invocation. The service information contained therein would often become stale as services went down or changed their interface and characteristics. Further, it was tied to specific technologies, such as SOAP. One way to address the abundance of choice is via automated API brokers that enable quick and easy selection of APIs that meet consumers' requirements. API brokers can be viewed as an emerging field of cognitive systems that can make these selection assessments, based on the dynamic insights about the ever-changing API ecosystem.¹¹ Gradually, API Broker recommendations will improve, taking into an account factors such as ease of integration and guidelines from consumer profiles.

API exposure. Mainstream adoption of APIs relies on the providers, as they encapsulate information and capabilities that will remain the bedrock of an API ecosystem. The provider must consider how it will enforce security, integrity, and quality of its exposed resources in such a diffused network of API providers and consumers. For example, given the growing need for predictive analysis for perfor-

a Docker: Build, Ship and Run Any App, Anywhere; <https://www.docker.com>

d UDDI Reference; <http://bit.ly/1RtL3H>

Enterprises wishing to establish or be part of an API ecosystem need to clear a number of challenges.

mance, where does monitoring start and who is accessing and controlling it? Moreover, how does one identify what usage trends are indicative of strains on the providers' resources? The provider has the added challenge of determining the risks involved and the investments required to mitigate them. The reality is that years of acquisitions, mergers, and crumbling knowledge of legacy IT systems have made the sheer possibility of exposing potentially valuable capabilities via APIs a complex undertaking. Any required infrastructure modernization by API provider must be balanced with incremental API exposure by carefully selecting the resources, understanding their provenance, and identifying the access methods. Furthermore, open use of APIs brings challenges of possible unintended consequences, which may be caused by, for example, the interplay between providers, consumers, who may be afraid of vendor lock-in, third-party agents, and the end users. Similarly, giving API consumers the means for evaluating API terms of service and tracking their change presents yet another barrier for enterprises. API management gateways^e are first steps at addressing these hurdles and help providers mitigate risk by throttling and protecting access to the backend resources.

API consumability. Serendipitous reuse of APIs, coupled with potential "network effects" bringing spiked API usage, creates unexpected workloads that require the provider to consider elastic cloud architectures to maintain a consistent quality of service. It also creates new challenges that call

for the development of metrics, which will evaluate the API output quality with respect to the ease of serendipitous reuse. An example is testing if the data exposed by the API follow the principles of linked data¹ so that they can be interlinked and become more useful. As a result, building a better semantic understanding of each API, its use, and attributes can improve consumability. As the ecosystem grows, it is important to nurture it and evolve the APIs in a manner that is consistent with the use that is observed, maintaining compatibility and optimizing for those scenarios that offer sound business value.

Overcoming the API Hype: Enabling Enterprise-Level API Ecosystems

Enterprises wishing to establish or be part of an API ecosystem need to clear a number of challenges. Firstly, they need to thrive in the flood of APIs, where the key will be to build a core set of competitively differentiated APIs. A capability that provides personalization of the API consumption experience will be vital to reach the target audience and business need. In parallel, it is critical to create an engaging ecosystem experience through consumable APIs and the data they manipulate, an environment that fosters partners to exploit the data via the APIs to create new offerings and makes users to keep coming back for more; hence, developing a "stickiness" for the ecosystem. Ecosystem members are looking for trusted providers who offer quality, reliable, long-lasting, and supported APIs. A study of API usage with open data APIs for civic apps⁶ found that while there was a general lack of success with the first generation of applications, there is more traction now with efforts to improve data quality for stickiness, and efforts to increase API awareness by engaging application developers in cities and providing sustained financial incentives.

Conclusion

Success lays ahead for providers with sound API strategies that iterate through the business models concurrently improving the technical aspects of the APIs. They need to maneuver the

challenges of legacy transformation and IT investments while they attract and retain consumers that drive real-time composition of their APIs, and fuel the growth of their ecosystem, successfully navigating around the hype surrounding APIs. 

References

1. Bizer C., Heath T., and Berners-Lee T. Linked data—The story so far. *International Journal on Semantic Web and Information Systems* (2009), 1–22.
2. Boyd, M. Real-World API Business Models that Worked; <http://bit.ly/1ZgfOwd>
3. Freedman, C. 5 Things to Understand About Successful 'Open' APIs; <http://bit.ly/1k6X6XX>
4. Holley, K. et al. The Power of the API Economy: Stimulate Innovation, Increase Productivity, Develop New Channels and Reach New Markets. IBM Redbook; <http://ibm.co/1QRtGa>
5. Jacobson D., Brail G., and Woods D. *APIs: A Strategy Guide: Creating Channels with Application Programming Interfaces*. O'Reilly Media, 2011.
6. Lee M., Almirall, E., and Wareham, J. Open Data & Civic Apps: 1st Generation Failures—2nd Generation Improvements. ESADE Business School Research Paper No. 256, (Oct. 2014).
7. Pautasso, C., Zimmermann, O., and Leymann, F. RESTful web services vs. "big" web services: Making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web (WWW)*. (ACM, 2008).
8. Salesforce Architecture. How they Handle 1.3 Billion Transactions a Day; <http://bit.ly/1mvauQp>
9. Takeuchi J. 8 Tips to Cultivating a Successful API Program; <http://bit.ly/1OB8F2a>
10. Vinoski, S. Serendipitous reuse. *IEEE Internet Computing* 12, 1 (Jan. 2008), 84–87.
11. Wittern, E. et al. A graph-based data model for API ecosystem insights. In *Proceedings of the 21st IEEE International Conference on Web Services (ICWS)*, (Anchorage, AK, 2014).

Maja Vukovic (maja@us.ibm.com) is a Research Staff Member, Master Inventor, and Member of the IBM Academy of Technology at the IBM T.J. Watson Research Center, Yorktown Heights, NY.

Jim Laredo (laredoj@us.ibm.com) is a Senior Technical Staff Member, Master Inventor, and Member of the IBM Academy of Technology at the IBM T.J. Watson Research Center, Yorktown Heights, NY.

Vinod Muthusamy (vmuthus@us.ibm.com) is a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY.

Aleksander Slominski (aslom@us.ibm.com) is a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY.

Roman Vaculin (vaculin@us.ibm.com) is a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY.

Wei Tan (watan@us.ibm.com) is a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY.

Vijay Naik (vknaik@us.ibm.com) is a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY.

Ignacio Silva-Lepe (isilval@us.ibm.com) is a Research Staff Member with IBM Watson Health, Yorktown Heights, NY.

Arun Kumar (kkarun@in.ibm.com) is a Senior Researcher and Research Manager, and Member of the IBM Academy of Technology, IBM Research-India.

Biplav Srivastava (sbiplav@in.ibm.com) is a Senior Researcher with IBM Research-India.

Joel W. Branch (joel.branch@espn.com) is a Manager and a Staff Software Engineer with the ESPN Advanced Technology Group (this work was done while he was a member of IBM Research).

Copyright held by authors.

^e IBM API Management; <http://ibm.co/1rl5iaiw>

Viewpoint

Paper Presentation at Conferences: Time for a Reset

Seeking an improved paper presentation process.

THE IMPORTANCE OF conferences as a publication medium is now well established in computer science. The methodology of conference reviewing and its role in selecting high quality archival publications has been a subject of much recent discussion. However, there has been little discussion of the role of paper presentation at a conference. This Viewpoint explores this issue and describes an experiment we ran at the VLDB conference.

The traditional paper presentation, at least in the conferences I go to, is 20-plus minutes in duration, most of which time is spent going over details more completely stated in the written paper. These details are also more efficiently understood from the written paper for many of us, who can read technical material faster than we can listen to it. Some attendees may hope for a more in-depth perspective through attending a session. However, when interesting questions do arise, there is usually limited time for discussion. Many would say the goal of a conference presentation should be to convince the audience to read the paper, not to teach them the contents of the paper. But this limited goal can probably be accomplished almost as well in much less time. Consequently, I personally see little value in attending a traditional research talk at a conference.



A remedy that has recently become popular in some conferences is to have a poster session for all papers. This provides an opportunity for the in-depth personal interaction that paper sessions do not provide. Additionally, it is an opportunity for an attendee to browse quickly through many papers, sometimes complemented with a gong-show for this purpose.

With this avenue for in-depth interaction in place, the traditional research paper session is no longer even pretending to fill this role. Indeed, some conferences have gone so far as to restrict oral presentation to only selected papers.

So what should the role of a session be? I think it is self-evident that an interesting session must be conceived and planned as a session on some topic, rather than as a mere union of independent paper presentations. If our focus is on an individual paper, we

can do much better by reading the paper and going to a poster session. The reason to go to a research session is to benefit from the presence in one room of multiple experts interested in some topic (or, at least, closely related topics). The question becomes how best to accomplish this.

In some fields, conferences are organized by session, and papers are invited to particular sessions. Such conferences find it easy to have cohesive sessions. But this flexibility is not available to most computing conferences, which have a carefully devised review process for paper acceptance. In short, in putting a conference program together, we get input at the unit of papers but must produce output at the unit of sessions. This is difficult, but not impossible.

I was recently program chair of the VLDB conference, and this gave me an opportunity to try some things out. So let me describe a few of the things we did, and how it turned out (see <http://www.vldb.org/2014/program/Menu.html>). My evaluation is based on both anecdotal evidence and the results from an attendee survey we conducted.

We asked session chairs (of research paper sessions) to present an overview of the research sub-area, we restricted paper presentations to 12 minutes, and we asked questions to be mostly deferred to a mini-panel at the end of the session. Since this was the

first time we were trying these changes, session chairs were given considerable leeway in how they implemented these changes, and were encouraged to be creative in building an interesting session. Thus we had 32 uncontrolled experiments, one per research paper session, and report here on the “results.”

The reduction in paper-presentation time to 12 minutes was initially resisted by many authors, who were appropriately worried they would not be able to meet the traditional expectations of a conference presentation within the limited time. It was surprisingly easy to modify expectations, attendees mostly loved it, and the shorter presentations were a resounding success. In fact, there may be room to shorten it further, perhaps to 10 minutes per paper presentation. By so doing, we make time for the more interesting parts of the session, described next.

The session chair introductions varied greatly in style. Even their length varied, from under five minutes to more than 15. The response to session chair introductions was generally positive, and most attendees felt it really helped them understand the big picture before diving into the weeds with individual papers. Based on feedback, my strong recommendation is the introductions try to present an overview of the research frontier in the sub-discipline, with a brief mention of where each paper fits in this scheme. The actual contributions of each paper, and the related earlier work that each depends on, are best left to the individual presentation.

The mini-panel at the end had somewhat mixed reactions but was positively received on balance. The two most salient criticisms were: the authors did not engage in discussion with one another and there seemed to be a strong recency effect, resulting in more audience questions addressed to talks later in the session. To counterbalance, an equally common “criticism” was there was insufficient time allotted to the most enjoyable part of the session. If I were doing this again, there are a few things I would do slightly differently. First, I would explicitly ask each author to read the other papers in the session and come prepared with one question they would like to discuss with each of the authors. In contrast,

As program chair, I spent a great deal of time coming up with a good first cut that partitioned papers into sessions.

at VLDB 2014, we only asked authors to communicate with session chairs, and vice versa, ahead of time. Second, I would explicitly ask the session chair to initiate and manage a discussion, selecting at least the first couple of topics to be discussed. At VLDB 2014, our instructions to session chairs were much weaker, merely asking them to come prepared with some questions they could ask the panel if the audience was too passive: in effect, asking them to provide a back-up rather than drive the intellectual stimulation.

How closely related the papers were in a session did appear to matter. In particular, it was evident from the start that we could not possibly have any “potpourri” sessions. To have some flexibility in session allocation, we constrained sessions to have four, five, or six papers. (In a later phase of program construction, sessions with the same number of papers were placed in parallel, to the extent possible).

As program chair, I spent a great deal of time coming up with a good first cut that partitioned papers into sessions. Then, I broadcast this tentative design to all authors, and got some good feedback. Many authors simply requested their own paper be moved to a different session. But many others had more ambitious suggestions, including two suggestions for completely new sessions (of which I was able to adopt one). In short, the authors and session chairs have a vested interest in at least local pieces of the conference organization. Getting them involved before setting things in stone was really helpful.

Even with all care and effort, some sessions were more cohesive than others. This probably cannot be helped.

However, there was no session where the session chair complained about a specific paper being too far of a reach for the session.

Finally, and completely unrelated to the aforementioned changes, we decided to recognize excellence in conference presentation. We deputized the 32 session chairs to nominate candidates for this (non-competitive) certificate, not necessarily from their own sessions. We received 13 nominations, including two from one session chair. This number is well under 10% of the (165) papers presented at the conference. Each of the 13 presenters was recognized with a certificate, and a mention on the conference website. Since this award was specifically for the presentation, only the presenters were recognized and not their co-authors.

We had an active email discussion among session chairs before, during, and after the conference. One issue that concerned many session chairs was the ability of a conference attendee to switch sessions to listen to specific paper presentations. This turned out to be a non-issue in practice. Where people had particular papers of interest, they met the author(s) at the poster session, and there was very little evidence of session hopping for particular paper presentations. Philosophically, I believe the right way to think about the issue is that the paper is the unit of acceptance (and poster presentation) but the session is the unit of conference organization. If attendees want a good conference experience, we need to be thinking about the experience in larger rather than smaller units. I think there is much that conference chairs can do to make research paper sessions first-class entities in the conference program, and we should insist they do, so that conferences remain intellectually stimulating events, and not become just networking venues. For additional details, see “Real Session Chairs Do More Than Just Keep Time” at <http://www.vldb.org/pvldb/vol7/FrontMatterVol7No12.pdf>. 

H.V. Jagadish (jag@umich.edu) is Bernard A. Galler Collegiate Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor.

Copyright held by author.

Interview

An Interview with Stanford University President John Hennessy

Stanford University President John Hennessy discusses his academic and industry experiences in Silicon Valley with UC Berkeley CS Professor David Patterson.

JOHNN HENNESSY JOINED Stanford in 1977 right after receiving his Ph.D. from the State University of New York at Stony Brook. He soon became a leader of Reduced Instruction Set Computers. This research led to the founding of MIPS Computer Systems, which was later acquired for \$320 million. There are still nearly a billion MIPS processors shipped annually, 30 years after the company was founded.

Hennessy returned to Stanford to do foundational research in large-scale shared memory multiprocessors. In his spare time, he co-authored two textbooks on computer architecture, which have been continuously revised and are still popular 25 years later. This record led to numerous honors, including ACM Fellow, election to both the National Academy of Engineering and the National Academy of Sciences.

Not resting on his research and teaching laurels, he quickly moved up the academic administrative ladder, going from the CS department chair to Engineering college dean to provost and finally to president in just seven years. He is Stanford's tenth president, its first from engineering, and he has governed it for an eighth of its existence. Since 2000, he doubled Stanford's endowment, including a record \$6.2 billion for a single campaign. He used those funds to launch many initia-



Stanford President John Hennessy.

tives—which often cross departmental lines—along with new buildings to house them. Undergraduate applications also doubled, for the first time making Stanford even more selective than Harvard.

This interview occurred in the Stanford President's office in July 2015, shortly after he announced that he would step down as president in 2016—Stanford's 125th anniversary. The interviewer is David Patterson,

who is a UC Berkeley CS professor and co-author of Hennessy's two textbooks.

Ending of Moore's Law

DAVID PATTERSON: You received your bachelor's degree in electrical engineering 40 years ago. Amazing things have happened since then in the information technology. What do you think about the next 40 years?

JOHN HENNESSY: You are absolutely right it has been transformative. We

don't even worry about the cost of hardware anymore because it's become so low. Similarly, the cost of communicating data has become equally low, both driven by Moore's Law. It is the era of software now, and because the hardware is so cheap, you can afford to deploy lots of hardware to solve problems. I think the next 40 years are going to be about the excitement that's happening on software.

I am really excited about what is happening on machine learning and deep learning; this is really the breakthrough that was promised for many years. I think the challenge is going to be on the hardware side. As Moore's Law begins to wane, how are we going to deliver the hardware that we need for these kinds of applications?

When I tell people that Moore's Law is ending, they think it means the technology is going to stop improving. We know technology advancement is not ending, but it's going to slow down. What do you think the impact will be?

Hardware has become so cheap that people don't think about throwing away their phone and getting a new one because it's a little bit better. We'll see those things begin to slow down. There hasn't been doubling for quite some time now, so we're already kind of in that sunset period. The challenge is if we continue to be inventive about the way we use information technology, how are we going to deliver the hardware that enables people to use it cost effectively?

Do you mean design?

Yes. How are we going to continue to make it cheaper? We still don't have our hands completely around the issue of parallelism. We switched to multicore, but we have not made it as useful as if we had just made single-threaded processors faster. We have a lot of work to do.

Since transistors are not getting much better, some think special-purpose processors are our only path forward.

Clearly, there has been this dance back and forth over time between general purpose and special purpose. For well-defined applications, special purpose can yield lots of performance, particularly when signal processing intensive. The question will always

Trying to retreat from technology to preserve old jobs didn't work in the Luddite era, and it's not going to work today.

be how general those things can be and how they compete against general-purpose machines. With a well-defined task, you can build hardware that does it very well.

The flip side of that is we're not done with software and new algorithms, and those are going to be much harder to put onto special-purpose pieces of hardware before we understand them.

Can we match the software to the hardware?

That's the hard problem. And it's one that we have made relatively little progress on. It requires probably both advances in compiler technology and also in how we think about programming those machines.

Impact of CS on Jobs

There is increasing concern about advances in information technology and what it is going to do to jobs.^a Changes were generational in the past, so you would do a job that would go away, but your children could learn another job. However, given the speed of change of information technology, jobs change within a single career.

A looming example is self-driving cars. It promises huge societal benefits—fewer deaths and injuries, helping the elderly or the disabled—but many jobs would go away: taxi cab drivers, car repair shops, car insurance agents. Should CS, academia, or society take action, or should we just let market forces run their course?

^a Vardi, M.Y. Is information technology destroying the middle class? *Commun. ACM* 58, 2 (Feb. 2015), 5.

Trying to retreat from technology to preserve old jobs didn't work back in the Luddite era, and it's not going to work today. You've got to figure out how you're going to retrain and redeploy people into new kinds of career opportunities. The interesting thing about self-driving cars is not only could we achieve some economic efficiency probably, but also we can achieve dramatic improvement in quality of life for everybody that's involved in it.

But I think it is important to think about technology. It is not that technology can't be abused; it can obviously be used for doing bad things as well as good things. We will face this problem with machine learning as AI gets better and better. The right solution is to figure out how to use it effectively and how to create value for society.

That will require us to retrain people and put them into new professions. Thomas Friedman once said, "The only way to keep up with the change in the world is to run faster. You can't run slower and try and retard the progress."

So we can't stop the change. Is there something that those of us that are inventing the technology that is changing the world can do to soften the impact of job displacement, or is that outside of our skill set?

It's hard to figure out exactly what role we have. Clearly, making education cheaper because in the end you're going to retrain people through education. For retraining functions, online education is the right way to go. So there's an opportunity to really try to get people involved in new things.

I can see why people like interviewing you; that's a good idea. If we're educators and there's going to be this massive displacement of jobs, we can either relieve the French revolution or we can help improve retraining.

Whither MOOCs

There's been this theme over your career about computing and education. You had faculty here a few years ago that said massive open online courses (MOOCs) were going to revolutionize campuses, offering Stanford degrees for a few dollars. That didn't happen. What do you think is going on with MOOCs?

MOOCs have both advantages and

disadvantages. They're certainly not the end-all and be-all answer. Their advantage, namely cost of production, comes from one course serving lots of students with very wide backgrounds and very little requirements in terms of prerequisites, but that's their disadvantage, too. It means that you've got a group of students who are very widely distributed in terms of ability.

For some of them, the course will be going too slow, for some it will be going way too fast, for some it will be too challenging, too easy. That's a really hard problem to solve. I think technology clearly has a role to play here. And if you said: well, I'm going to use it to teach moderate-sized courses of screened students, where they've had certain prerequisites, then I think there's a great role there and it's a way to probably get some increase in scale in our institutions.

There's another key role, which I think is critical here. In much of the developing world, there is just no access at all to higher education. You won't deliver access unless you can do it extremely inexpensively—there MOOCs have a real role to play. I think we have already seen the role that MOOCs can play in educating the educators in these institutions, as well as providing students without any other alternative with a route to education.

We have to improve the way we do things in higher education. Clearly, we've got to become more efficient over time, but to try to make a quantum leap to MOOCs is probably not the right way to do it.

My colleague Armando Fox and I were early MOOC adopters. Our attraction was the large international audience and solving the continuing education problem.

Continuing education; 100% this is going to go to online very quickly, whether it goes to MOOCs or large classes where students have similar backgrounds, but it's the only way to really solve two problems, namely people who are out there working are simply too busy to come to an institution. The time shifting that occurs and balancing people's schedules, so I think online will happen. I wouldn't be surprised to see professional master's degrees become a hybrid, partly online, partly experiential. In a field

Creative destruction is embraced in the Valley, and that's why things happen; the old makes way for the new.

like ours, you do want some interaction, teamwork, experimental work, this kind of thing.

Armando Fox says it is currently like early days of CS where everything sucked. We eventually made it better. This era is the early version of MOOC software, and things might get better.

The real Golden Fleece is to actually get a system that does adaptive learning, and really responds to how are people learning. You did this little quiz question, now what do you do? The big problem in education is, in some sense, motivation. How do you get students to really engage? Because simply looking at whether it's an instructor sitting in a classroom passively or sitting in front of a screen passively is not a learning experience. There is some good data out there. The data around so-called interactive active learning models does work because it engages students more, and that is what we've got to focus on. We've got to experiment, try things, and the academy is sometimes a little reluctant to experiment.

Critiques of His Presidency

As far as I can tell in your amazing 15-year record, the biggest criticism of your tenure—which appeared, for some reason, in *The New Yorker*—is that students leave to do startups and that you served on several corporate boards. Let me do these one at a time. There can't be very many undergraduate students leaving.

Essentially none.

Are graduate students leaving?

Essentially none. This was driven by one story about one company, where the student happened to be my advisee, and I told him to finish your degree and talk to your parents about what

you do. He did finish his degree. All his co-founders, all his startup team finished their degrees.

What's the story there?

We have a program through StartX, where students will get enough money to eat beans for the summer and try out their idea and see if it works. Most of them conclude: "Well, it's worth continuing working on, but it's not the hottest thing going." For undergraduates, we try to emphasize the value of their undergraduate degree and what that's going to mean in their long-term future, not their short-term future, and that's worked just fine.

It's a whole different situation with graduate students. If you are Larry Page and Sergey Brin or are Jerry Yang and David Filo, and you can create Google or Yahoo, you ought to go do that.

The second issue was where there were doubters even on your campus about whether a president should be serving on boards of corporations. If I was in your shoes, I'd think I'm going to be running this big organization, so if I can serve on boards and see how these companies deal with challenging problems, maybe I'll get some ideas.

Yes.

With the economic turmoils in 2007, you did cuts in advance of the big downturn at Stanford, whereas most other college presidents reacted after it happened.

Certainly from my experience from being on boards, I saw what was happening. I saw how much the endowment was going to be impacted. I have learned having gone through layoffs at MIPS that the best thing to do is to get through it as quickly as possible and reset the budget and reset the new normal and then move forward. That's what we tried to do and we recovered very fast. Things were over in a year or 18 months for Stanford. We began then a positive course forward.

Anything else?

One of the most important things I think I've imported from boards is to focus on leadership training and leadership preparation.

We don't do a good job of this in the university. We have an opening, and we find that we don't have anybody who has the right kind of background and experi-

ence. They haven't had the opportunity to be coached, develop some of these leadership skills, to look at big problems across the university.

Thus, we put in place a set of leadership programs to try to do that with the goal that we would always have at least one internal viable candidate for any position in the university.

What does university leadership training look like?

The big one we do is the leadership academy, that's our kind of premier program. It's taught by Chuck Holloway from the business school and John Morgridge, the former CEO of Cisco. It's everything from: How do you think about solving problems? How do you think about issues of diversity and equity in the university? Looking at various case studies, how do universities deal with public affairs crisis, but then also some personal growth, whether it's communications coaching or it's coaching on how do you deal with difficult problems, getting reviews, giving people feedback, those are all things you have to learn along the way. Learning them and having some exposure to them certainly helps people when they're in a position where they have to do it.

The Magic of Silicon Valley

Both of us came to the San Francisco Bay area in 1976–1977. I am just amazed at the success of this region. It has arguably the number-one private university and arguably the number-one public university. You've got the leading information and technology center in the world, one of the best wines regions, and even our professional sports teams do well.

I know we have nice weather and we're near the Pacific Ocean and the Sierras, but it's got to be more than climate and geography. Is it that our universities are attracting and training great people, and they stay there, the lure of Silicon Valley, or ...?

A couple of other things helped. This is a place that is very welcoming to people from around the world. No matter where you come from, you can probably find people who are similar to you in the Valley. It's not a perfect meritocracy, it's a ways from it—but it's a pretty good meritocracy.

Grading on the curve, it's pretty good.



John Hennessy and David Patterson circa 1991 with their textbook *Computer Architecture: A Quantitative Approach*, which was published the year before and is now in its fifth edition.

Yes. And the curve is very uneven, and it's not always equitable, and certainly there are things, biases and things that come in, but it's pretty good overall. It's also going back to the days of Hewlett and Packard; it tends to have organizations that are relatively flat. They tend to have open management styles.

That's the tradition?

The tradition of Hewlett and Packard and then Intel after that. It's a tradition in the Valley that everybody's got a role to play and everybody can make a contribution, even in a large company. You and I both came when if you wanted to talk to the movers and shakers in the computer industry, we had to get on a plane and fly to New York or fly to Boston.

Now it's the other way around. Creative destruction is embraced in the Valley, and that's why things happen; the old makes way for the new. In creating that capability, management and leadership talent as much as engineering talent becomes key. It's great to have a team of young engineers, but if you don't have anybody on that team that's ever delivered a product, they're challenged to figure out what the issues are and how do you make that work.

CS Popularity on Campus

We've seen an explosion in the popularity of CS everywhere; CS is now the biggest major on your campus. What is going to happen? CS faculty size is going to have to grow in response to demand, or since

it's a zero-sum game, only slow changes?

We're going to struggle to meet the demand. All institutions are facing this because CS graduates have lots of good opportunities, both within the academy as well as outside it. We're also going to see other fields begin to change, but change takes a long time. Universities change at a very slow pace getting new faculty.

However, I think we can already see the rise of quantitative empirical analytical approaches, for example take a discipline like political science. The action is in doing big data analysis of elections and surveys. It requires a different kind of expertise and background. We will slowly see the political science department starting to focus on training people. The next generation of young faculty they're bringing in—young meaning less than 50 years old—does a lot of this kind of work, right?

Sounds young to me.

I think we'll see a shift that we'll still have to teach them some basic CS, but maybe instead of being a major, they'll be a minor, and they'll do their work in this other discipline.

If computer scientists do not do the teaching, I worry about the quality of the courses over time. I think to do that you probably need more professors. Another way of asking this: Do you think schools of CS are going to get more popular on campuses, or will CS stay in college of

engineering?

I think it could stay. It's a question of how big can it be. They are now one of the biggest departments, but compared to a medical school, they are small. Maybe we need to rethink how that plays. CS can grow and merge over time.

Publishing: Limiting the LPU

There is a lot of concern in CS about publishing. When you see how many dozen papers new Ph.D.'s have, it looks like lots of LPU (least publishable unit), where quantity trumps quality. One proposed solution is to limit the number of papers that count on job applications or for promotion.^b Are we the only field with the problem? Is the proposed solution plausible?

Other fields do a lot of publication. Look at the biological sciences and look at the list of publications they have. They would even have more than many computer scientists. I do worry that in this drive to publish and the use of conferences that we have become a little

incremental. It's easy enough in the academic setting to say: tell me about the five or six things that you've done in preparation for tenure? What are the most important five or six papers? Because the reality is, nobody is going to look much deeper than that, if they look carefully. We need to move away from just counting the number of publications and saying: okay, that's enough publications. What has the person really done? In the end, it's about impact. What impact did they have, either on an industry or on other researchers; because that's what you really care about when you're trying to evaluate the research work of a faculty member.

So then plausibly having a policy that limits to what your five best works or something like that?

Yes. I don't think it would be a big issue.

Computer Science and X

From your view at the top of one of the great universities, what promising new fields should CS interact with?

One is really trying to crack the genome. What is this genome code for

and how does it match against human disease and other kinds of health problems? We have made progress, but there is a lot more to do.

We're seeing the era of big data creating interesting changes in the social sciences. People have recently studied life outcomes of people looking at the IRS database to see what determines what your economic prosperity is going to be over your life. Where you were born? Where you were educated? What kind of family you lived in? What kind of community you lived in? What kind of schools of thought? Trying to do that experiment as a social science experiment, where I take let's say two schools, I run method A, I run one method B, is very hard and it takes a long time.

On the other hand, if you can find two examples of schools that have been run by two different methods and look at the data historically, and look at what the outcomes are like, you can get insights that are impossible to get any other way. We're going to see a rise of the influence of big data in the social sciences, and they're going to become more quantitative in how we formulate policy.

^b Vardi, M.Y. Incentivizing quality and impact in computing research. *Commun. ACM* 58, 5 (May 2015), 5.

Master of COMPUTER SCIENCE. ONLINE.

Designed for working professionals with a bachelor's degree in computer science or engineering. Our in-demand graduate program offers you a 30-credit hour, non-thesis plan of study with several competitive specialization areas. Learn more today, www.proed.purdue.edu.

VIRTUAL CLASSROOM, PURDUE DEGREE.

WWW.CS.PURDUE.EDU/GRADUATE/ONLINE.HTML

PURDUE
UNIVERSITY.

Rising College Costs

Changing topics again, you're concerned about the rising cost of college tuition, as going faster than inflation just isn't sustainable. What are your thoughts about getting college tuition back under the rate of inflation?

The first thing to say is that college is still a good investment. If you look at the data, it is overwhelmingly clear that college is a good investment. As I told a group of parents recently, you're better off investing in your kid's education than you are investing in the stock market—the return is better.

In return to the parent?

In return to the family. Well, if you care about the economic outcome of your kids. What we've got to do is figure out a way to kind of balance costs better. We're going to have to find ways to figure out how to keep our costs under control and bend the cost curve a little bit. It doesn't take much, but it takes a little so that the growth is something that families can deal with.

Note this is driven as much by wage stagnation in the U.S. If salaries and wages in the U.S. were still going up at faster than inflation, which was traditional, it would remain affordable, but they're not, and so we've got this dilemma that we're going to have to solve.

Hennessy's Past and Future

Let's go back to high school. Were you class president or valedictorian?

I was kind of a science nerd. My big science fair project was building a tic-tac-toe machine with my friend Steve Angle out of surplus relays, because at that time real integrated circuits were too expensive. It had green and red lights for machine and person. Lots of people don't realize tic-tac-toe is a very simple game. When I brought it to see my future wife's family, they were really impressed, so it was a really good thing for me.

Speaking as a fellow person with Irish ancestry, the Irish have somewhat of a reputation of temper. You've had as far as I can tell a spotless public record; are you missing the Irish temper gene?

Once I did lose my temper, but haven't lost it since. That doesn't mean I can't use stern language with somebody;

You're better off investing in your kid's education than you are investing in the stock market—the return is better.

temper didn't work out, and so I haven't done it.

So you're too smart to lose your temper?

I certainly sometimes complain either to my longtime assistant or my wife. Sometimes I'll compose email, and I'll send it to the provost and say here's what I think. And he says you're right, but don't send that email. It's not going to be helpful. It's not going to solve our problem.

It just makes you feel better.

Yes.

I know you're smart, hard working, apparently indefatigable, down-to-earth, have common sense, are a good judge of people, and likeable: is there anything else that you need to be a successful administrator?

You need to listen.

Be a good listener.

You need to keep in touch with people. You need to hear what they're thinking. The people who really lead the university are the faculty in the end. Those of us that are in administrative roles are enablers, and we look for opportunities that can come by making connections across pieces of the institution. In order to do that job well, you really have to know what the faculty are thinking and what they are thinking about where we should go.

I spend a lot of time just interacting with faculty and listening to what they're thinking about, where they see an opportunity. The other great thing is that students are terrific, they're really great.

You've been spectacular at everything you've done in your professional career:

teacher, inventor, researcher, book author, entrepreneur, and administrator. You've been a role model to a lot of us. It made me wonder if I had challenged myself enough. (In fact, that's why I ran for ACM president.) This may be a strange question for Stanford's president, but have *you* ever wondered whether you have challenged yourself enough?

It's always a question: Are you willing to take on any new challenges? Are you worried about doing it? I don't think I'd be particularly good in government because I just don't quite have the patience. Being an academic leader requires some amount of patience, but being in government requires a lot more patience.

It takes even longer.

Having talked to people who do those jobs, I mean, whether it's the President or people who serve in Congress, that's really hard work, and it requires a real depth of patience and resilience. That's not my style; I mean I come from Silicon Valley. I'm a change person—and I wouldn't be a good caretaker either. If what the university needed was a caretaker president, I would have been the wrong person to pick because I wasn't going to find that job rewarding.

Let me wrap up. You have been called the most outstanding president of your generation. I see you as an intellectual descendant of two great California educators, Clark Kerr and Fred Terman. You have been a shining example for CS; all computer scientists benefit from the reflected glory from career. And you've been a wonderful colleague, co-author, and friend. You even disproved Leo Durocher's saying, by showing that nice guys can finish first. A grateful field thanks you for what you've accomplished and the style with which you've done it. ■

David Patterson (pattsrn@cs.berkeley.edu) is the E.H. and M.E. Pardee Chair of Computer Science at UC Berkeley and is a past president of ACM.

Copyright held by author.



Watch the authors discuss their work in this exclusive *Communications* video. <http://cacm.acm.org/videos/an-interview-with-stanford-university-president-john-hennessy>

For the full-length video, please visit <https://vimeo.com/146145543>

**A discussion with Amin Vahdat,
David Clark, and Jennifer Rexford.**

A Purpose-Built Global Network: Google's Move to SDN

EVERYTHING ABOUT GOOGLE is at scale, of course—a market cap of legendary proportions, an unrivaled talent pool, enough intellectual property to keep armies of attorneys in Guccis for life, and, oh yeah, a private WAN bigger than you can possibly imagine that also happens to be growing substantially faster than the Internet as a whole.

Unfortunately, bigger is not always better, at least not where networks are concerned, since along with massive size come massive costs, bigger management challenges, and the knowledge that traditional solutions probably are not going to cut it. And then there is this: specialized network gear does not come cheap.

Adding it all up, Google found itself on a cost curve it considered unsustainable. Perhaps even worse,

it saw itself at the mercy of a small number of network equipment vendors that have proved to be slow in terms of delivering the capabilities requested by the company. Which is why Google ultimately came to decide it should take more control of its own networking destiny. That is when being really, really big proved to be a nice asset after all, since being at Google scale means you can disrupt markets all on your own.

So this is the story of what Google ended up doing to get out of the box it found itself in with its backbone network. Spoiler alert: software-defined networks (SDNs) have played a major role. **Amin Vahdat**, Google's tech lead for networking, helps us tell the story of how that has all played out. He's both a Distinguished Engineer and a Google Fellow. Somehow he also manages to find time to teach at UC San Diego and Duke.

Jennifer Rexford, a computer science professor at Princeton renowned for her expertise in SDN, also contributes to the discussion, drawing on her early work designing SDN-like architectures deployed in AT&T's backbone network, as well as her recent research on novel programming abstractions for SDN controller platforms.

Finally, most of the questions that drive this discussion come courtesy of **David Clark**, the Internet pioneer who served as chief protocol architect of the network throughout the 1980s. He is also a senior research scientist at the MIT Computer Science and Artificial Intelligence Laboratory, where he has been working for nearly 45 years.

DAVID CLARK: I wonder if people have a full appreciation for the scale of your private wide area network.

AMIN VAHDAT: Probably not. I think our private-facing WAN is among the biggest in the world, with growth characteristics that actually outstrip the Internet. Some recent external measurements indicate that our backbone carries the equivalent of 10% of all the traffic on the global Internet. The rate

at which that volume is growing is faster than for the Internet as a whole.

This means the traditional ways of building, scaling, and managing wide area networks weren't exactly optimized or targeted for Google's use case. Because of that, the amount of money we had been allocating to our private WAN, both in terms of capital expenditures and operating expenses, had started to look unsustainable—meaning we really needed to get onto a different curve. So we started looking for a different architecture that would offer us different properties.

We actually had a number of unique characteristics to take into account there. For one thing, we essentially run two separate networks—a public-facing one and a private-facing one that connects our datacenters worldwide. The growth rate on the private-facing network has exceeded that on the public one; yet the availability requirements weren't as strict, and the number of interconnected sites to support was actually relatively modest.

In terms of coming up with a new architecture, from a traffic-engineering perspective, we quickly concluded that a centralized view of global demand would allow us to make better decisions more rapidly than would be possible with a fully decentralized protocol. In other words, given that we control all the elements in this particular network, it would clearly be more difficult to reconstruct a view of the system from the perspective of individual routing and switching elements than to look at them from a central perspective. Moreover, a centralized view could potentially be run on dedicated servers—perhaps on a number of dedicated servers, each possessing more processing power and considerably more memory than you would find with the embedded processors that run in traditional switches. So the ability to take advantage of general-purpose hardware became something of a priority for us as well. Those considerations, among many others, ultimately led us to an SDN architecture.



JENNIFER REXFORD: I would add that SDN offers network-wide visibility, network-wide control, and direct control over traffic in the network. That represents a significant departure from the way existing distributed control planes work, which is to force network administrators to coax the network into doing their bidding. Basically, what I think Google and some other companies find attractive about SDN is the ability to affect policy more directly from a single location with one view of the network as a whole.

DC: When did you first start looking at this?

AV: We started thinking about it in 2008, and the first implementation efforts probably kicked off in 2009, with the initial deployment coming in 2010.

DC: What were the features of SDN your engineers found most appealing as they were first trying to solve these problems back in 2008?

AV: Starting with the caveat that everything is bound to look a lot wiser in retrospect, I think the best way to answer that would be to talk about why we weren't satisfied with the prevailing architectures at the time. Our biggest frustration was that hardware and software were typically bundled together into a single platform, which basically left you at the mercy of certain vendors

to come up with any of the new features you needed to meet requirements already confronting you. So if we bought a piece of hardware from a vendor to handle our switching and routing, we would then also be dependent on that vendor to come up with any new protocols or software capabilities we might need later.

That was a huge issue for us since we already were playing in a high-end, specialized environment that required specialized platforms—meaning exorbitantly expensive platforms since the big vendors would quite naturally want to recoup their substantial engineering investments over the relatively small number of units they would have any hope of selling.

What's more, buying a bundled solution from a vendor meant buying all the capabilities any customer of that vendor might want, with respect to both hardware and software. In many cases, this was overkill for our use cases. I should probably add we initially were looking only to provide for high-volume but relatively low-value traffic. This probably helps explain why we didn't want to invest in totally bulletproof, ironclad systems that offered state-of-the-art fault tolerance, the most elaborate routing protocols, and all the other bells and whistles.

Over time, this evolved as we started moving higher-value traffic to the network. Still, our underlying philosophy remains: add support as necessary in the simplest way possible, both from a features and a management perspective.

Another big issue for us was that we realized decentralized protocols wouldn't necessarily give us predictability and control over our network, which at the time was already giving us fits in that convergence of the network to some state depended on ordering events that had already occurred across the network and from one link to another—meaning we had little to no control over the final state the system would wind up in. Certainly, we couldn't get to a “global optimum,” and beyond that, we couldn't even predict which of the many local optimums the system might converge to. This made network planning substantially harder. It also forced us to overprovision much more than we wanted. Now, mind you, I don't think any of these considerations are unique to Google.

And here's another familiar pain point that really bothered us—one I'm sure you'll have plenty of perspective on, Dave—and that is, we were tired of being at the mercy of the IETF (Internet Engineering Task Force) standardization process in terms of getting new functionality into our infrastructure. What we really wanted was to get to where we could write our own software so we would be able to get the functionality we needed whenever we needed it.

JR: The high-end equipment for transit providers not only has reliability mechanisms that might be more extensive than what was warranted for this particular network, but also offers support for a wide range of link technologies to account for all the different customers, peers, or providers a transit network might ever end up linking to. That's why you'll find everything in there, from serial links to Packet over SONET. Google's private WAN, on the other hand, is far more homogeneous, meaning there's really no need to support such a wide range of line-card technologies. Moreover, since there's no need for a private WAN to communicate with the global Internet, support for large routing tables was also clearly unnecessary. So, for any number of rea-

sons, the sorts of boxes the big carriers might be looking to purchase clearly would have been a poor fit for Google from the perspective of both line-card diversity and routing scalability.

DC: I can certainly see it wouldn't be cost effective to buy commercial high-end routers.

AV: What's interesting is that even Cisco and Juniper are now increasingly starting to leverage commodity silicon, at least for their lower-end datacenter products.

DC: Aren't you building your own routers?

AV: Well, they're routers in the sense they provide for external BGP (Border Gateway Protocol) peering, but they would never be mistaken for Cisco routers. Yet we've found we can achieve considerable cost savings by building just for what we need without taking on support for every single protocol ever invented.

DC: Also, there's that matter of centralized control. It's my sense that some people have an overly simplistic view of what SDN offers there, in that they imagine you have a bunch of routers and a centralized controller, but what you actually have is more sophisticated than that. In fact, as I understand it, there's a hierarchy of control in this network, with one controller for each site.

AV: That's correct.

DC: And that's not running a peer-to-peer distributed algorithm either. You get conceptually centralized control, but it's realized in a fairly sophisticated way. So that raises two questions. First, is that mental model generally in keeping with the level of complexity SDN is going to involve in practice? Also, just how much of that did you end up building yourself? My impression is you had to code a considerable amount of the controller, which seems like quite a price to pay to avoid getting trapped by standards.

AV: Yes, we built a huge amount of the infrastructure and wrote all the software. We also collaborated with some people externally. But I'd say we managed to do that with a moderate-size team—not small, but certainly nothing like a software team at a major vendor. Again, that's because we purpose-built our infrastructure.

Still, I would agree it's not a simple

system. Some of the complexity involved in maintaining hierarchical, multilevel control is inherent, given the need to isolate failure domains. I won't say SDN is necessarily simpler than the existing architectures, but I do think it offers some distinct advantages in terms of enabling rapid evolution, greater specialization, and increased efficiency.

JR: For all the talk about where this might lead, I notice that in the SIGCOMM paper where you describe this network [B4: Experience with a Globally Deployed Software-Defined WAN, 2013], you also talk about all the effort made to incorporate IS-IS (Intermediate System to Intermediate System) and BGP as part of the solution. That struck me as strange, given that each of the endpoints within the B4 network or connected to it is under Google's control—meaning you clearly could have chosen not to use any legacy protocols whatsoever. What value did you see in holding onto them?

AV: That actually was a critically important decision, so I'm glad you brought it up. We decided on an incremental deployment strategy after much consideration, and that's something we wanted to emphasize for the benefit of ISPs when we were writing that SIGCOMM paper.

The question was: Did we want to have a flag day where we flipped all our datacenters over to SDN in one fell swoop? Or did we want to do it one datacenter at a time while making it look like everything was just the same as ever to all the other sites? So you could say we ended up making huge investments just to re-create what we already had, only with a less mature system. That took quite a while.

Also, there was a fair amount of time where we had only baseline SDN—without any traffic engineering—deployed. Basically, that was the case throughout the whole period we were bringing up SDN one datacenter at a time. I still think that was the right approach since it gave us an opportunity to gain some much-needed experience with SDN.

So, while I agree that BGP and IS-IS are not where we want to be long term, they certainly have provided us with a critical evolution path to move from a non-SDN network to an SDN one.

DC: You're making some really im-



“

JENNIFER REXFORD

If SDN is going to prove successful in a much broader context ... it's going to be because there are reusable platforms available, along with the ability to build applications on top of those platforms.

”

portant points here. For a large ISP like Comcast, for example, the equivalent of doing one datacenter at a time might be focusing on just one metropolitan area at a time. But even just transitioning a single metropolitan area would be complicated enough, so it would be good to think in terms of approaching that incrementally such that they could always fall back to stuff known to work, like shortest-path routing.

AV: Oh, yes, I view that as critically important. You really need to have that sort of hybrid deployment model. In fact, even now we continue to have a big red button that lets us fall back to shortest-path routing should we ever feel the need to do so. And that's not even taking long-term considerations like backward compatibility into account. It's just that whenever you're deploying for any system as large and complex as the Internet—or our private WAN for that matter—it's really, really important to take a hybrid approach.

As with any pioneering effort, Google's push into software-defined networking has come with a number of risks—most notably the potential for breaking the Internet's time-honored fate-sharing principle, along with its established mechanism for distributed consensus.

For any network engineer schooled over the past few decades, this ought to be more than enough to set off alarms, since it has been long accepted that any scenario that could potentially lead to independent failures of the brain and body might easily result in bizarre failure patterns from which recovery could prove tremendously difficult. But now, after a careful reexamination of the current Internet landscape, it appears these are risks that could actually be mitigated through the combination of centralized control and a bit of clever traffic engineering.

DC: In rolling out the network, what was the biggest risk you faced?

AV: Probably what most concerned me was that we were breaking the fate-sharing principle—which is to say we were putting ourselves in a situation where either the controller could fail without the switch failing, or the switch could fail without the controller failing. That generally leads to big problems in distributed computing, as many people learned the hard way once remote procedure calls became a dominant paradigm.

DC: I find your comments about fate sharing somewhat amusing since back when we started doing the Internet,



AMIN VAHDAT

Through centralized traffic engineering and quality-of-service differentiation, we've managed to distinguish high-value traffic from the bulk traffic that's not nearly as latency-sensitive. That has made it possible to run many of our links at near 100% utilization levels.



we were quite critical of the telephone company because it didn't really have a good system for dynamic routing. So it would gold-plate all its technology and then run with these stupid, feeble routers that crashed all the time, since that basically was all that was available back then. Dynamic routing was supposed to give us the network resilience we would need to get away with running those crappy routers. But I think what we've learned is that dynamic routing might have been a good idea had the protocols actually proved responsive enough to let people make timely compensating engineering decisions.

In the early days of routing, however, we didn't know how to do any of that. We went with the distributed protocols for the simple reason that they were the only ones we knew how to build. What I mean is that this idea of breaking fate sharing was absolutely terrifying to us since we knew a partition in the network might separate the controller from the switches that needed to be managed.

Basically, if the controller were to lose its view of the network, then there would be no way to reach into the network and put it back together again. Back in those days we just didn't have any idea how to deal with that. That got

us started down the road to the original Internet religious holding that we don't have to make the switches expensively robust if we have a strategy for rebuilding the network once something breaks, assuming that can be done fast enough and effectively enough to let us restore the necessary services.

JR: We should also note that in addition to fate sharing, SDN is criticized for breaking distributed consensus, which is where the routers talk amongst themselves to reach agreement on a common view of network state. Anyway, the perception is that distributed consensus might end up getting broken since one or more controllers could get in the way.

But I would just like to say I think both of those battles have already been lost anyway—even before SDN became particularly prominent. That is, I think if you look closely at a current high-end router from Cisco or Juniper, you'll find they also employ distributed-system architectures, where the control plane might be running in a separate blade from the one where the data plane is running. That means those systems, too, are subject to these same problems where the brain and the body might fail independently.

DC: Another concern from the old

days is that whenever you have to rely on distributed protocols essentially to rebuild the network from the bottom up, you have to realize you might end up with a network that's not exactly the way you would want it once you've taken into account anything other than just connectivity. Basically, that's because we've never been very good at building distributed protocols capable of doing anything more than simply restoring shortest-path connectivity.

There was always this concern that knowledge of a failure absolutely had to be propagated to the controller so the controller could then respond to it. Mind you, this concern had nothing to do with unplanned transient failures, which I think just goes to show how little we anticipated the problems network managers would actually face down the road. But when you think about it, knowledge of unplanned transient failures really does need to be propagated. Part of what worried us was that, depending on the order in which things failed in the network, the controller might end up not being able to see all that had failed until it actually started repairing things.

That, of course, could lead to some strange failure patterns, caused perhaps by multiple simultaneous failures or possibly just by the loss of a component responsible for controlling several other logical components—leaving you with a Baltimore tunnel fire or something along those lines, where the controller has to construct the Net over and over and over again to obtain the topological information required to fix the network and restore it to its previous state. Is that an issue you still face with the system you now have running?

AV: Failure patterns like these were exactly what we were trying to take on. As you were saying, the original Internet protocols were focused entirely on connectivity, and the traditional rule of thumb said you needed to overprovision all your links by a factor of three to meet the requirements of a highly available “network fabric.” But at the scale of this particular network, multiplying all the provisioning by three simply was not a sustainable model. We had to find a way out of that box.

DC: That gets us back to the need to achieve higher network utilization. One of the things I find really inter-

esting and distinctive is how you've managed to exploit traffic engineering to achieve some very high link loadings. I've always had it in the back of my mind that by identifying classes of traffic, some of which are more tolerant of being slowed down than others, and then employing a bit of traffic engineering and quality of service, you ought to be able to get some higher link loadings just by knowing which traffic can be slowed down. To what extent is SDN actually necessary to accomplish that? Prior to this, it seemed that Google was using DiffServ tags, so I just assumed DiffServ tags could be used to increase link loading by ensuring latency-sensitive traffic didn't get disrupted. To what extent is traffic engineering dependent on moving to an SDN architecture or at least the SDN approach?

AV: That isn't dependent on SDN. There's nothing in that respect that couldn't have been achieved by some other means. I think it really comes down to efficiency and iteration speed. I should add that you were absolutely right in your supposition: DiffServ can indeed be used to increase link loading. Our main concern, though, had to do with failures, and we had no way of predicting how the system would converge. So the overprovisioning was always to protect the latency-sensitive—or, if you will, revenue-generating—traffic. Basically, for us to hit our SLAs (service-level agreements), that meant overprovisioning to cover worst-case convergence scenarios in a decentralized environment. Upon moving to a centralized environment, however, we found we could actually predict how things were going to converge under failure conditions, which meant we could get away with substantially less overprovisioning across our global network while still managing to hit our SLAs.

JR: Plus, you could control exactly what intermediate stage the network goes through when it transitions from one configuration to another, whereas if you let the distributed protocols do it, then all bets are off as to which router ends up going first.

AV: Exactly. So I think the total amount of improvement we realized through our centralized scheme relative to a decentralized scheme in

steady state actually proved to be relatively modest—let's say a 10%, maybe 15% improvement in the best case. What proved to be far more important was the predictability under failure, the improved ability to analyze failure conditions, and the means for transitioning the system from one state to another—again in a predictable manner that allowed for the protection of latency-sensitive traffic. That's what really made it possible for us to get away with less overprovisioning.

JR: Also, if you're using legacy protocols, even to the extent you can predict what they're going to do, the network management tools you use to make that prediction need essentially to invert the control plane so you can model what it's likely to do once it's poked and prodded in various ways. But with a centralized network, if you want to be able to predict what's going to happen when you perform planned maintenance or need to deal with some particular failure scenario, you can just run the exact same code the controller is going to run so you'll know exactly what's going to happen.

AV: To put this in some perspective, what we've really managed to accomplish is to lay some important groundwork. That is, I think we still have a long road ahead of us, but the traffic-engineering aspect is an important early step on that journey. It's one that drives a lot of capital-expenditure savings, and it's now also an architecture on top of which we'll be able to deliver new functionality more rapidly and under software control—which is to say, we'll be able to deliver that functionality under *our* control. We'll no longer have to wait for someone else to deliver critical functionality to us. Working in small teams, we should be able to deliver substantial functionality in just a matter of months in a tested, reproducible environment and then roll out that functionality globally. Ultimately, I think that's going to be the biggest win of all—and the first demonstration of that is traffic engineering.

Increased autonomy isn't the only win, of course. Significantly improved link loadings and the ability to scale quickly in response to increased demand are two other obvious advantages Google has already managed to realize with its

private backbone WAN. In fact, the experience so far with both SDN and centralized management has been encouraging enough that efforts are under way to take much the same approach in retooling Google's public-facing network. The challenges that will be encountered there, however, promise to be much greater.

DC: Getting right to the punchline, what do you see as the biggest improvements you've managed to achieve by going with SDN?

AV: Well, as we were saying earlier, through a combination of centralized traffic engineering and quality-of-service differentiation, we've managed to distinguish high-value traffic from the bulk traffic that's not nearly as latency-sensitive. That has made it possible to run many of our links at near 100% utilization levels.

DC: I think that comment is likely to draw some attention.

AV: Of course, our experience with this private-facing WAN hasn't been uniformly positive. We've certainly had our hiccups and challenges along the way. But, overall, it has exceeded all of our initial expectations, and it's being used in ways we hadn't anticipated for much more critical traffic than we had initially considered. What's more, the growth rate has been substantial—larger than what we've experienced with our public-facing network, in fact.

Now, given that we have to support all the different protocol checkbox features and line cards on our public-facing network, our cost structures there are even worse, which is why we're working to push this same approach—not the exact same techniques, but the general approach—into our public-facing network as well. That work is already ongoing, but it will surely be a long effort.

DC: What are some of the key differences between the public-facing Net and the private Net that you'll need to take into account?

AV: For one thing, as you can imagine, we have many more peering points in our public-facing network. Our availability requirements are also much higher. The set of protocols we have to support is larger. The routing tables we have to carry are substantially larger—certainly more than a million Inter-

net prefixes and millions of different advertisements from our peers, just for starters. So, basically, as we move from the private Net to the public one, the overall number of sites, the size of the traffic exchanges, the robustness required to talk to external peers, and the sorts of interfaces we have to support will all change substantially. That means the public Net is clearly a harder problem, but given the understanding we've gained from our experience with the private Net, I'd say that undertaking now looks far less daunting than it did a few years ago.

DC: Does this suggest any similar sort of transition for the big carriers?

AV: What I personally find exciting in that respect is the possibilities for what I call *SDN peering*. BGP takes a distrustful view of the world, but what if individual ISPs—or peers, if you will—decide they want to at least selectively open up some additional information about their networks dynamically? Looking at it naively, I think if they were to share some information about downstream traffic patterns, they would be able to make end-to-end transit times a lot faster and basically improve the user experience tremendously. By making it possible for the ISPs to use their more lightly loaded paths better, the carriers themselves would also benefit.

JR: In general, current routing is strictly destination-based and doesn't consider the nature of applications. You can imagine, then, that SDN might be a great way to let the recipient of traffic reach upstream to say, "No, drop this traffic," or "Rate limit this traffic," or "Route this traffic differently because I can tell you something about the best paths to reach me that the upstream party doesn't know about." Similarly, I might say, "Hey, I want this video traffic to take this other path," or "I want it to pass through this other box," which again is something that's hard to accomplish with today's destination-based forwarding.

AV: We've already talked to some customers who are interested in SDN-based peering, and I can tell you they're particularly interested in application-specific peering. They would like to be able to say, "Hey, I want my video traffic to go through this peer, while my non-video traffic goes through this other peer," either for performance or pricing reasons.

And that's just awkward to do right now.

DC: I think some evidence of how a different technology might enable a win-win here between what are otherwise adversarial interests would actually go a long way toward clarifying some of the business conversations currently going on.

JR: One of the challenges for ISPs such as Comcast or AT&T, should they decide they want to move to SDN, is that they have a lot fewer end nodes than Google does. A transit network really needs to carry full routes, if you will. Also, the big ISPs tend to have tremendous heterogeneity in their edge router equipment, and they don't upgrade everything at the same time either, so some of that equipment might be four or five years old, if not older.

Therefore, SDN deployments for the large carriers are going to be significantly more challenging than what Google has faced. I still think it's a promising direction they should pursue, but for various practical reasons it's just going to take longer for them to get there.

DC: Earlier you alluded to some of the traffic-engineering advantages you believe SDN offers. Can you go into a bit of detail about some of the specific challenges you were looking to solve in order to build a more cost-effective network, given your particular set of problems?

AV: As far as I can tell, the state of the art in network management still involves logging into individual network switches and managing them through a CLI (command-line interface). That just scales terribly in terms of people costs. It also scales horribly in terms of the myriad network interactions human beings need to keep track of inside their heads when it comes to how some action on one box might end up resulting in ripple effects across the whole network fabric.

DC: For somebody who hasn't actually lived in the network operations world, it would be really hard to understand just how bad that can actually be. The idea that people are still programming routers using CLIs is a little mind-boggling. And the very idea that human beings are expected to figure out the global consequences of what might happen if they should make one little fix here or another little fix there ... it's like we never escaped the 1980s!



DAVE CLARK

The idea that people are still programming routers using CLIs is a little mind-boggling. And the very idea human beings are expected to figure out the global consequences of what might happen if they should make one little fix here or another little fix there ... it's like we never escaped the 1980s!



I'm sure there are some traditional network engineers who take great pride in their ability to keep all that junk in their heads. In fact, I imagine there has been some resistance to moving to higher-level management tools for the same reason some people back in the day refused to program in higher-level programming languages—namely, they were sure they would lose some efficiency by doing so. But when it comes to SDN, I hear you saying the exact opposite—that you can actually become far more efficient by moving to centralized control.

AV: True, but change is always going to meet with a certain amount of resistance. One of the fundamental questions to be answered here has to do with whether truth about the network actually resides in individual boxes or in a centrally controlled infrastructure. You can well believe it's a radical shift for some network operators to come around to accepting that they shouldn't go looking for the truth in individual boxes anymore. But that hasn't been an issue for us since we've been fortunate enough to work with a talented—and tolerant—operations team at Google that's been more than willing to take on the challenges and pitfalls of SDN-based management.

Another interesting aspect of making the transition to SDN is that when things break, or at least don't work as you expect them to, unless you have a reasonable mental model of what the controller is trying to do, you might find it very difficult to diagnose what's going on. In fact, I think one of the advantages network operations people have now when they're working with these protocols they know so well is that—while they may have only a very limited view and so have difficulties diagnosing everything—they at least have a familiar mental model they can work from when they're trying to debug and diagnose problems. Whereas with SDN, whenever things go bump in the night, someone who wasn't involved in writing the software in the first place is probably going to find it a lot more difficult to debug things.

DC: This leads to a larger question I hear a lot of people asking now: Do network engineers need to be trained in computer science? Many aren't at this point. While it's one thing to go through the Cisco certification process, one might argue that in an SDN world people might need to pop up a level to master more general computer science concepts, particularly those having to do with distributed systems.

AV: I think that's probably a fair comment. But I also think there are lots of very talented network engineers out there who are fully capable of adapting to new technologies.

DC: That being said, I think most of those network engineers probably don't currently do a lot of software development. More likely, they just assume they have more of a systems-integration role. It's possible that in the fullness of time, the advocates of SDN will try to supply enough components so that people with systems-integration skills, as opposed to coding skills, will find it easier to use SDN effectively. But I wonder whether, at that point, the complexity of SDN will have started to resemble the complexity you've been trying to shed by stripping down your network. That is, I wonder whether the trade-off between writing your own code or instead taking advantage of something that already offers you plenty of bells and whistles is somehow inherent—meaning you won't be able to entirely escape that by migrating to SDN.

AV: I would argue that a lot of that has been driven by management requirements. I certainly agree that the Google model isn't going to work for everyone. One of the biggest reasons we've been able to succeed in this effort is because we have an operations team that's supportive of introducing new risky functionality.

With regard to your question about whether we'll truly be able to shed some of the complexity, I certainly hope so. By moving away from a box-centric view of network management to a fabric-centric view, we should be able to make things inherently simpler. Yet I think this also remains the biggest open question for SDN: Just how much progress will we actually realize in terms of simplifying operations management?

JR: I think it's natural the two highest-profile early successes of SDN—namely, as a platform for network virtualization and the WAN deployment effort we're talking about here—are both instances where the controller platform, as well as the application that runs on top of the controller, have been highly integrated and developed by the same people. If SDN is going to prove successful in a much broader context—one where you don't have a huge software development team at your dispos-

al as well as a supportive organization behind you—it's going to be because there are reusable platforms available, along with the ability to build applications on top of those platforms.

Just as important, you would want to believe that many of those apps could come from parties other than those responsible for creating the platforms. We're actually starting to see a lot of innovation in this area right now, with work happening on lots of different controller platforms and people starting to consider abstractions that ought to make it possible to build applications on top.

But even before that happens, there are things SDN brings that perhaps were not critical for Google but likely will prove useful in other settings. One is that SDN could make it possible to scale back on much of the heterogeneity in device interfaces. Many of the companies that work on enterprise network management employ armies of developers just so they can build device drivers that speak at the CLI level with lots of different switches, routers, firewalls, and so on, meaning that a gradual move toward a more standard and open interface for talking to devices ought to go far to reduce some of the low-level complexity of automating management.

Beyond that, I think Google's design demonstrates that if you can separate the distributed management of state required for your network control logic from the network control logic itself, you can avoid reinventing the wheel of how to do reliable distributed state management while also separating that from every single protocol. Basically, with each new protocol we design, we reinvent how to do distributed state management. But it turns out the distributed-systems community already has a number of really good reusable solutions for that.

DC: Part of what I'm taking away here is that not everything Google did with its private WAN is going to be readily transferable into other operator contexts. There are some good reasons why this approach was an especially good fit for Google, both with regard to Google's specific requirements and the particular skills it has on hand in abundant supply. Also, as Amin noted, it helps that Google has a business culture that's more tolerant when it

comes to following paths that initially put resilience and reliability at somewhat greater risk.

JR: But I think some of the same cost arguments will ultimately apply to large carriers as well as to many large enterprises, so that might end up serving as an impetus for at least some of those organizations collectively subsidizing the R&D required to develop a suitable suite of SDN products they then could use. Otherwise, they might find themselves on an unsustainable cost curve when it comes to the purchase and operation of new network equipment.

For example, if you look at other domains, like the cellular core, you again find back offices full of exorbitantly expensive equipment that's typically quite brittle. I think you find much the same thing in enterprise. Changes are clearly going to proceed more slowly in those settings since they face much more difficult deployment challenges, far stricter reliability requirements, and maybe even some harder scaling requirements. I mean, there's a good reason we're seeing SDN surface first in datacenters and private WANs. Just the same, I think CAPEX (capital expenditure) and OPEX (operating expense) are ultimately going to prove to be compelling arguments in these other settings as well.

AV: If you take it as inevitable, for example, that all video content is going to be distributed across the Internet at some point in the near future, then we're surely looking at some phenomenal network growth, which suggests the large carriers will at minimum soon become quite interested in seizing upon any CAPEX and OPEX savings they possibly can. 

Related articles on queue.acm.org

The Road to SDN

Nick Feamster, Jennifer Rexford, Ellen Zegura
<http://queue.acm.org/detail.cfm?id=2560327>

OpenFlow: A Radical New Idea in Networking

Thomas A. Limoncelli
<http://queue.acm.org/detail.cfm?id=2305856>

A Guided Tour through Data-center Networking

Dennis Abts, Bob Felderman
<http://queue.acm.org/detail.cfm?id=2208919>



**Thoughts on trust and merit
in software team culture.**

BY KATE MATSUDAIRA

The Paradox of Autonomy and Recognition

WHO DOESN'T WANT recognition for their hard work and contributions?

Early in my career I wanted to believe if you worked hard, and added value, you would be rewarded. I wanted to believe in the utopian ideal that hard work, discipline, and contributions were the fuel that propelled you up the corporate ladder. Boy, was I wrong.

You see, I started my career as a shy, insecure, but smart, programmer. I worked hard (almost every weekend), I wrote code for fun when I was not working on my work projects (still do, actually), and I was loyal and dedicated to my company.

Once, for six months, I worked on a project with four other people—and I felt like my contributions in terms of functionality and hours contributed were at the top of the group. So you can imagine my surprise when at our launch party, the GM of the group stood up and recognized “Josh and the other team members for their hard work.” I stood there stunned, thinking, What?!? How was it the GM was so out of touch with the team? Didn't our manager look at the check-ins and the problems being resolved? How did Josh, who had probably contributed the second-least amount to the project,

end up being the only person singled out from the group? Not me, not the most senior person on the project, but the guy who seemed to spend a lot of time talking to my boss and the other people on the project.

Many of us have been there, though. We work tirelessly, doing our very best to get things done on time, but somehow we get passed over—in favor of someone else whom you honestly believe did not contribute nearly as much as you did. What happened to meritocracy and being recognized for your work?

Fast forward to the present day. I work with a team of 18 amazing technologists, and it is my job to judge their performance. Only recently have I realized in many ways it is nearly impossible to do so.

There is no objective, quantifiable way of doing this at scale without resorting to micromanagement—which

is not something you even want to contemplate with a talented team. If this doesn't make sense to you, then let me describe various metrics people have suggested for judging performance (note that some of these are measurable, but others are more subjective):

► **Hours.** Ugh. I hate hour watchers. Writing code, at least for me, is like art, and when I am just not in the mood I can't force myself to get things done. To use time spent as a productivity measure does not fairly represent the creativity and mentation required of a developer. Beyond that, it's not really all that feasible to track hours in a highly virtual environment. I love that people on my team can work from home—or whatever environment where they do their best work (this is the reason we have “no meeting” days)—but how can I possibly track someone's hours if they aren't in the office? Just like beans, counting hours sucks. Don't do it.

► **Lines of code.** This measure is flawed for many reasons—from the mantra that “the best code is the lines you don't write” to the simple anecdotal fact that it once took me three days to write a single line of code, while another day I wrote more than 10,000 lines of code (although, admittedly, part of that count included some substantial cut and paste). And, of course, deleting code can be very productive, too.

► **Bug counts.** Quality is obviously important, but finding bugs in production belonging to developers who otherwise write great code is not rare. This metric is seriously flawed in a profound way—it does not take into account that your best developers often produce a moderate number of serious bugs because they have been entrusted to design and implement the most complicated and critical pieces of an application/system. Penalizing your best players for having highly impacting bugs is tantamount to rewarding mediocrity.

► **Features.** Functionality is key, since when it comes to contributions, the features built into or added to the product should be directly tied to customer value. Of course, judging on features can get complicated when multiple people contribute to one feature. Further, the details of the implementation can dramatically affect the effort and hours involved. For example, consider a recent project to add login to an existing site:

implementing the feature using interstitial pages would have taken a few hours; however, the design involved using lightboxes, which increased the complexity around security and added days to the project to accommodate. Even looking at functionality and features as a performance metric can be misleading if you don't dive into the technical details of the implementation and its trade-offs.

► **Maintainability.** It is difficult to measure and track something as subjective as writing solid, maintainable code—but anyone who has had to struggle with legacy spaghetti code will tell you that maintainability is worth the extra time for code that will involve long-term usage in production. Coders who spend the extra time to write highly robust, maintainable code are often penalized for it, simply because their true contributions will not be realized until years later.

► **Building skills and knowledge.** How do you measure the benefit of the time invested in learning a new technology well enough to use it effectively; or researching and choosing the proper tools to optimize your productivity; or making careful and deliberate strategic choices that ultimately make a project feasible and successful? Obviously, these are critically important steps to take, but an outside perspective might point out that a lot more work could have been accomplished in the same amount of time spent developing skills and acquiring knowledge.

► **Helping others.** Many programmers are great, not for the work they do but for the way they enable others to be great. Just having these people on the team makes everyone else better. Mentoring and selfless assistance to others are critical to building and preserving a highly productive and cohesive team, but quantifying an individual's role in such activities can be incredibly difficult, despite the reality of the contribution.

There are probably 101 more factors that could be used to judge programmers' achievements—including the way they present themselves (having a good attitude, for example), how dependable they are, or how often they contribute innovative ideas and solutions. Very few of these are objective, concrete factors that can be totaled up

and given a grade—and it is difficult to do so without diving into minute details or micromanaging a project.

Then How Do You Get Noticed?

If there is no reliable managerial measure of your contributions, how do you get noticed? Really it all comes down to one thing: trust.

Trust is like a currency. When managers give their team members autonomy and independence, they trust them to complete the assigned task, making wise and strategic decisions along the way, while proactively communicating problems long before they become problems. These managers are, in fact, investing their money in the team, and when they see the returns on that investment, they, just like any lucky investor, are quite pleased.

Trust, though, takes time, patience, and consistency. If you cannot build a relationship with your manager, all that work means nothing. For someone to invest in you, you have to show you are, in fact, an investment. Ask yourself these questions:

- Does your boss trust you?
- Do your team and your peers trust you?
- Have you done a good job to earn their trust?
- How would your peers describe you to someone else?
- How influential are you within your organization?

As a manager, I have been, at various times, very fond of a particular employee, but then noticed this person's peers did not care for him or her, or they held negative impressions of his or her performance. In these cases, given the trust level I have with my entire team, the opinions of the collective can easily outweigh personal preferences. Think of trust as a graph and each arc between the people you interact with as a weight—so when it comes to performance, those weights really matter.

Projects, products, performance, and companies are not just judged on their output, but on how they produce the output.

In the example project I recalled at the outset of this article, the one thing Josh did differently was he did not just do the work, but he also made sure management—my boss and the GM—

knew what our team was doing. In retrospect, he was the reason our project was singled out in an organization with so many people. At the time, I resented Josh; but now, many years later, I realize his contributions to our team were not just his code, but also his communication skills and the way he did his job.

As an aside, though, certain company cultures may reward Josh's approach more than others. The problem with some people like Josh is that over time they can optimize on "trust" and create a distorted view of their contributions. This is what I mean when I say "office politics"—and this is not good, either.

One of my very smart friends told me a story about joining one big company and meeting tons of super-smart, highly functional, and productive people who were all about creating trust with their superiors by being hyper-visible:

"They talked the most at meetings, they interrupted people, they sent extremely verbose emails at 3 A.M. detailing the minutia of a meeting that took place the previous day, they cc'd long lists of seemingly irrelevant but high-ranking people on their emails, etc. And their bosses loved them and they got the best reviews, etc. After meeting these individuals and being both amazed and disgusted by their shtick, it started to become clear to us that the whole culture self-selects to this type of person. It didn't take us long to understand why so much 'work' happens but so little gets done."

What Can You Do as a Manager?

As an employee, I want to be judged by my contributions and be part of a team that is a meritocracy. I also want autonomy and the ability to own substantial parts of a project and not have someone looking over my shoulder.

As a manager, I want to give recognition and praise to the people who deserve it, and I do not want to micromanage and spend my days being big brother.

This implies an implicit contract:

I will give you autonomy and independence, but it is your responsibility to share status and information with me.

For example, a team member once told me he had worked so hard and had really given it his best; from my view-

point, however, his progress was not up to the level of his teammates. When he was leaving the company, he told me all these things he had done, and I asked him, "Why didn't you share this with me before?" With that information I could have advised him to spend his time elsewhere on priorities that were more important to the business. His response: "I thought you would know." Don't make that mistake.

It is also important that as a manager you recognize improvement. This means understanding a person's strengths and weaknesses. If you observe someone's performance and see substantial improvements in one of that person's development areas, then that is definitely worth recognizing. For example, if you have an amazing engineer who is typically a poor communicator, but who then steps up and contributes not just great coding prowess to a project but also keeps other team members abreast of evolving risk factors, those sorts of achievements are worth praise.

Make sure you consider all the factors of a person's involvement in the organization. Take steps to ask good questions and solicit feedback from other members of the organization. Finally, let each person know your expectations around communication and progress.

And What Can You Do Now?

My conclusion from all of this is: If you want autonomy and the ability to own and control your own domain and projects, then it is your job to push information and build trust with your team members.

In other words, you need to learn and do the following:

► *Follow through.* Do what you say, and consistently deliver on your commitments.

► *Proactively communicate.* When a task takes longer than you thought, explain why.

► *Improve your communication skills.* In order for others to hear you, sometimes you have to hone the way you deliver your message.

► *Volunteer information.* Make an effort to explain vague or hard-to-understand ideas and concepts. Share the details of your decisions and diversions. This is also important when you make

mistakes—letting others know before they figure it out on their own will show ownership of the situation and can prevent misunderstandings later.

► *Be forthright and authentic with your feelings.* Even when you may hold a contrary opinion, communicate your thoughts (respectfully and with tact).

► *Don't talk behind the backs of others.* It is very difficult to build trust if someone knows you will say something negative about your boss, the company leadership, or another co-worker.

► *Be objective and neutral in difficult situations.* Learn how to be calm under pressure, and act as a diplomat resolving conflicts instead of causing them.

► *Show consistency in your behavior.* This is important not only in follow-through but also in eliminating any double standards that may exist.

► *Learn to trust your team members.* This is one of the most difficult goals to accomplish, but trust is a two-way street. Giving others the benefit of the doubt and learning how to work with them is essential to a strong mutual working relationship.

In turn, you may be lucky enough to have a good manager who will be able to ask you good questions and take the time to understand your contributions. If that is not your situation, then make sure you are sharing information with those around you, such as your peers, your boss, and other stakeholders.

Good leadership means keeping everyone on the same page. If you want independence, then it is on you to make sure people know what you are contributing. ■

Related articles on queue.acm.org

The Science of Managing Data Science

Kate Matsudaira

<http://queue.acm.org/detail.cfm?id=2767971>

Evolution of the Product Manager

Ellen Chisa

<http://queue.acm.org/detail.cfm?id=2683579>

Web Services and IT Management

Pankaj Kumar

<http://queue.acm.org/detail.cfm?id=1080876>

Kate Matsudaira (katemats.com) is the founder of her own company, Popforms. Previously she worked in engineering leadership roles at companies such as Decide (acquired by eBay), Moz, and Amazon.

Copyright held by author.
Publication rights licensed to ACM. \$15.00.

The “Leftover Principle” requires increasingly more highly skilled humans.

BY TOM LIMONCELLI

Automation Should Be Like Iron Man, Not Ultron

Q: DEAR TOM: A few years ago we automated a major process in our system administration team. Now the system is impossible to debug. Nobody remembers the old manual process and the automation is beyond what any of us can understand. We feel like we’ve painted ourselves into a corner. Is all operations automation doomed to be this way?

A: The problem seems to be this automation was written to be like Ultron, not Iron Man.

Iron Man’s exoskeleton takes the abilities that Tony Stark has and accentuates them. Tony is a smart, strong guy. He can calculate power and trajectory on his own. However, by having his exoskeleton do this for him, he can focus on other things. Of course, if he disagrees or wants to do something the program was not coded to do, he can override the trajectory.

Ultron, on the other hand, was intended to be fully autonomous. It did everything and was basically so complex that when it had to be debugged the only choice was (spoiler alert!) to destroy it.

Had the screenwriter/director Joss Whedon consulted me (and Joss, if you are reading this, you really should have), I would have found a way to insert the famous Brian Kernighan quote, “Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

Before we talk about how to prevent this kind of situation, we should discuss how we get into it.

The first way we get into this trap is by automating the easy parts and leaving the rest to be done manually. This sounds like the obvious way to automate things and, in fact, is something I generally encouraged until my awareness was raised by John Allspaw’s excellent two-part blog post “A Mature Role for Automation.”

You certainly should not automate the difficult cases first. What we learn while automating the easy cases makes us better prepared to automate the more difficult cases. This is called the “leftover principle.” You automate the easy parts and what is “left over” is done by humans.

In the long run this creates a very serious problem. The work left over for people to do becomes, by definition, more difficult. At the start of the process, people were doing a mixture of simple and complex tasks. After a while the mix shifts more and more toward the complex. This is a problem because people are not getting smarter over time. Moore’s Law predicts computers will get more powerful over time, but sadly there is no such prediction about people.

Another reason the work becomes more difficult is that it becomes rarer. Easier work, done frequently, keeps a person’s skills fresh and keeps us ready for the rare but difficult tasks.

Taken to its logical conclusion, this paradigm results in a need to employ impossibly smart people to do impossibly difficult work. Maybe this is why Google's recruiters sound so painfully desperate when they call about joining their SRE team.

One way to avoid the problems of the leftover principle is called the "compensatory principle." There are certain tasks that people are good at that machines do not do well. Likewise, there are other tasks that machines are good at that people do not do well. The compensatory principle says people and machines should each do what they are good at and not attempt what they do not do well. That is, each group should compensate for the other's deficiencies.

Machines do not get bored, so they are better at repetitive tasks. They do not sleep, so they are better at tasks that must be done at all hours of the night. They are better at handling many operations at once, and at operations that require smooth or precise motion. They are better at literal reproduction, access restriction, and quantitative assessment.

People are better at improvisation and being flexible, exercising judgment, and coping with variations in written material, perceiving feelings.

Let's apply this principle to a monitoring system. The monitoring system collects metrics every five minutes, stores them, and then analyzes the data for the purposes of alerting, debugging, visualization, and interpretation.

A person could collect data about a system every five minutes, and with multiple shifts of workers they could do it around the clock. However, the people would become bored and sloppy. Therefore, it is obvious the data collection should be automated. Alerting requires precision, which is also best done by computers. However, while the computer is better at visualizing the data, people are better at interpreting those visualizations. Debugging requires improvisation,



another human skill, so again people are assigned those tasks.

John Allspaw points out that only rarely can a project be broken down into such clear-cut cases of functionality this way.

Doing Better

A better way is to base automation decisions on the complementarity principle. This principle looks at automation from the human perspective. It improves the long-term results by considering how people's behavior will change as a result of automation.

For example, the people planning the automation should consider what is learned over time by doing the process manually and how that would be changed or reduced if the process was automated. When a person first learns a task, they are focused on the basic functions needed to achieve the goal. However, over time, they understand the ecosystem that surrounds the process and gain a big-picture view. This lets them perform global optimizations. When a process is automated the automation encapsulates learning thus far, permitting new people to perform the task without having to experience that learning. This stunts or prevents future learning. This kind of analysis is part of a cognitive systems engineering (CSE) approach.

The complementarity principle combines CSE with a joint cognitive system (JCS) approach. JCS examines how automation and people work together. A joint cognitive system is characterized by its ability to stay in control of a situation.

In other words, if you look at a highly automated system and think, "Isn't it beautiful? We have no idea how it works," you may be using the leftover principle. If you look at it and say, "Isn't it beautiful how we learn and grow together, sharing control over the system," then you have done a good job of applying the complementarity principle.

Designing automation using the complementarity principle is a relatively new concept and I admit I am no expert, though I can look back at past projects and see where success has come from applying this principle by accident. Even the blind squirrel finds some acorns!



The compensatory principle says people and machines should each do what they are good at and not attempt what they do not do well. That is, each group should compensate for the other's deficiencies.



For example, I used to be on a team that maintained a very large (for its day) cloud infrastructure. We were responsible for the hundreds of physical machines that supported thousands of virtual machines.

We needed to automate the process of repairing the physical machines. When there was a hardware problem, virtual machines had to be moved off the physical machine, the machine had to be diagnosed, and a request for repairs had to be sent to the hardware techs in the data center.

After the machine was fixed, it needed to be reintegrated into the cloud.

The automation we created abided by the complementarity principle. It was a partnership between human and machine. It did not limit our ability to learn and grow. The control over the system was shared between the automation and the humans involved.

In other words, rather than creating a system that took over the cluster and ran it, we created one that partnered with humans to take care of most of the work. It did its job autonomously, but we did not step on each other's toes.

The automation had two parts. The first part was a set of tools the team used to do the various related tasks. Only after these tools had been working for some time did we build a system that automated the global process, and it did so more like an exoskeleton assistant than like a dictator.

The repair process was functionally decomposed into five major tasks, and one tool was written to handle each of them. The tools were:

- ▶ *Evacuation*: any virtual machines running on the physical machine needed to be migrated live to a different machine;
- ▶ *Revivification*: an evacuation process required during the extreme case where a virtual machine had to be restarted from its last snapshot;
- ▶ *Recovery*: attempts to get the machine working again by simple means such as powering it off and on again;
- ▶ *Send to Repair Depot*: generate a work order describing what needs to be fixed and send this information to the data center technicians who actually fixed the machine; and
- ▶ *Reassimilate*: once the machine has been repaired, configure it and reintroduce it to the service.

As the tools were completed, they replaced their respective manual processes. However, the tools provided extensive visibility as to what they were doing and why.

The next step was to build automation that could bring all these tools together. The automation was designed based on a few specific principles:

- It should follow the same methodology as the human team members.
- It should use the same tools as the human team members.
- If another team member was doing administrative work on a machine or cluster (group of machines), the automation would step out of the way if asked, just like a human team member would.

► Like a good team member, if it got confused it would back off and ask other members of the team for help.

The automation was a state-machine-driven repair system. Each physical machine was in a particular state: normal, in trouble, recovery in progress, sent for repairs, being reassembled, and so on. The monitoring system that would normally page people when there was a problem instead alerted our automation. Based on whether the alerting system had news of a machine having problems, being dead, or returning to life, the appropriate tool was activated. The tool's result determined the new state assigned to the machine.

If the automation got confused, it paused its work on that machine and asked a human for help by opening a ticket in our request tracking system.

If a human team member was doing manual maintenance on a machine, the automation was told to not touch the machine in an analogous way to how human team members would be, except people could now type a command instead of shouting to their co-workers in the surrounding cubicles.

The automation was very successful. Previously, whoever was on call was paged once or twice a day. Now we were typically paged less than once a week.

Because of the design, the human team members continued to be involved in the system enough so they were always learning. Some people focused on making the tools better. Others focused on improving the software release and test process.

As stated earlier, one problem with the leftover principle is the work left over for humans requires increasingly higher skill levels. At times we experienced the opposite! As the number of leftover tasks was reduced, it was easier to wrap our brains around the ones that remained. Without the mental clutter of so many other tasks, we were better able to assess the remaining tasks. For example, the most highly technical task involved a particularly heroic recovery procedure. We reevaluated whether or not we should even be doing this particular procedure. We shouldn't.

The heroic approach risked data loss in an effort to avoid rebooting a virtual machine. This was the wrong priority. Our customers cared much more about data loss than about a quick reboot. We actually eliminated this leftover task by replacing it with an existing procedure that was already automated. We would not have seen this opportunity if our minds had still been cluttered with so many other tasks.

Another leftover process was building new clusters or machines. It happened infrequently enough that it was not worthwhile to fully automate. However, we found we could Tom Sawyer the automation into building the cluster for us if we created the right metadata to make it think all the machines had just returned from repairs. Soon the cluster was built for us.

Processes requiring ad hoc improvisation, creativity, and evaluation were left to people. For example, certifying new models of hardware required improvisation and the ability to act given vague requirements.

The resulting system felt a lot like Iron Man's suit: enhancing our skills and taking care of the minutiae so we could focus on the big picture. One person could do the work of many, and we could do our jobs better thanks to the fact that we had an assistant taking care of the busy work. Learning did not stop because it was a collaborative effort. The automation took care of the boring stuff and the late-night work, and we could focus on the creative work of optimizing and enhancing the system for our customers.

I do not have a formula that will always achieve the benefits of the

complementarity principle. However, by paying careful attention to how people's behavior will change as a result of automation and by maintaining shared control over the system, we can build automation that is more Iron Man, less Ultron. 

Further Reading

"A Mature Role for Automation," J. Allspaw; <http://www.kitchensoap.com/2012/09/21/a-mature-role-for-automation-part-i>.

Joint Cognitive Systems: Foundations of Cognitive Systems Engineering, by D. Woods and E. Hollnagel, Taylor and Francis, Boca Raton, FL, 2005.

Chapter 12. *The Practice of Cloud System Administration*, by T. A. Limoncelli, S.R. Chalup, and C.J. Hogan; <http://the-cloud-book.com>.

Related articles on queue.acm.org

Weathering the Unexpected

Kripa Krishnan

<http://queue.acm.org/detail.cfm?id=2371516>

Swamped by Automation

George Neville-Neil

<http://queue.acm.org/detail.cfm?id=2440137>

Automated QA Testing at EA: Driven by Events

Michael Donat, Jafar Husain, and Terry Coatta

<http://queue.acm.org/detail.cfm?id=2627372>

Thomas A. Limoncelli is a site reliability engineer at Stack Exchange, Inc., in NYC. His books include *The Practice of Cloud Administration and Time Management for System Administrators*. His Everything Sysadmin column appears in *acmqueue* (<http://queue.acm.org>); he blogs at EverythingSysadmin.com.

DOI:10.1145/2812803

To encourage repeatable research, fund repeatability engineering and reward commitments to sharing research artifacts.

BY CHRISTIAN COLLBERG AND TODD A. PROEBSTING

Repeatability in Computer Systems Research

IN 2012, WHEN reading a paper from a recent premier computer security conference, we came to believe there is a clever way to defeat the analyses asserted in the paper, and, in order to show this we wrote to the authors (faculty and graduate students in a highly ranked U.S. computer science department) asking for access to their prototype system. We received no response. We thus decided to reimplement the algorithms in the paper but soon encountered obstacles, including a variable used but not defined; a function defined but never used; and a mathematical formula that did not typecheck. We asked the authors for clarification and received a single response: “I unfortunately have few recollections of the work ...”

We next made a formal request to the university for the source code under the broad Open Records Act (ORA) of the authors’ home state. The university’s

legal department responded with: “We have been unable to locate a confirmed instance of [system’s] source code on any [university] system.”

Expecting a research project of this magnitude to be developed under source code control and properly backed up, we made a second ORA request, this time for the email messages among the authors, hoping to trace the whereabouts of the source code. The legal department first responded with: “... the records will not be produced pursuant to [ORA sub-clause].” When we pointed out reasons why this clause does not apply, the university relented but demanded \$2,263.66 “... to search for, retrieve, redact and produce such records.” We declined the offer.

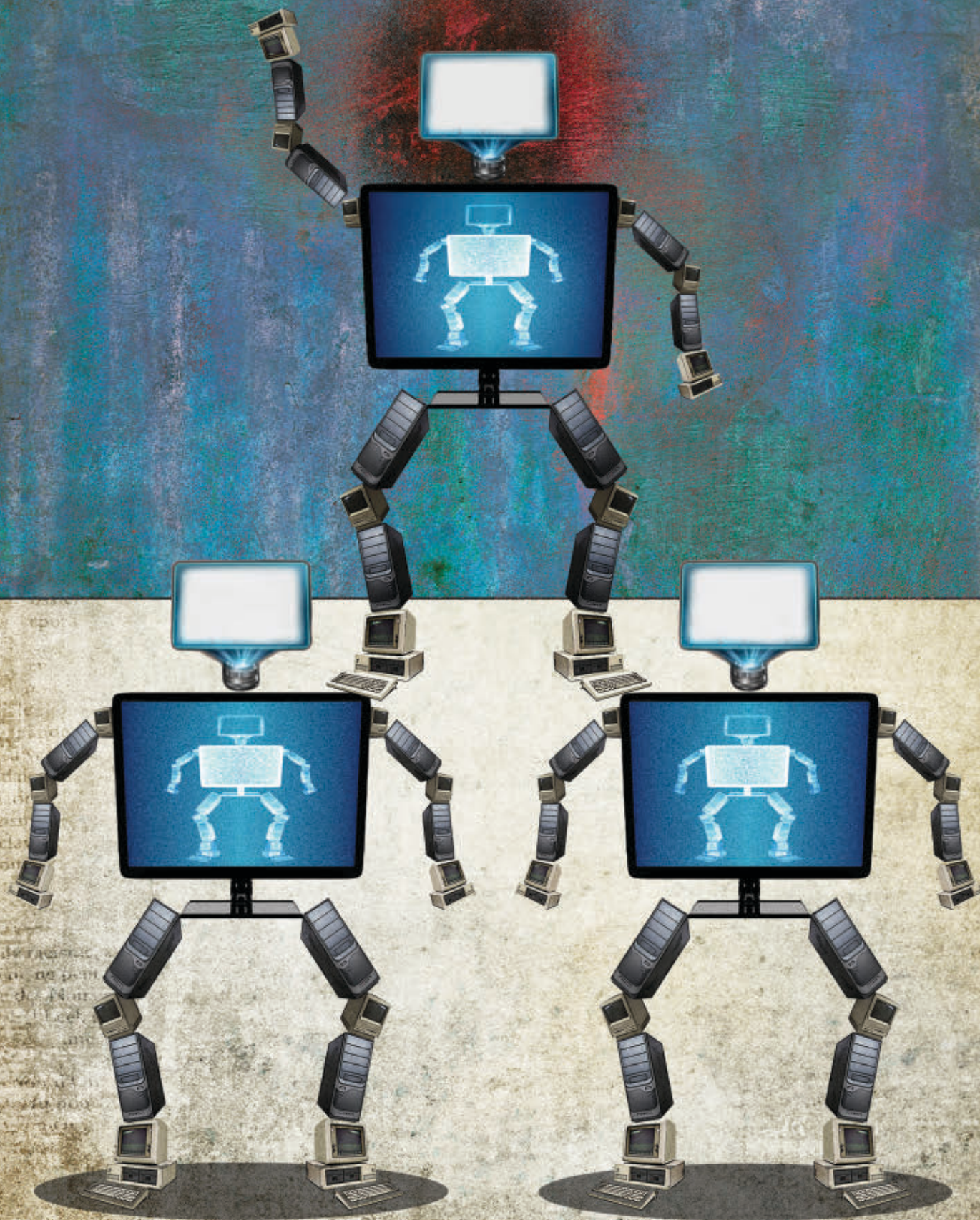
We instead made a Freedom of Information Act request to the National Science Foundation for the funded grant proposals that supported the research. In one, the principal investigator wrote, “We will also make our data and software available to the research community when appropriate.” In the end, we concluded, without assistance from the authors to interpret the paper and with the university obstructing our quest for the source code of the prototype system, we would not be able to show the analyses put forth could be defeated.

Reproducibility, repeatability, benefaction. There are two main reasons to share research artifacts: repeatability and benefaction.^{2,10,16,20} We say research is repeatable if we can re-run

» key insights

- **Published computer systems research is not always accompanied by the code that supports the research, which impedes peers’ ability to repeat the experiments.**
- **Sharing research software presents many challenges, so funding agencies should provide support for the engineering resources necessary to enable repeatable research.**
- **To incentivize authors to share their research artifacts, publishers should require pre-publication declarations from authors specifying their commitment to sharing code and data.**

ILLUSTRATION BY ANTHONY FREDA



the researchers' experiment using the same method in the same environment and obtain the same results.¹⁹ Sharing for repeatability is essential to ensure colleagues and reviewers can evaluate our results based on accurate and complete evidence. Sharing for benefaction allows colleagues to build on our results, better advancing scientific progress by avoiding needless replication of work.

Unlike repeatability, reproducibility does not necessarily require access to the original research artifacts. Rather, it is the independent confirmation of a scientific hypothesis,¹⁹ done post-publication, by collecting different properties from different experiments run on different benchmarks, and using these properties to verify the claims made in the paper. Repeatability and reproducibility are cornerstones of the scientific process, necessary for avoiding dissemination of flawed results.

In light of our discouraging experiences with sharing research artifacts, we embarked on a study to examine the extent to which computer systems researchers share their code and data, reporting the results here. We also make recommendations as to how to improve such sharing, for the good of both repeatability and benefaction.

The study. Several hurdles must be cleared to replicate computer systems research. Correct versions of source code, input data, operating systems, compilers, and libraries must be available, and the code itself must build

and run to completion. Moreover, if the research requires accurate measurements of resource consumption, the hardware platform must be replicated. Here, we use the most liberal definitions of repeatability: Do the authors make the source code used to create the results in their article available, and will it build? We will call this “weak repeatability.”

Our study examined 601 papers from ACM conferences and journals, attempting to locate any source code that backed up published results. We examined the paper itself, performed Web searches, examined popular source-code repositories, and, when all else failed, emailed the authors. We also attempted to build the code but did not go so far as trying to verify the correctness of the published results.

Recommendations. Previous work on repeatability describes the steps that must be taken in order to produce research that is truly repeatable^{11,12} or describes tools or websites that support publication of repeatable research.^{4,6} Our recommendations are more modest. We recognize that, as a discipline, computer science is a long way away from producing research that is always, and completely, repeatable. But, in the interim, we can require authors to conscientiously inform their peers of their intent with respect to sharing their research artifacts. This information should be provided by the authors when submitting their work for publication; this would allow reviewers to

take the expected level of repeatability into consideration in their recommendation to accept or reject. To this end, we make a recommendation for adding sharing contracts to publications—a statement by authors as to the level of repeatability readers can expect.

Background

Three previous empirical studies explored computer science researchers' willingness to share code and data. Kovačević⁵ rated 15 papers published in the *IEEE Transactions on Image Processing* and found that while all algorithms had proofs, none had code available, and 33% had data available. Vandewalle et al.¹⁸ examined the 134 papers published in *IEEE Transactions on Image Processing* in 2004, finding “... code (9%) and data (33%) are available online only in a minority of the cases ...” Stodden¹⁵ reported while 74% of the registrants at the Neural Information Processing Systems (machine-learning) conference said they were willing to share post-publication code and 67% post-publication data, only “... 30% of respondents shared some code and 20% shared some data on their own websites.” The most common reasons for not sharing code were “The time it takes to clean up and document for release,” “Dealing with questions from users about the code,” “The possibility that your code may be used without citation,” “The possibility of patents, or other IP constraints,” and “Competitors may get an advantage.” Stodden¹⁴ has since proposed “The Open Research License,” which, if universally adopted, would incentivize researchers to share by ensuring “... each scientist is attributed for only the work he or she has created.”¹³

Public repositories can help authors make their research artifacts available in perpetuity. Unfortunately, the “if you build it they will come” paradigm does not always work; for example, on the RunMyCode¹⁷ and ResearchCompendia Web portals,^a only 143 and 236 artifacts, respectively, had been registered as of January 2016.

One attractive proposition for researchers to ensure repeatability is to bundle code, data, operating system,

a <http://RunMyCode.org> and <http://research-compendia.org>

Table 1. Notation used in Table 2 and the figure.

Notation	Number of papers ...
HW	excluded due to replication requiring special hardware
NC	excluded due to results not being backed by code
EX	excluded due to overlapping author lists
BC	where the results are backed by code
Article	where code was found in the paper itself
Web	where code was found through a Web search
EM ^{yes}	where the author provides code after receiving an email message
EM ^{no}	where the author responds to an email message saying code cannot be provided
EM ⁰	where the author does not respond to email requests within two months
OK ^{≤30}	where code is available and we succeed in building the system in ≤30 minutes
OK ^{>30}	where code is available and we succeed in building the system in >30 minutes
OK ^{Auth}	where code is available and we fail to build, and the author says the code builds with reasonable effort
Fails	where code is available and we fail to build, and the author says the code may have problems building

and libraries into a virtual machine image.^{4,9} However, this comes with its own problems, including how to perform accurate performance measurements; how to ensure the future existence of VM monitors that will run my VM image; and how to safely run an image that contains obsolete operating systems and applications to which security patches may have not been applied.

From 2011 until January 2016, 19 computer science conferences^b participated in an “artifact evaluation process.”^c Submitting an artifact is voluntary, and the outcome of the evaluation does not influence whether or not a paper is accepted for publication; for example, of the 52 papers accepted by the 2014 Programming Language Design and Implementation (PLDI) conference, 20 authors submitted artifacts for evaluation, with 12 classified as “above threshold.”^d For PLDI 2015, this improved to 27 accepted artifacts out of 58 accepted papers, reflecting an encouraging trend.

Study Process

Our study employed a team of undergraduate and graduate research assistants in computer science and engineering to locate and build source code corresponding to the papers from the latest incarnations of eight ACM conferences (ASPLOS’12, CCS’12, OOPSLA’12, OSDI’12, PLDI’12, SIGMOD’12, SOSP’11, and VLDB’12) and five journals (TACO’12, TISSEC’12/13, TOCS’12, TODS’12, and TOPLAS’12).^e

We inspected each paper and removed from further consideration any that reported on non-commodity hardware or whose results were not backed by code. For the remaining papers we searched for links to source code by looking over the paper itself, examining the authors’ personal websites, and searching the Web and code repositories (such as GitHub, Google Code, and SourceForge). If still unsuccessful, we sent an email request to the authors, excluding some papers to avoid sending each author more than one request.

^b <http://evaluate.inf.usi.ch/artifacts>

^c <http://www.artifact-eval.org>

^d <http://pldi14-aec.cs.brown.edu>

^e See Collberg et al.¹ for a description of the process through which the study was carried out.

Repeatability and reproducibility are cornerstones of the scientific process, necessary for avoiding dissemination of flawed results.

We sent each request to all authors for whom we could determine an address and reminder email messages to those who did not respond.

In the following cases we marked a paper as “code not available” when we found only partial code or binary releases; when the authors promised they would “send code soon” but we heard nothing further; when we were asked to sign a license or non-disclosure agreement; when the authors requested credit for any follow-up work; or when we received code more than two months after the original email request.

We next made two attempts to build each system. This often required editing makefiles and finding and installing specific operating system and compiler versions, and external libraries. We first gave a research assistant a 30-minute time limit, and, if that failed, we gave another assistant “unlimited time” to attempt the build.^f

Upon completing the build process we conducted an online survey of all authors to help verify the data we had gathered. We resolved cases where we had misclassified a paper, where our Web searches had turned up the wrong code, or where there had been a misunderstanding between us and the authors. We also asked the authors if the version of the code corresponding to the results in their papers was available and (in cases where we had failed to build their code) if they thought the code ought to build. The survey also let the authors comment on our study.

Study Results

We define three measures of weak repeatability—weak repeatability A, B, and C—with notation we outline in Table 1:

$$\frac{OK^{\leq 30}}{BC} \quad (A)$$

$$\frac{OK^{\leq 30} + OK^{> 30}}{BC} \quad (B)$$

$$\frac{OK^{\leq 30} + OK^{> 30} + OK^{Auth}}{BC} \quad (C)$$

^f A group of independent researchers set out to verify our build results through a crowd-sourced effort; <http://cs.brown.edu/~sk/Memos/Examining-Reproducibility>

Weak repeatability A models scenarios where limited time is available to examine a research artifact, and when communicating with the author is not an option (such as when reviewing an artifact submitted alongside a conference paper). Weak repeatability B models situations where ample time is available to resolve issues, but the lead developer is not available for consultation. The latter turns out to be quite common. We saw situations where the student responsible for development had graduated, the main developer had passed away, the authors' email addresses no longer worked, and the authors were too busy to provide assistance. Weak repeatability C measures the extent to which we were able to build the code or the authors believed their code builds with reasonable effort. This model approximates a situation where ample time is available to examine the code and the authors are responsive to requests for assistance.

The results of our study are listed in Table 2 and outlined in the figure here, showing repeatability rates of A=32.3%, B=48.3%, and C=54.0%. Here, C is limited by the response rate to our author survey, 59.5%.

Does public funding affect sharing? The National Science Foundation Grant Proposal Guide⁷ says, "Investigators and grantees are encouraged to share software and inventions created under the grant or otherwise make them or their products widely available and usable." However, we did not find significant differences in the weak repeatability rates of NSF-funded vs. non-NSF-funded research.^g

Does industry involvement affect sharing? Not surprisingly, papers with authors only from industry have a low rate of repeatability, and papers with authors only from academic institutions have a higher-than-average rate. The reasons joint papers also have a lower-than-average rate of code sharing is not immediately obvious; for instance, the industrial partner might have imposed intellectual-property restrictions on the collaboration, or the research could be the result of a student's summer internship.

^g We tracked only whether papers mentioned NSF support; it is possible some might have had other sources of public funding.



We noticed published code does not always correspond to the version used to produce the results in the corresponding paper.



Does the right version exist? From the responses we received from authors, we noticed published code does not always correspond to the version used to produce the results in the corresponding paper. To see how common this is, in our author survey we asked, "Is your published code identical to the version you ran to get the results in the paper (ignoring inconsequential bug fixes)?" It was encouraging to see that out of the 177 responses to this question, 83.1% answered "yes," 12.4% answered "No, but it is possible to make that version available," and only 4.5% answered, "No, and it is not possible to make that version available."

Why is code not shared? The email responses we received were generally pleasant, accommodating, and apologetic if code could not be provided. In the following paragraphs, we explore several representative examples of email responses from authors who turned down our request.

In order for research to be truly repeatable, the correct version of all artifacts must be available, which is not always the case; for example, one respondent said, "I'm not very sure whether it is the final version of the code used in our paper, but it should be at least 99% close."

Authors often told us once their code was cleaned up we could have access to their system, in one case saying, "Unfortunately the current system is not mature enough at the moment, so it's not yet publicly available. We are actively working on a number of extensions and things are somewhat volatile." Eventually making (reworked) code available may be helpful for benefaction, but for repeatability, such delayed releases are ineffectual; it will never be possible for a reviewer or reader to verify the results presented in the paper.

Several authors acknowledged they never had the intention to make the code available, in one case saying, "I am afraid that the source code was never released. The code was never intended to be released so is not in any shape for general use."

In some cases, the one person who understood the system had left, with one respondent saying, "For the paper we used a prototype that included many moving pieces that only [student] knew how to operate and we did not have the

time to integrate them in a ready-to-share implementation before he left.”

Lack of proper backup procedures was also a problem, with one respondent saying, “Unfortunately, the server in which my implementation was stored had a disk crash in April and three disks crashed simultaneously ... my entire implementation for this paper was not found ... Sorry for that.”

Researchers employed by commercial entities were often not able to release their code, with one respondent saying, “The code owned by [company], and AFAIK the code is not open-source.” This author added this helpful suggestion: “Your best bet is to reimplement :(Sorry.”

Even academic researchers had licensing issues, with one respondent saying, “Unfortunately, the [system] sources are not meant to be opensource [sic] (the code is partially property of [three universities]).” Some universities put restrictions on the release of the code, with one respondent saying, “... we are making a collaboration release available to academic partners. If you’re interested in obtaining the code, we only ask for a description of the research project

that the code will be used in (which may lead to some joint research), and we also have a software license agreement that the University would need to sign.”

Some systems were built on top of other systems that were not publicly

available, with one respondent saying, “We implemented and tested our ... technique on top of a commercialized static analysis tool. So, the current implementation is not open to public. Sorry for this.” And, some systems were built

Summary of the study’s results. Blue numbers represent papers we excluded from the study, green numbers papers we determined to be weakly repeatable, red numbers papers we determined to be non-repeatable, and orange numbers represent papers for which we could not conclusively determine repeatability (due to our restriction of sending at most one email request per author).

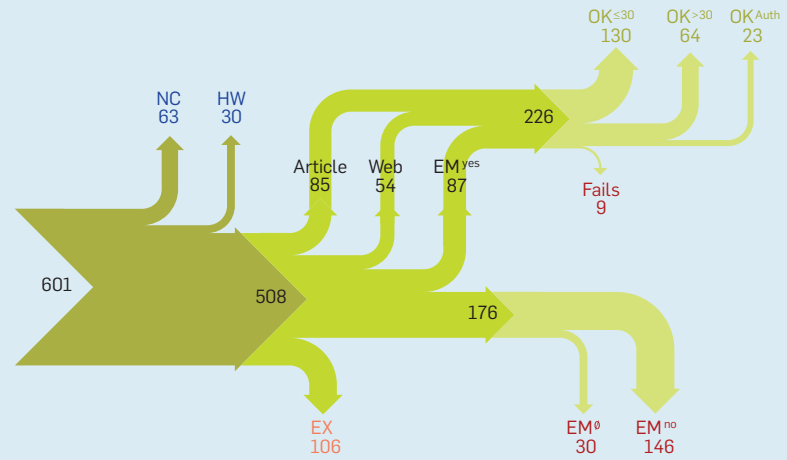


Table 2. Detailed results of the study.

Group	#	Classification				Code Location					Build Result				Weak Repeatability (%)		
		HW	NC	EX	BC	Article	Web	EM ^{yes}	EM ^{no}	EM ^o	OK _{≤30}	OK _{>30}	OK _{Auth}	Fails	A	B	C
ASPLOS'12	36	4	2	7	23	2	0	6	14	1	4	3	1	0	17.4	30.4	34.8
CCS'12	75	5	14	19	37	4	4	15	12	2	16	5	2	0	43.2	56.8	62.2
OOPSLA'12	73	0	12	5	56	29	3	10	13	1	21	17	2	2	37.5	67.9	71.4
OSDI'12	24	0	0	7	17	4	4	2	7	0	7	3	0	0	41.2	58.8	58.8
PLDI'12	48	0	1	7	40	9	6	10	14	1	9	13	3	0	22.5	55.0	62.5
SIGMOD'12	46	1	0	19	26	3	6	8	8	1	11	3	3	0	42.3	53.8	65.4
SOSP'11	27	0	1	6	20	3	4	3	9	1	3	3	2	2	15.0	30.0	40.0
TACO'12	60	18	3	2	37	7	2	6	17	5	8	2	2	3	21.6	27.0	32.4
TISSEC'12/13	13	1	6	0	6	3	0	1	2	0	2	0	2	0	33.3	33.3	66.7
TOCS'12	13	1	0	0	12	3	2	0	5	2	2	3	0	0	16.7	41.7	41.7
TODS'12	29	0	12	2	15	1	3	3	3	5	6	1	0	0	40.0	46.7	46.7
TOPLAS'12	16	0	5	2	9	7	1	1	0	0	4	4	0	1	44.4	88.9	88.9
VLDB'12	141	0	7	30	104	10	19	22	42	11	37	7	6	1	35.6	42.3	48.1
Total	601	30	63	106	402	85	54	87	146	30	130	64	23	9	32.3	48.3	54.0
NSF	252	11	15	57	169	36	25	43	54	11	55	31	14	4	32.5	50.9	59.2
No NSF	349	19	48	49	233	49	29	44	92	19	75	33	9	5	32.2	46.4	50.2
Academic	409	20	47	64	278	66	43	69	81	19	102	51	20	5	36.7	55.0	62.2
Joint	148	8	8	36	96	13	11	16	48	8	24	11	1	4	25.0	36.5	37.5
Industrial	44	2	8	6	28	6	0	2	17	3	4	2	2	0	14.3	21.4	28.6
Conferences	470	10	37	100	323	64	46	76	119	18	108	54	19	5	33.4	50.2	56.0
Journals	131	20	26	6	79	21	8	11	27	12	22	10	4	4	27.8	40.5	45.6

on top of obsolete systems, with one respondent saying, “Currently, we have no plans to make the scheduler’s source code publicly available. This is mainly because [ancient OS] as such does not exist anymore ... few people would manage to get it to work on new hardware.”

Some authors were worried about how their code might be used, with one respondent saying, “We would like to be notified in case the provided implementation will be utilized to perform (and possibly publish) comparisons with other developed techniques ... based on earlier (bad) experience, we would like to make sure that our implementation is not used in situations that it was not meant for.”

Producing artifacts solid enough to be shared is clearly labor intensive, with one researcher explaining how he had to make a draconian choice, saying, “[Our system] continues to become more complex as more Ph.D. students add more pieces to it ... In the past when we attempted to share it, we found ourselves spending more time getting outsiders up to speed than on our own research. So I finally had to establish the policy that we will not provide the source code outside the group.”

Unlike researchers in other fields, computer security researchers must contend with the possible negative consequences of making their code public, with one respondent saying, “... we have an agreement with the [business-entity] company, and we cannot release the code because of the potential privacy risks to the general public.”

Some authors used unusual lan-

guages and tools that make it difficult for others to benefit from their code, with one respondent saying, “The code ... is complete, but hardly usable by anyone other than the authors ... due to our decision to use [obscure language variant] for the input language.”

Recommendations

To improve the state of repeatability in computer science research we could simply require, along with every paper submitted for publication, the authors attach the corresponding code, perhaps in the form of a virtual machine image. Unfortunately, based on our study, it is unrealistic to expect computer science researchers to always make their code available to others. There are several reasons for this: the code may not be clean enough for public distribution; they may be considering commercialization; (part of) the code may have licensing restrictions; they may be too busy to answer questions about their system; or they may worry about not receiving proper attribution for any follow-up work.

We thus make a much more modest proposal that would require only minor changes to how public funding agencies and academic publishers operate:

Fund repeatability engineering. Funding agencies should encourage researchers to request additional funds for “repeatability engineering,” including hiring programming staff to document and maintain code, do release management, and assist other research groups wanting to repeat

published experiments. In the same way funding agencies conduct financial audits to ensure costs claimed by grantees are allowed, they should also conduct random audits to ensure research artifacts are shared in accordance with what was promised in the grant application; and

Require sharing contract. Publishers of conference proceedings and journals should require every article include a sharing contract specifying the level of repeatability to which its authors will commit.

While the first point will have the effect of shifting some funding from pure research to engineering and oversight, both are important because they ensure research results continue to benefit the academic community—and the public funding it—past the project end date. Here, we expand on the second point.

Sharing contracts. The sharing contract should be provided by the authors when a paper is submitted for publication (allowing reviewers to consider the expected level of repeatability of the work), as well as in the published version (allowing readers to locate research artifacts). The contract commits the author to making available certain resources that were used in the research leading up to the paper and committing the reader/reviewer to take these resources into account when evaluating the contributions made by the paper.

Table 3 lists the data that should be part of a sharing contract, including the external resources that back up the results in the paper, the locations where these resources can be found or ways to contact the authors, and the level of technical support the authors will provide.

Resources can include code, data, and media. For each resource, the contract must state if it is accessible and at what cost, a deadline after which it might no longer be available, and whether it is available in source or binary form or accessible as a service. Code accessed as a service could be running as a Web service or be executed by the authors themselves on input data provided by the reader. We include an optional comment field to handle unusual situations.

Sharing is different from licensing. A sharing contract represents a com-

Table 3. Data needed for sharing contracts.	
Location	▶ Email address and/or website
Resource	▶ Types (code, data, media, documentation ...)
	▶ Availability (no access, access, NDA access ...)
	▶ Expense (free, non-free, free for academics ...)
	▶ Distribution form (source, binary, service ...)
	▶ Expiration date
	▶ License
	▶ Comment
Support	▶ Kinds (resolve installation issues, fix bugs, upgrade to new language and operating system versions, port to new environments, improve performance, add features ...)
	▶ Expense (free, non-free, free for academics ...)
	▶ Expiration date

mitment on behalf of the author to make resources available to the wider community for scrutiny. A license, on the other hand, describes the actions allowed on these resources (such as modification, redistribution, and reverse engineering). Since copyright bars reuse without permission of the author(s), both licensing and sharing specifications are necessary; for example, if a license prohibits reverse engineering, the community's ability to verify the actions performed by the software are consistent with what is described in the publication is diminished. Likewise, benefaction is hampered by code that makes use of libraries whose license prohibits redistribution.

The contract must also specify the level of technical support the authors commit to provide, for how long they will provide it, and whether that support is free; Table 3 includes a non-exhaustive list of possible types of support.

In some situations authors will want to make their artifacts available under more than one sharing contract, where each contract is targeted at a different audience (such as academic and commercial).

Example of a sharing contract. Publishers must design a concrete syntax for sharing contracts that handles most common situations, balancing expressiveness and conciseness. For illustrative purposes, here is an example contract for the research we have presented, giving free access to source code and data in perpetuity and rudimentary support for free until at least the end of 2016:

Sharing

<http://repeatability.cs.arizona.edu>;
<mailto:collberg@gmail.com>;

code: access, free, source;

data: access, free, source, “sanitized”;

support: installation, bug fixes, free, 2016-12-31;

In this research, we must sanitize email exchanges before sharing them. We express this in the comment field.

Discussion

While there is certainly much room for novel tools for scientific provenance, licensing frameworks that reassure researchers they will be

properly attributed, and repositories that can store research artifacts in perpetuity, we must acknowledge the root of the scientific-repeatability problem is sociological, not technological; when we do not produce solid prêt-à-partager artifacts or attempt to replicate the work of our peers it is because there is little professional glory to be gained from doing so. Nosek⁸ wrote, “Because of strong incentives for innovation and weak incentives for confirmation, direct replication is rarely practiced or published,” and “Innovative findings produce rewards of publication, employment, and tenure; replicated findings produce a shrug.”

The real solution to the problem of researchers not sharing their research artifacts lies in finding new reward structures that encourage them to produce solid artifacts, share these artifacts, and validate the conclusions drawn from the artifacts published by their peers.³ Unfortunately, in this regard we remain pessimistic, not seeing a near future where such fundamental changes are enacted.

In the near term, we thus propose two easily implementable strategies for improving this state of affairs: a shift in public funding to repeatability engineering and adopting sharing specifications. With required sharing contracts, authors—knowing reviewers are likely to take a dim view of a paper that says, up front, its results are not repeatable—will thus be incentivized to produce solid computational artifacts. Adjusting funding-agency regulations to encourage engineering for repeatability will provide them with the resources to do so.

Acknowledgments

We would like to thank Saumya Debray, Shriram Krishnamurthi, Alex Warren, and the anonymous reviewers for valuable input. 

References

1. Collberg, C., Proebsting, T., and Warren, A.M. *Repeatability and Benefaction in Computer Systems Research: A Study and a Modest Proposal*. Technical Report TR 14-04. Department of Computer Science, University of Arizona, Tucson, AZ, Dec. 2014; <http://repeatability.cs.arizona.edu/v2/RepeatabilityTR.pdf>
2. Feitelson, D.G. From repeatability to reproducibility and corroboration. *SIGOPS Operating Systems Review* 49, 1 (Jan. 2015), 3–11.
3. Friedman, B. and Schneider, F.B. *Incentivizing Quality and Impact: Evaluating Scholarship in Hiring, Tenure, and Promotion*. Computing Research Association Best

- Practices Memo, Feb. 2015; http://archive2.cra.org/uploads/documents/resources/bpmmemos/BP_Memo.pdf
4. Gorp, P.V. and Mazanek, S. Share: A Web portal for creating and sharing executable research papers. *Procedia Computer Science* 4 (2011), 589–597.
5. Kovačević, J. How to encourage and publish reproducible research. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Volume IV* (Honolulu, HI, Apr. 15–20). IEEE Computer Society, 2007, 1273–1276.
6. Li-Thiao-Té, S. Literate program execution for reproducible research and executable papers. *Procedia Computer Science* 9 (2012), 439–448.
7. National Science Foundation. *Grant Policy Manual 05-131*. Arlington, VA, July 2005; http://www.nsf.gov/pubs/manuals/gpm05_131
8. Nosek, B.A. An open, large-scale, collaborative effort to estimate the reproducibility of psychological science. *Perspectives on Psychological Science* 7, 6 (Nov. 2012), 657–660.
9. Perianayagam, S., Andrews, G.R., and Hartman, J.H. Rex: A toolset for reproducing software experiments. In *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine* (Hong Kong, Dec. 18–21). IEEE Computer Society, 2010, 613–617.
10. Rozier, K.Y. and Rozier, E.W.D. Reproducibility, correctness, and buildability: The three principles for ethical public dissemination of computer science and engineering research. In *Proceedings of the IEEE International Symposium on Ethics in Science, Technology, and Engineering* (Chicago, IL, May 23–24). IEEE Computer Society, 2014, 1–13.
11. Sandve, G.K., Nekrutenko, A., Taylor, J., and Hovig, E. Ten simple rules for reproducible computational research. *PLoS Computational Biology* 9, 10 (Oct. 24, 2013).
12. Schwab, M., Karrenbach, N., and Claerbout, J. Making scientific computations reproducible. *Computing in Science Engineering* 2, 6 (Nov. 2000), 61–67.
13. Stodden, V. Enabling reproducible research: Licensing for scientific innovation. *International Journal of Communications Law & Policy* 13 (Winter 2009), 22–46.
14. Stodden, V. The legal framework for reproducible scientific research: Licensing and copyright. *IEEE Computing in Science and Engineering* 11, 1 (Jan.-Feb. 2009), 35–40.
15. Stodden, V. *The Scientific Method in Practice: Reproducibility in the Computational Sciences*. Technical Report Working Paper 4773-10. MIT Sloan School of Management, Cambridge, MA, Feb. 2010; <http://web.stanford.edu/~vcs/papers/SMPRCS2010.pdf>
16. Stodden, V., Borwein, J., and Bailey, D.H. ‘Setting the default to reproducible’ in computational science research. *SIAM News* 46, 5 (June 2013).
17. Stodden, V., Hurlin, C., and Perignon, C. RunMyCode.org: A novel dissemination and collaboration platform for executing published computational results. In *Proceedings of the Eighth IEEE International Conference on E-Science* (Chicago, IL, Sept. 15). IEEE Computer Society, 2012, 1–8.
18. Vandewalle, P., Kovačević, J., and Vetterli, M. Reproducible research in signal processing: What, why, and how. *IEEE Signal Processing Magazine* 26, 3 (May 2009), 37–47.
19. Vitek, J. and Kalibera, T. Repeatability, reproducibility, and rigor in systems research. In *Proceedings of the 11th ACM International Conference on Embedded Software* (Taipei, Taiwan, Oct. 9–14). ACM Press, New York, 2011, 33–38.
20. Yale Law School Roundtable on Data and Code Sharing. Reproducible research: Addressing the need for data and code sharing in computational science. *Computing in Science and Engineering* 12, 5 (Sept./Oct. 2010), 8–13.

Christian Collberg (collberg@gmail.com) is a professor of computer science in the Department of Computer Science at the University of Arizona, Tucson, AZ.

Todd A. Proebsting (proebsting@email.arizona.edu) is a professor of computer science in the Department of Computer Science at the University of Arizona, Tucson, AZ.

DOI:10.1145/2795228

MINIX shows even an operating system can be made to be self-healing.

BY ANDREW S. TANENBAUM

Lessons Learned from 30 Years of MINIX

WHILE LINUX IS well known, its direct ancestor, MINIX, is now 30 and still quite spry for such aged software. Its story and how it and Linux got started is not well known, and there are perhaps some lessons to be learned from MINIX's development. Some of these lessons are specific to operating systems, some to software engineering, and some to other areas (such as project management). Neither MINIX nor Linux was developed in a vacuum. There was quite a bit of relevant history before either got started, so a brief introduction may put this material in perspective.

In 1960, the Massachusetts Institute of Technology, where I later studied, had a room-size vacuum-tube-based scientific computer called the IBM 709. Although a modern Apple iPad is 70,000x faster and has 7,300x more RAM, the IBM 709 was the most powerful computer in the world when introduced. Users wrote programs, generally in FORTRAN, on

80-column punched cards and brought them to the human operator, who read them in. Several hours later the results appeared, printed on 132-column fan-fold paper. A single misplaced comma in a FORTRAN statement could cause a compilation failure, resulting in the programmer wasting hours of time.

To give users better service, MIT developed the Compatible Time-Sharing System (CTSS), which allowed users to work at interactive terminals and reduce the turnaround time from hours to seconds while at the same time using spare cycles to run old-style batch jobs in the background. In 1964, MIT, Bell Labs, and GE (then a computer vendor) partnered to build a successor that could handle hundreds of users all over the Boston area. Think of it as cloud computing V.0.0. It was called MULTiplexed Information and Computing Service, or MULTICS. To make a long and complicated story very short, MULTICS had a troubled youth; the first version required more RAM than the GE 645's entire 288kB memory. Eventually, its PL/1 compiler was improved, and MULTICS booted and ran. Nevertheless, Bell Labs soon tired of the project and pulled out, leaving one of its programmers on the project, Ken Thompson, with a burning desire to reproduce a scaled-down MULTICS on cheap hardware. MULTICS itself was released commercially in 1973 and ran at a number of installations worldwide until the last one was shut down on Oct. 30, 2000, a run of 27 years.

Back at Bell Labs, Thompson found a discarded Digital Equipment Corp. PDP-7 minicomputer and wrote a stripped down version of MULTICS in PDP-7 assembly code. Since it could handle only one user at a time, Thompson's col-

» key insights

- Each device driver should run as an independent, user-mode process.
- Software can last a long time and should be designed accordingly.
- It is very difficult to get people to accept new and disruptive ideas.



MINIX's longtime mascot is a raccoon, chosen because it is agile, smart, usually friendly, and eats bugs.

league Brian Kernighan dubbed it the UNIplexed Information and Computing Service, or UNICS. Despite puns about EUNUCHS being a castrated MULTICS, the name UNICS stuck, but the spelling was later changed to UNIX. It is sometimes now written as Unix since it is not really an acronym anymore.

In 1972, Thompson teamed up with his Bell Labs colleague Dennis Ritchie, who designed the C language and wrote a compiler for it. Together they reimplemented UNIX in C on the PDP-11 minicomputer. UNIX went through several internal versions until Bell Labs decided to license UNIX V6 to universities in 1975 for a \$300 fee. Since the PDP-11 was enormously popular, UNIX spread fast worldwide.

In 1977, John Lions of the University of New South Wales in Sydney,

Australia, wrote a commentary on the V6 source code, explaining line by line what it meant, a technological version of a line-by-line commentary on the Bible. Hundreds of universities worldwide began teaching UNIX V6 courses using Lions's book as the text.

The lawyers at AT&T, which owned Bell Labs, were aghast that thousands of students were learning all about their product. This had to stop. So the next release, V7 (1979), came equipped with a license that explicitly forbade anyone from writing a book about it or teaching it to students. Operating systems courses went back to theory-only mode or had to use toy simulators, much to the dismay of professors worldwide. The early history of UNIX has been documented in Peter Salus's 1994 book.¹⁴

MINIX Is Created

There matters rested until 1984, when I decided to rewrite V7 in my spare time while teaching at the Vrije Universiteit (VU) in Amsterdam in order to provide a UNIX-compatible operating system my students could study in a course or on their own. My idea was to write the system, called MINI-uNIX, or MINIX, for the new IBM PC, which was cheap enough (starting at \$1,565) a student could own one. Because early PCs did not have a hard disk, I designed MINIX to be V7 compatible yet run on an IBM PC with 256kB RAM and a single 360kB 5¼-inch floppy disk—a far smaller configuration than the PDP-11 V7 ran on. Although the system was supposed to run on this configuration (and did), I realized from the start that to actually compile and build the whole system

on a PC, I would need a larger system, namely one with the maximum possible RAM (640kB) and two 360kB 5¼-inch floppy disks.

My design goals for MINIX were as follows:

- ▶ Build a V7 clone that ran on an IBM PC with only a single 360kB floppy disk;
- ▶ Build and maintain the system using itself, or “self-hosting”;
- ▶ Make the full source code available to everyone;
- ▶ Have a clean design students could easily understand;
- ▶ Make the (micro) kernel as small as possible, since kernel failures are fatal;
- ▶ Break the rest of the operating system into independent user-mode processes;
- ▶ Hide interrupts at a very low level;
- ▶ Communicate only by synchronous message passing with clear protocols; and
- ▶ Try to make the system port easily to future hardware.

Initially, I did software development on my home IBM PC running Mark Williams Coherent, a V7 clone written by alumni of the University of Waterloo. Its source code was not publicly available. Using Coherent was initially necessary because at first I did not have a C compiler. When my programmer, Cerial Jacobs, was able to port a C compiler based on the Amsterdam Compiler Kit,¹⁸ written at the VU as part of my research, the system became self-hosting. Because I was now using MINIX to compile and build MINIX, I was extremely sensitive to any bugs or flaws that turned up. All developers should try to use their own systems as early as feasible so they can see what users will experience.

Lesson. Eat your own dog food.

The microkernel was indeed small. Only the scheduler, low-level process management, interprocess communication, and the device drivers were in it. Although the device drivers were compiled into the microkernel’s executable program, they were actually scheduled independently as normal processes. This was a compromise because I felt having to do a full address space switch to run a device driver would be too painful on a 4.77MHz 8088, the CPU in the IBM PC. The microkernel was compiled as a standalone executable program. Each of the other operating



**Be careful what
you put out on the
Internet; it might
come back to haunt
you decades later.**



system components, including the file system and memory manager, was compiled as a separate program and run as a separate process. Because the 8088 did not have a memory management unit (MMU), I could have taken shortcuts and put everything into one executable but decided against it because I wanted the design to work on future CPUs with an MMU.

It took me approximately two years to get it running, working on it only evenings and weekends. After the system was basically working, it tended to crash after an hour of operation for no reason at all and in no discernible pattern. Debugging the operating system on the bare metal was well nigh impossible and I came within a hair of abandoning the project.

I then made one final effort. I wrote an 8088 simulator on which to run MINIX, so when it crashed I could get a proper dump and stack trace. To my horror, MINIX would run flawlessly for days, even weeks, at a time on the simulator. It never once crashed. I was totally flummoxed. I mentioned this peculiar situation of MINIX running on the simulator but not on the hardware to my student, Robbert van Renesse, who said he heard somewhere that the 8088 generated interrupt 15 when it got hot. I told him there was nothing in the 8088 documentation about that, but he insisted he heard it somewhere. So I inserted code to catch interrupt 15. Within an hour I saw this message on the screen: “Hi. I am interrupt 15. You will never see this message.” I immediately made the required patch to catch interrupt 15. After that MINIX worked flawlessly and was ready for release.

Lesson. Do not trust documentation blindly; it could be wrong.

Thirty years later the consequences of Van Renesse’s offhand remark are enormous. If he had not mentioned interrupt 15, I would probably have eventually given up in despair. Without MINIX, it is inconceivable there would have been a Linux since Linus Torvalds learned about operating systems by studying the MINIX source code in minute detail and using it as a base to write Linux. Without Linux, there would not have been an Android since it is built on top of Linux. Without Android, the relative stock prices of Apple and Samsung might be quite

different today.

Lesson. Listen to your students; they may know more than you.

I wrote most of the basic utilities myself. MINIX 1.1 included 60 of them, from `ar` to `wc`. A typical one was approximately 4kB. A boot loader today can be 100x bigger. All of MINIX, including the binaries and sources, fit nicely on eight 360kB floppy disks. Four of them were the boot disk, the root file system, `/usr`, and `/user` (see Figure 1). The other four contained the full operating system sources and the sources to the 60 utilities. Only the compiler source was left out, as it was quite large.

Lesson. Nathan Myhrvold's Law is true: Software is a gas. It expands to fill its container.

With some discipline, developers can try to break this “law” but have to try really hard. The default is “more bloat.”

Figuring out how to distribute the code was a big problem. In those days (1987) almost nobody had a proper Internet connection (though newsgroups on USENET via the UUCP program and email existed at some universities). I decided to write a book¹⁵ describing the code, like Lions did before me, and have my publisher, Prentice Hall, distribute the system, including all source code, as an adjunct to the book. After some negotiation, Prentice Hall agreed to sell a nicely packaged box containing eight 5¼-inch floppy disks and a 500-page manual for \$69. This was essentially the manufacturing cost. Prentice Hall had no understanding of what software was but saw selling the software at cost as a way to sell more books. When high-capacity 1.44MB 3½-inch floppies became available later, I also made a version using them.

Lesson. No matter how desirable your product is, you need a way to market or distribute it.

Within a few days of its release, a USENET newsgroup, `comp.os.minix`, was started. Before a month had gone by, it had 40,000 readers, a huge number considering how few people even had access to USENET. MINIX became an instant cult item.

I soon received an email message from Dan Doernberg, co-founder of the now-defunct Computer Literacy bookstore in Silicon Valley inviting me to speak about MINIX if I was ever there. As it turned out, I was going to the Bay

Area in a few weeks to attend a conference, so I accepted. I was expecting him to set up a table and chair in his store for me to sign books. Little did I know he would rent the main auditorium at the Santa Clara Convention Center and do enough publicity to nearly fill it. After my talk, the questions went on until close to midnight.

I began getting hundreds of email messages asking for (no, demanding) this feature or that feature. I resisted some (but not all) demands because I was concerned about the possibility the system would become so big it would require expensive hardware students could not afford, and many people, including me, expected either GNU/Hurd or Berkeley Software Distribution (BSD) to take over the niche of full-blown open-source production system, so I kept my focus on education.

People also began contributing software, some very useful. One of the many contributors was Jan-Mark Wams, who wrote a hugely useful test suite that helped debug the system. He also wrote a new compression program that was better than all the exist-

ing ones at the time. This reduced the number of floppy disks in the distribution by two disks. Even when the distribution later went online this was important because not many people had a super-speed 56kbps modem.

Lesson. Size matters.

In 1985, Intel released its 386 processor with a full protected-mode 32-bit architecture. With the help of many users, notably Bruce Evans of Australia, I was able to release a 32-bit protected mode version of MINIX. Since I was always thinking about future hardware, from day 1, the code clearly distinguished what code ran in “kernel mode” and what code ran as separate processes in “user mode,” even though the 8088 had only one mode. This helped a lot when these modes finally appeared in the 386. Also, the original code clearly distinguished virtual addresses from physical addresses, which did not matter on the 8088 but did matter (a lot) on the 386, making porting to it much easier. Also around this time two people at the VU, Kees Bot and Philip Homburg, produced an excellent 32-bit version with virtual memory, but I

Figure 1. Four of the original 5¼-inch MINIX 1 floppy disks.



decided to stick with Evans's work since it was closer to the original design.

Lesson. Try to make your design be appropriate for hardware likely to appear in the future.

By 1991, MINIX 1.5, had been ported to the Apple Macintosh, Amiga, Atari, and Sun SPARCstation, among other platforms (see Figure 2).

Lesson. By not relying on idiosyncratic features of the hardware, one makes porting to new platforms much easier.

As the system developed, problems cropped up in unexpected places. A particularly annoying one involved a network card driver that could not be debugged. Someone eventually discovered the card did not honor its own specifications.

Lesson. As with software, hardware can contain bugs.

A hardware "feature" can sometimes be viewed as a hardware bug. The port of MINIX to a PC clone made by Olivetti, a major Italian computer manufacturer at the time, was causing problems until I realized, for inex-

plicable reasons, a handful of keys on the Olivetti keyboard returned different scan codes from those returned by genuine IBM keyboards. This led me to realize that many countries have their own standardized keyboards, so I changed MINIX to support multiple keyboards, selectable when the system is installed. This is useful for people with Italian, French, German, and other national keyboards. So, my initial annoyance at Olivetti was tempered when I saw a way to make MINIX better for people in countries other than the U.S. Likewise, in several future cases, what were initially seen as bugs motivated me to generalize the system to improve it.

Lesson. When someone hands you a lemon, make lemonade.

Linus Torvalds Buys a PC

On January 5, 1991, Linus Torvalds, a hitherto-unknown Finnish student at the University of Helsinki, made a critical decision. He bought a fast (33MHz) large (4MB RAM, 40MB hard

disk) PC largely for the purpose of running MINIX and studying it. On March 29, 1991, Torvalds posted his first message to the USENET newsgroup, comp.os.minix:

"Hello everybody, I've had minix for a week now, and have upgraded to 386-minix (nice), and duly downloaded gcc for minix ..."

His second posting to comp.os.minix was on April 1, 1991, in response to a simple question from someone else:

"RTFSC (Read the F***ing Source Code :-)—It is heavily commented and the solution should be obvious ..."

This posting shows that in 10 days, Torvalds had studied the MINIX source code well enough to be somewhat disdainful of people who had not studied it as well as he had. The goal of MINIX at the time was, of course, to be easy for students to learn; in Torvalds' case, it was wildly successful.

Then on August 25, 1991, Torvalds made another post to comp.os.minix:

"Hello everybody out there using minix—I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since April, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things)."

During the next year, Torvalds continued studying MINIX and using it to develop his new system. This became the first version of the Linux kernel. Fossilized remains of its connection to MINIX were later visible to software archaeologists in things like the Linux kernel using the MINIX file system and source-tree layout.

On January 29, 1992, I posted a message to comp.os.minix saying micro-kernels were better than monolithic designs, except for performance. This posting unleashed a flamewar that still, even today, 24 years later, inspires many students worldwide to write and tell me their position on this "debate."

Lesson. The Internet is like an elephant; it never forgets.

That is, be careful what you put out on the Internet; it might come back to haunt you decades later.

Figure 2. MINIX 1.5 for four different platforms.



It turns out performance is more important to some people than I had expected. Windows NT was designed as a microkernel, but Microsoft later switched to a hybrid design when the performance was not good enough. In NT, as well as in Windows 2000, XP, 7, 8, and 10, there is a hardware abstraction layer at the very bottom (to hide differences between motherboards). Above it is a microkernel for handling interrupts, thread scheduling, low-level interprocess communication, and thread synchronization. Above the microkernel is the Windows Executive, a group of separate components for process management, memory management, I/O management, security, and more that together comprise the core of the operating system. They communicate through well-defined protocols, just like on MINIX, except on MINIX they are user processes. NT (and its successors) were something of a hybrid because all these parts ran in kernel mode for performance reasons, meaning fewer context switches. So, from a software engineering standpoint, it was a microkernel design, but from a reliability standpoint, it was monolithic, because a single bug in any component could crash the whole system. Apple's OS X has a similar hybrid design, with the bottom layer being the Mach 3.0 microkernel and the upper layer (Darwin) derived from FreeBSD, a descendant of the BSD system developed at the University of California at Berkeley.

Also worth noting is in the world of embedded computing, where reliability often trumps performance, microkernels dominate. QNX, a commercial UNIX-like real-time operating system, is widely used in automobiles, factory automation, power plants, and medical equipment. The L4 microkernel¹¹ runs on the radio chip inside more than one billion cellphones worldwide and also on the security processor inside recent iOS devices like the iPhone 6. L4 is so small, a version of it consisting of approximately 9,000 lines of C was formally proven correct against its specification,⁹ something unthinkable for multimillion-line monolithic systems. Nevertheless, microkernels remain controversial for historical reasons and to some extent due to somewhat lower performance.¹⁶



What was new about MINIX research was the attempt to build a fault-tolerant multi-server POSIX-compliant operating system on top of the microkernel.



On the newsgroup comp.os.minix in 1992 I also made the point that tying Linux tightly to the 386 architecture was not a good idea because RISC machines would eventually dominate the market. To a considerable extent this is happening, with more than 50 billion (RISC) ARM chips shipped. Most smartphones and tablets use an ARM CPU, including variants like Qualcomm's Snapdragon, Apple's A8, and Samsung's Exynos. Furthermore, 64-bit ARM servers and notebooks are beginning to appear. Linux was eventually ported to the ARM, but it would have been much easier had it not been tied so closely to the x86 architecture from the start.

Lesson. Do not assume today's hardware will be dominant forever.

Also in this vein, Linux is so tightly tied to the gcc compiler that compiling it with newer (arguably, better) compilers like clang/LLVM requires major patches to the Linux code.

Lesson. When standards exist (such as ANSI Standard C) stick to them.

In addition to the real start of Linux, another major development occurred in 1992. AT&T sued BSDI (a company created by the developers of Berkeley UNIX to sell and support the BSD software) and the University of California. AT&T claimed BSD contained pieces of AT&T code and also BSDI's telephone number, 1-800-ITS-UNIX, violated AT&T's intellectual property rights. The case was settled out of court in 1994, until which time BSD was handcuffed, giving the new Linux system critical time to develop. If AT&T had been more sensible and just bought BSDI as its marketing arm, Linux might never have caught on against such a mature and stable competitor with a very large installed base.

Lesson. If you are running one of the biggest corporations in the world and a tiny startup appears in an area you care about but know almost nothing about, ask the owners how much they want for the company and write them a check.

In 1997, MINIX 2, now changed to be POSIX-compatible rather than UNIX V7-compatible, was released, along with a second edition of my book *Operating Systems Design and Implementation*, now co-authored with Albert Woodhull, a professor at Hampshire College in Massachusetts.

In 2000, I finally convinced Prentice

Hall to release MINIX 2 under the BSD license and make it (including all source code) freely available on the Internet. I should have tried to do this much earlier, especially since the original license allowed unlimited copying at universities, and it was being sold at essentially the publisher's cost price anyway.

Lesson. Even after you have adopted a strategy, you should nevertheless reexamine it from time to time.

MINIX as Research Project

MINIX 2 continued to develop slowly for a few more years, but the direction changed sharply in 2004 when I received a grant from the Netherlands Organisation for Scientific Research (<http://www.nwo.nl>) to turn what had been an educational hobby into a serious, funded research project on building a highly reliable system; until 2004, there was no external funding. Shortly thereafter, I received an Academy Professorship from the Royal Netherlands Academy of Arts and Sciences in Amsterdam. Together, these grants provided almost \$3 million for research into reliable operating systems based on MINIX.

Lesson. Working on something important can get you research funding, even if it is outside the mainstream.

MINIX was not, of course, the only research project looking at microkernels. Early systems from as far back as 1970 included Amoeba,¹⁷ Chorus,¹² L3,¹⁰ L4,¹¹ Mach,¹ RC 4000 Nucleus,³ and V.⁴ What was new about MINIX research was the attempt to build a fault-tolerant multi-server POSIX-compliant operating system on top of the microkernel.

Together with my students and programmers in 2004, I began to develop MINIX 3. Our first step was to move the device drivers entirely out of the microkernel. In the MINIX 1 and MINIX 2 designs, device drivers were treated and scheduled as independent processes but lived in the microkernel's (virtual) address space. My student Jorrit Herder's master's thesis consisted of making each driver a full-blown user-mode process. This change made MINIX far more reliable and robust. During his subsequent Ph.D. research at the VU under my supervision, Herder showed failed drivers could be replaced on the fly, while the system was running, with no adverse effects at all.⁷ Even a failed disk driver could be replaced on the fly, since a copy

was always kept in RAM; the other drivers could always be fetched from disk. This was a first step toward a self-healing system. The fact that MINIX could now do something—replace (some) key operating system components that had crashed without rebooting and without running application processes even noticing it—no other system could do this, which gave my group confidence we were really onto something.

Lesson. Try for an early success of some kind; it builds up everyone's morale.

This change made it possible to implement the Principle of Least Authority, also called Principle of Least Privilege,¹³ much better. To touch device registers, even for its own device, a driver now had to make a call to the microkernel, which could check if that driver had permission to access the device, greatly improving robustness. In a monolithic system like Windows or Linux, a rogue or malfunctioning audio driver has the power to erase the disk; in MINIX, the microkernel will not let it. If an I/O memory-management unit is present, mediation by the microkernel is not needed to achieve the same effect.

In addition, components could communicate with other components only if the microkernel approved, and components could make only approved microkernel calls, all of this controlled by tables and bitmaps within the microkernel. This new design with tighter restrictions on the operating system components (and other improvements) was called MINIX 3 and coincided with the third edition of my and Woodhull's book *Operating Systems Design and Implementation, Third Edition*.

Lesson. Each device driver should run as an unprivileged, independent user-mode process.

Microsoft clearly understood and still understands this and introduced the User-Mode Driver Framework for Windows XP and later systems, intending to encourage device-driver writers to make their drivers run as user-mode processes, just as in MINIX.

In 2005, I was invited to be the keynote speaker at ACM's Symposium on Operating System Principles (<http://www.sosp.org>), the top venue for operating systems research. It was held in October at the Grand Hotel in Brighton, U.K., that year. I decided in my talk I would formally announce MINIX

3 to the assembled operating system experts. Partway through my talk I removed my dress shirt on stage to reveal a MINIX 3 T-shirt. The MINIX website was set up to allow downloading starting that day. Needless to say, I wanted to be online during the conference to see if the server could handle the load. Since I was the honored guest of the conference, I was put in the Royal Suite, where the Queen of England would stay should she choose to visit Brighton. It is a massive room, with a magnificent view of the sea. Unfortunately, it was the only room in the hotel lacking an Internet connection, since apparently the Queen is not a big Internet user. To make it worse, the hotel did not have Wi-Fi. Fortunately, one of the conference organizers took pity on me and was willing to swap rooms so I could have a standard room but with that oh-so-important Ethernet port.

Lesson. Keep focused on your real goal.

That is, do not be distracted when something seemingly nice (like a beautiful hotel room) pops up but is actually a hindrance.

By 2005, MINIX 3 was a much more serious system, but so many people had read the *Operating Systems Design and Implementation* book and studied MINIX in college it was very difficult to convince anyone it was not a toy system anymore. So I had the irony of a very well-known system but had to struggle to get people to take it seriously due to its history. Microsoft was smarter; early versions of Windows, including Windows 95 and Windows 98, were just MS-DOS with a graphical shell. But if they had been marketed as "Graphical MS-DOS" Microsoft might not have done as well as renaming them "Windows," which Microsoft indeed did.

Lesson. If V3 of your product differs from V2 in a really major way, give it a totally new name.

In 2008, the MINIX project received another piece of good luck. For some years, the European Union had been toying with the idea of revising product liability laws to apply to software. If one in 10 million tires explode, killing people, the manufacturer cannot get off the hook by saying, "Tire explosions happen." With software, that argument works. Since a country or other jurisdiction cannot legislate something that is technically impossible, the European

Research Council, which is funded by the E.U., decided to give me a European Research Council Advanced Grant of roughly \$3.5 million to see if I could make a highly reliable, self-healing operating system based on MINIX.

While I was enormously grateful for the opportunity, this immense good fortune also created a major problem. I was able to hire four expert professional programmers to develop “MINIX 3, the product” while also funding six Ph.D. students and several postdocs to push the envelope on research. Before long, each Ph.D. student had copied the MINIX 3 source tree and began modifying it in major ways to use in his research. Meanwhile, the programmers were busy improving and “productizing” the code. After two or three years, we were unable to put Humpty Dumpty back together again. The carefully developed prototype and the students’ versions had diverged so much we could not put their changes back in, despite our using *git* and other state-of-the-art tools. The versions were simply too incompatible. For example, if two people completely rewrite the scheduler using totally different algorithms, they cannot be automatically merged later.

Also, despite my stated desire to put the results of the research into the product, the programmers strongly resisted, since they had been extremely meticulous about their code and were not enthusiastic (to put it mildly) about injecting a lot of barely tested student-quality code into what had become a well-tested production system. Only with a lot of effort would my group possibly succeed with getting one of the research results into the product. But we did publish a lot of papers; see, for example Appuswamy et al.,² Giuffrida et al.,⁵ Giuffrida et al.,⁶ and Hruby et al.⁸

Lesson. Doing Ph.D. research and developing a software product at the same time are very difficult to combine.

Sometimes both researchers and programmers would run into the same problem. One such problem involved the use of synchronous communication. Synchronous communication was there from the start and is very simple. It also conflicts with the goal of reliability. If a client process, C, sends a message to a server process, S, and C crashes or gets stuck in an infinite loop without listening for the response, the

server hangs because it is unable to send its reply. This problem is inherent in synchronous communication. To avoid it, we were forced to introduce virtual endpoints, asynchronous communication, and other things far less elegant than the original design.

Lesson. Einstein was right: Things should be as simple as possible but not simpler.

What Einstein meant is everyone should strive for simplicity and make sure their solution is comprehensive enough to do the job but no more. This has been a guiding principle for MINIX from the start. It is unfortunately absent in far too much modern bloated software.

Around 2011, the direction we were going to take with the product began to come into better focus, and we made two important decisions. First, we came to realize that to get anyone to use the system it had to have applications, so we adopted the headers, libraries, package manager, and a lot more from BSD (specifically, NetBSD). In effect, we had reimplemented the NetBSD user environment on a much more fault-tolerant substructure. The big gain here was 6,000 NetBSD packages were suddenly available.

Lesson. If you want people to use your product, it has to do something useful.

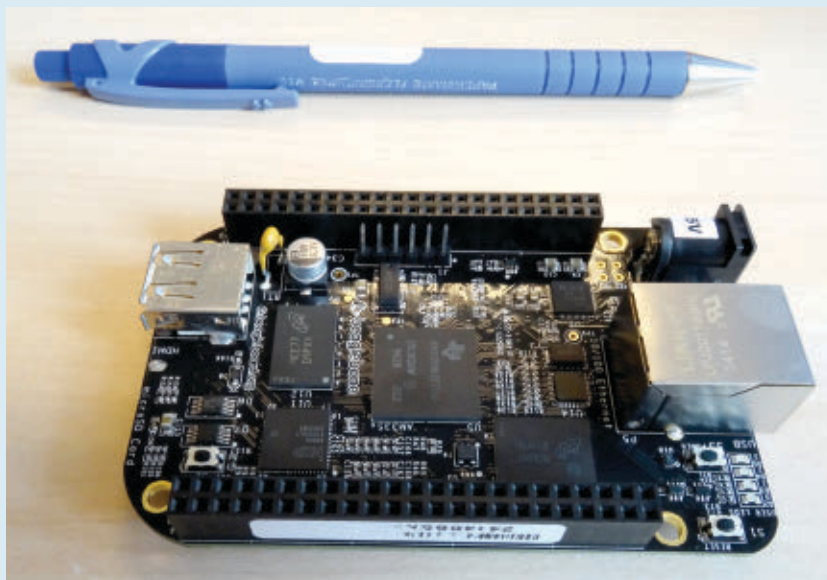
Second, we realized winning the desktop war against Windows, Linux,

OS X, and half a dozen BSDs was a tall order, although MINIX 3 could well be used in universities as a nice base for research on fault-tolerant computing. So we ported MINIX 3 to the ARM processor and began to focus on embedded systems, where high reliability is often crucial. Also, when engineers are looking for an operating system to embed in a new camera, television set, digital video recorder, router, or other product, they do not have to contend with millions of screaming users who demand the product be backward compatible to 1981 and run all their MS-DOS games as fast as their previous product did. All the users see is the outside, not the inside. In particular, we got MINIX 3 running on the BeagleBone series of single-board computers that use the ARM Cortex-A8 processor (see Figure 3). These boards are essentially complete PCs and retail for about \$50. They are often used to prototype embedded systems. All of them are open source hardware, which made figuring out how they work easy.

Lesson. If marketing the product according to plan A does not work, invent plan B.

Retrospective. With 20-20 hindsight, some things stand out now. First, the idea of a small microkernel with user-level system components protected from each other by the hardware MMU is probably still the best way to aim for

Figure 3. A BeagleBone Black Board.



highly reliable, self-healing systems because this design keeps problems in one component from spreading to others. It is perhaps surprising that in 30 years, almost no code was moved into the MINIX microkernel. In fact, some major software components, including all the drivers and much of the scheduler, were moved out of it. The world is also moving (slowly) in this direction (such as Windows User-mode drivers and embedded systems). Nevertheless, having most of the operating system run as user-mode processes is disruptive, and it takes time for disruptive ideas to take hold; for example, FORTRAN, Windows XP, mainframes, QWERTY keyboards, the x86 architecture, fax machines, magnetic-stripe credit cards, and the interlaced NTSC color television standard made sense when they were invented but not so much anymore. However, they are not about to exit gracefully. For example, according to Microsoft, as of March 2016, the obsolete Windows XP still runs on 250 million computers.

Lesson. It is very difficult to change entrenched ways of doing things.

Furthermore, in due course, computers will have so much computing power, efficiency will not matter so much. For example, Android is written in Java, which is far slower than C, but nobody seems to care.

My initial decision back in 1984 to have fixed-size messages throughout the system and avoid dynamic memory allocation (such as `malloc`) and a heap in the kernel has not been a problem and avoids problems that occur with dynamic storage management (such as memory leaks and buffer overruns).

Another thing that worked well in MINIX is the event-driven model. Each driver and server has a loop consisting of

```
{ get_request();
  process_request();
  send_reply();
}
```

This design makes them easy to test and debug in isolation.

On the other hand, the simplicity of MINIX 1 limited its usability. Lack of features like kernel multithreading and full-demand paging were not a realistic option on a 256kB IBM PC with one floppy disk. We could have added them

(and all their complexity) at some point, but we did not (although we have some workarounds) and are paying a price today, as porting some software is more difficult than it would otherwise be.

Although funding has now ended, the MINIX project is not ending. It is instead transitioning to an open source project, like so many others. Various improvements are in progress now, including some very interesting ones (such as being able to update nearly all of the operating system drivers, file system, memory manager, and process manager) on the fly to major new versions (potentially with different data structures) while the system is running.^{5,6} These updates require no down time and have no effect on running processes, except for the system freezing very briefly before continuing. The structure of the system as a collection of servers makes live update much simpler than in traditional designs, since it is possible to do a live update on, say, the memory manager, without affecting the other (isolated) components because they are in different address spaces. In systems that pass pointers between subsystems within the kernel, live updating one piece without updating all of them is very difficult. This area is one of the few where the research may make it into the product, but it is an important one that few, if any, other systems have.

MINIX 3 can be downloaded for free at <http://www.minix3.org>.

Acknowledgments

I would like to thank the hundreds of people who have contributed to the MINIX project over the course of 30 years. Unfortunately there are far too many to name here. Nevertheless, some stand out for a special mention: Kees Bot, Ben Gras, Philip Homburg, Kees Jongenburger, Lionel Sambuc, Arun Thomas, Thomas Veerman, and Jan-Mark Wams. This work was supported in part by the Netherlands Organisation for Scientific Research, as well as by European Research Council Advanced Grant 227874 and ERC Proof of Concept Grant 297420. 

References

1. Accetta, M., Baron, R., Golub, D., Rashid, R., Tevian, A., and Young, M. Mach 1986: A new kernel foundation for Unix development. In *Proceedings of the USENIX Summer Conference* (Atlanta, GA, June 9–13).

2. Appuswamy, R., van Moolenbroek, D.C., and Tanenbaum, A.S. Loris: A dependable, modular file-based storage stack. In *Proceedings of the 16th Pacific Rim International Symposium of Dependable Computing* (Tokyo, Dec. 13–15). IEEE Computer Society, Washington, D.C., 2010, 165–174.
3. Brinch Hansen, P. The nucleus of a multiprogramming system. *Commun. ACM* 13, 4 (Apr. 1970), 238–241.
4. Chertoff, D.R. The V kernel, a software base for distributed systems. *IEEE Software* 1, 4 (Apr. 1984), 19–42.
5. Giuffrida, C., Iorgulescu, C., Kuijsten, A., and Tanenbaum, A.S. Back to the future: Fault-tolerant live update with time-traveling state transfer. In *Proceedings of the 27th Large Installation System Administration Conference* (Washington D.C., Nov. 3–8). USENIX Association, Berkeley, CA, 2013, 89–104.
6. Giuffrida, C., Kuijsten, A., and Tanenbaum, A.S. Safe and automatic live update for operating systems. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, TX, Mar. 16–20). ACM Press, New York, 2013, 279–292.
7. Herder, J. *Building a Dependable Operating System, Fault Tolerance in MINIX 3*. Ph.D. Thesis, Vrije Universiteit, Amsterdam, the Netherlands, 2010; <http://www.cs.vu.nl/~ast/Theses/herder-thesis.pdf>
8. Hruby, T., Bos, H., and Tanenbaum, A.S. When slower is faster: On heterogeneous multicores for reliable systems. In *Proceedings of the Annual Technical Conference* (San Jose, CA, June 26–28). USENIX Association, Berkeley, CA, 2013, 255–266.
9. Klein G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Swell, T., Tuch, H., and Winwood, S. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd Symposium on Operating Systems Principles* (Big Sky, MT, Oct. 11–14). ACM Press, New York, 2009, 207–220.
10. Liedtke, J. Improving IPC by kernel design. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles* (Asheville, NC, Dec. 5–8). ACM Press, New York, 1993, 174–188.
11. Liedtke, J. On microkernel construction. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Copper Mountain Resort, CO, Dec. 3–6). ACM Press, New York, 1995, 237–250.
12. Rozier, M., Abrossimov, V., Armand, F., Boule, I., Gien, M., Guillemont, M., Herrmann, F., Kaiser, C., Langlois, S., Leonard, P., and Neuhauser, W. The CHORUS distributed operating system. *Computing Systems Journal* 1, 4 (Dec. 1988), 305–370.
13. Saltzer, J.H. and Schroeder, M.D. The protection of information in computer systems. *Proceedings of the IEEE* 63, 9 (Sept. 1975), 1278–1308.
14. Salus, P.H. *A Quarter Century of UNIX*. Addison-Wesley, Reading, MA, 1994.
15. Tanenbaum, A.S. *Operating Systems Design and Implementation, First Edition*. Prentice Hall, Upper Saddle River, NJ, 1987.
16. Tanenbaum, A.S., Herder, J., and Bos, H.J. Can we make operating systems reliable and secure? *Computer* 39, 5 (May 2006), 44–51.
17. Tanenbaum, A.S. and Mullender, S.J. A capability-based distributed operating system. In *Proceedings of the Conference on Local Networks & Distributed Office Systems* (London, U.K., May 1981), 363–377.
18. Tanenbaum, A.S., van Staveren, H., Keizer, E.G., and Stevenson, J.W. A practical toolkit for making portable compilers. *Commun. ACM* 26, 9 (Sept. 1983), 654–660.

Andrew S. Tanenbaum (ast@cs.vu.nl) is a professor emeritus of computer science in the Department of Computer Science in the Faculty of Sciences at the Vrije Universiteit, Amsterdam, the Netherlands and an ACM Fellow.

Copyright held by the author.



Watch the author discuss his work in this exclusive *Communications* video. <http://cacm.acm.org/videos/lessons-learned-from-30-years-of-minix>

UPON Lite focuses on users, typically domain experts without ontology expertise, minimizing the role of ontology engineers.

BY ANTONIO DE NICOLA AND MICHELE MISSIKOFF

A Lightweight Methodology for Rapid Ontology Engineering

WE ARE LIVING in a reality that, thanks to economic globalization and the Internet, is increasingly interconnected and complex. There is thus a growing need for semantic technology solutions that can help us better understand it, particularly from a conceptual point of view. Ontologies represent an essential

» key insights

- **Ontology engineering helps domain experts better understand their own business reality by systematically exploring and describing it.**
- **The lightweight UPON Lite methodology for developing enterprise ontologies gives end users and business experts a central role, substantially reducing the need for ontology engineers.**
- **Ontology engineering is difficult for non-ontology specialists like business experts, but UPON Lite helps them develop ontologies without ontology engineers until the final step of formalization.**

component to developing the Web of Data (such as Linked Open Data¹) and Semantic Web applications. An ontology is a conceptual model of (a fragment of) an observed reality; it is, in essence, a repository of interlinked concepts pertaining to a given application domain. Traditionally, construction of an ontology (and its constant evolution, necessary to keep it aligned with reality) is lengthy and costly.

A high-quality ontology requires a rigorous, systematic engineering approach. Existing methodologies are quite complex, conceived primarily for skilled on-

tology engineers trained to develop large, industrial-strength ontologies. However, before embarking on a full-scale ontology project, it is useful to pursue pilot projects with experimental implementations, testing the applicability of semantic technologies in a confined enterprise area. From this perspective, available ontology engineering methodologies are often unsuitable, overly complex, and demanding in terms of time, cost, and skilled human resources.

There is thus a growing need for simpler, easy-to-use methods for ontology building and maintenance, conceived and designed for end users (such as domain experts, stakeholders, and even casual users in the relevant business domain), reducing the role of (and dependence on) ontology engineers. The objective is to shift responsibility for ontology building toward a community of end users through a social, highly participative approach supported by an easy-to-use method and tools.

We propose a simple, agile ontology engineering method intended to place end users at the center of the process. The proposed method, derived from the full-fledged Unified Process for Ontology building (UPON) methodology,⁷ guarantees a rigorous, systematic approach but also reflects an intuitive nature. This method, or UPON Lite (to reflect its origin in its name), is conceived for a wide base of users (typically domain experts) without specific

ontology expertise. UPON Lite is organized as an ordered set of steps, each releasing a self-contained artifact readily available to end users. Moreover, it is progressive and differential, with each new step using the outcome of the preceding step, providing well-defined enrichment to it.

The UPON Lite methodology contributes significantly to the “disintermediation” of ontology engineers. Before UPON Lite, it was necessary to assign the work of developing an ontology to a joint team of ontology engineers and domain experts; the latter bring knowledge of the domain, and the former are in charge of the ontology building process, following a rigorous method and notation. With UPON Lite, an ontology can be constructed largely by domain experts (along with end users) without ontology engineers. Only in the last step—once the domain content is elicited, organized, and validated—do ontology engineers intervene to deliver final ontology formalization before releasing it to users. We are confident there will soon be effective tools supporting domain experts (also in the last step), transforming semi-formal ontological knowledge into a formal ontological encoding.

UPON Lite is based on three main pillars of ontology development: a user-centered approach emphasizing the role of domain experts; a social approach leveraging the collec-

tive intelligence of domain experts, working to progressively achieve the steps in the method; and an ontology-building process articulated in six well-defined steps, each producing readily usable output.

The rest of this article explores the six steps of UPON Lite, the essence of the social approach, and a description of each step, showing the end user's role in progressively enriching the ontology base. It then covers related work and a comparative evaluation of the method. The final section draws a number of conclusions.

Six Steps to Light Ontology Building

UPON Lite is organized as a sequence of steps, where the outcome of each one is enriched and refined in the succeeding step; the steps produce the following outcomes (see also Figure 1):

Step 1. Domain terminology. The domain lexicon listing the domain terms that characterize the observed domain;

Step 2. Domain glossary. The terms of the lexicon associated with a textual description, indicating also possible synonyms;

*Step 3. Taxonomy.*¹¹ Domain terms organized in a generalization/specialization (ISA) hierarchy;

*Step 4. Predication.*¹⁷ Terms representing properties from the glossary identified and connected to the entities they characterize;

*Step 5. Parthood (meronymy).*⁹ Complex entity names connected to their components, with all names needing to be present in the glossary; and

Step 6. Ontology. This last step produces the formally encoded ontology by using, say, the Web Ontology Language, or OWL, containing the conceptual knowledge collected in the five previous steps.

While this step numbering suggests a sort of sequencing, Figure 1 outlines the dependence among the different steps. In particular, there is no inherent dependency among intermediate steps 3, 4, and 5, as they can be performed in parallel. Moreover, the ontology-building process lacks a simple, linear progression, and the n^{th} step also provides feedback to improve the previous steps; to improve legibility, the figure omits backward arrows. Finally, depending on context and business objectives, users can skip one or two

Figure 1. Sequence of steps in UPON Lite.

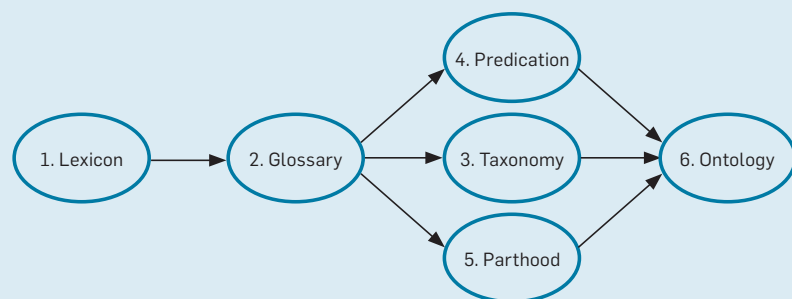


Table 1. Excerpt from a purchasing lexicon.

Address	Postal address	Request for quote
Customer	Price	RFQ
Delivery address	Purchasing conditions	Supplier
Invoice	Purchase order	Unit price
PO	Request for quotation	

intermediate steps. For instance, if interested in relational database design, they can concentrate on step 4, skipping step 3 and step 5, representing the rest of the knowledge as relational attributes; if interested in developing a product lifecycle management solution, they can also focus on step 5.

Before detailing the steps, we first explore the social approach of UPON Lite.

A Social Approach to Rapid Ontology Engineering

The traditional responsibility of an ontology-building project is given to a team of ontology engineers working with domain experts. However, it involves serious limitations as to the diffusion of ontologies and, more generally, semantic technologies, for several reasons. First is the shortage of ontology engineers with specialized technical competencies not generally available in the job market; second, ontology engineers, no matter how experienced, are seldom able to take in all relevant aspects of the application domain and, when an ontology is first released, there is always a need for domain-driven corrections, integrations, and extensions. Related to this need, and as the ontology is a sort of “conceptual image” of reality, even a “perfect” ontology must be maintained over time and, following the direction of domain experts, periodically realigned with the ever-changing world.

The idea of a “closed” team, no matter how articulate and skilled its members, can hardly respond to the indicated needs of the ontology to be developed. Conversely, the extensive involvement of users and stakeholders¹⁵ is indeed the optimal solution. Users thus need to proceed along three lines:

Adopt simple tools. Simple tools for conceptual-modeling activities shield stakeholders, including domain experts and end users, from the intricacy and technical details of semantic technologies;

Open boundaries. The boundaries of an ontology team can be opened by adopting a social, participative approach to the collection, modeling, and validation of domain knowledge; and

Rethink the process. The ontology engineering process must be rethought to simplify the method, making it readily adoptable by non-ontology expert users (such as domain experts) and enforcing the methodological rigor nec-



The objective is to shift responsibility for ontology building toward a community of end users through a social, highly participative approach supported by an easy-to-use method and tools.



essary to guarantee the quality of the resulting ontology.

There is also an organizational issue. Along with an enlarged, fluid organization for the ontology-engineering team comes a pivotal role for an ontology master with the expertise of an ontology engineer and the responsibility of monitoring and coordinating advancement of ontology-engineering activities.²

In UPON Lite, the whole ontology-building-and-management process is carried out through a socially oriented approach (in a transparent and participatory way) on a social-media platform. Here, we propose examples based on the Google Docs suite. In particular we experimented with shared Google Sheets for ontology engineering, plus Google Forms and Google+ for other functions (such as debating and voting on contentious issues).

The steps in UPON Lite are presented here through an example reflecting a (simplified) purchasing process, thus dealing with concepts like request for quotation, purchase order, and invoice. Purchasing is an activity that takes place in all business sectors; to make the example as general as possible, we thus concentrate on the business part, skipping the product-specific issues (such as the domain-specific goods or services to be purchased).

Step 1. Terminological Level

The first step in UPON Lite involves creating a domain-specific terminology, or list of terms characterizing the domain at hand. The outcome of this step is a domain lexicon, or information structure used to answer questions like “What are the characterizing words, nouns, verbs, and adjectives typically used while doing business in this domain? This is a preliminary step to start identifying domain knowledge and drawing the boundaries of the observed domain. The terminology in Table 1 is part of a purchasing lexicon. Note, at this level the terminology is a simple, flat, undifferentiated (with respect to, say, nouns and verbs) list of terms, including synonyms as separate entries. The basic rule for inclusion of terms in the list is the (statistical) evidence that a domain professional in the procurement sector would recognize each term as relevant.

In building a lexicon, users need not start from scratch. The Web makes it

Table 2. Glossary, including synonyms, kinds, and descriptions.

Term	Synonyms	Kind	Description [source]
Delivery address	Shipping address	Complex property	Location to which goods are to be sent. ¹
Invoice	Bill	Object	Itemized list of goods shipped, usually specifying the price and terms of sale. ²
Postal address	Address	Complex property	Information that locates and identifies a specific address, as defined by the postal service. ³
Purchasing conditions	Purchase terms and conditions	Object	Conditions related to the transaction and the trade. ⁴
Purchase order	PO	Object	Commercial document issued by a buyer to a seller, indicating types, quantities, and agreed prices for products or services the seller will provide to the buyer. ⁵
Customer	Client	Actor	One who purchases a commodity or service. ²
Invoicing	Issuing invoice	Process	Making or issuing an invoice for goods or services. ⁶
Purchasing	Buying	Process	Acquisition of something for payment. ⁶

Sources: 1. <http://glosbe.com>; 2. <http://www.merriam-webster.com>; 3. <http://www.ebxml.org>; 4. <http://docs.oasis-open.org/ubl/os-UBL-2.0/UBL-2.0.html>; 5. <http://www.wikipedia.org>; 6. <http://www.thefreedictionary.com>

possible to find knowledge resources (such as textual documents, directories, dictionaries, taxonomies, standards, and ontologies) dealing with the business sector. If standards exist, it is worth taking advantage of them; for instance, business documents involve standards like the Universal Business Language²² International Data Dictionary that may be useful in the ontology to be developed. Another method is extraction of the terminology from reference textual documents (such as manuals, textbooks, and whitepapers). A number of natural language processing tools are available for extracting terminology from text, including AlchemyAPI²³ and Open Calais,²⁴ which are readily available through a Web interface, and Term Extractor,¹⁴ which is able to analyze a domain corpus and identify the terminology.

Challenges. The main challenge is deciding if a term is relevant and thus to identify the boundaries of the target application domain to include in the lexicon only the appropriate terminology. Identifying this boundary is not easy, since in nature there are no domain boundaries, and all is connected. Users must then consider two key dimensions: Along the “horizontal,” what is the scope of the sought ontology?; imagine you are defining an ontology in a medical domain, how much of the

technology domain (such as associated with electro-medical devices) should be included? There are two main strategies to defining the related scope: enriching the domain ontology with “foreign” terms or, if a “foreign” ontology is available, creating appropriate links to it. Along the “vertical” is the problem of the right level of detail, or granularity, that should be considered—not too much to avoid ineffective overloading, not too little to avoid critical omissions.

Another important challenge concerns the initial resources to be used as references. We mentioned there are plenty of terminological resources for all possible application domains. The problem is how users should select the ones they consider most representative. Domain corpora are important, but, despite the growing reliability and effectiveness of term extractors, post-processing of the extracted terms requires intervention of domain experts to produce a first list of relevant terms.

Social validation is another key challenge. While users have no problem publishing a particular lexicon online, the challenge they have concerns the method they choose for open consultation. There are a large number of off-the-shelf solutions for a deliberative online consultation. Here, users can adopt the simple method: “Like/Do Not Like”

voting, since we find social participation decreases when people are asked to vote with more alternatives. Furthermore, our research suggests people, especially with a large lexicon, do not get through the whole list, tending instead to quickly browse the list and stop at terms they find objectionable.²¹ An effective approach considers the listed terms as accepted if not explicitly rejected. Therefore, domain experts voting on relevant terms have only one option: “Do Not Like.” Terms with a high number of rejections (above a given threshold) are removed from the lexicon. A richer method of social participation could include the option of proposing new terms and more sophisticated voting methods; a vast literature is available, starting with Parveen et al.¹²

Step 2. Glossary Level

Having produced a first lexicon, users could, in this step, enrich it by associating a textual description with each entry. This is critical; in fact, there are terms with a well-defined and widely accepted meaning, even defined by regulations and laws (such as those that apply to invoices), but there are also widely used terms that may have a different meaning in different business situations. For instance, what is a “delayed payment”? and How many days must elapse to classify a payment as “delayed”? Furthermore, a good engineering practice is not to “invent” descriptions but import them from authoritative sources.

Besides descriptions, users can start to add extra bits of semantics in this step. To this end, we adopt a method that uses the conceptual categories of the Object, Process, Actor modeling Language, or OPAL,⁵ an ontology-structuring method that groups the concepts into three main categories—object, process, and actor—plus three auxiliary categories—complex, atomic, and reference properties. The actor category gathers active entities of a business domain, able to activate, perform, or monitor a process. The object category gathers passive entities on which a process operates. The process category gathers activities and operations aimed at helping achieve a business goal. We refer to such categories as “kinds” as a first semantic tagging of the terms representing the domain concepts.

Finally, having the description of

terms, it is easy to identify the synonyms. In identifying synonyms it is necessary to pinpoint the “preferred term” and label the others as synonyms (see Table 2).

Challenges. Users often find contradictory descriptions or descriptions pertaining to different points of view, or different roles in the enterprise, as when, say, an accounting department describes “inventory” differently from a stock management department. In case of multiple descriptions, users can create a synthesis or, according to the objective and scope of the ontology, privilege one over the other. This decision is typically left to the ontology master or to the “wisdom of the crowd”; the glossary is therefore first published with terms having more than one description, leaving it to the social-validation phase to converge toward a unique term description.

Another challenge is related to synonyms that require deciding what is the “preferred term.” Voting, in this case, is a good way to achieve the result.

Step 3. Taxonomy Level

The first two knowledge levels reflect a terminological nature and exhibit a simple organization, a list of entries organized in alphabetical order. But the concepts denoted by the listed terms hide a rich conceptual organization users intend to represent through three different hierarchies. The first is a taxonomy based on the specialization relation, or the ISA relationship connecting a more specific concept to a more general one (such as invoice ISA business document). A taxonomy represents the backbone of an ontology, and its construction can be a challenge. It requires a good level of domain expertise and a consistent knowledge-modeling effort, since users must not only identify ISA relations between existing terms but also introduce more abstract terms or generic concepts seldom used in everyday life but that are extremely useful in organizing knowledge. During this step users thus provide feedback to the two previous knowledge levels—lexicon and glossary—since taxonomy building is also an opportunity to validate the two previous levels and extend them with new terms.

The example outlined in Table 3 is based on the use of a spreadsheet, where the specialization levels are organized, from left to right, in different columns.

Challenges. Defining a good taxonomy is difficult. Also difficult is organizing a flat list of concept names, or glossary terms, into a taxonomy. Care must be taken in considering different perspectives and opinions. The basic mechanism consists of the clustering of concepts, or terms, linking them to a more general concept (the bottom-up approach). Identifying a general concept is often not easy, and concepts can be clustered in different ways; in our simplified approach we avoid multiple generalization for a concept. Moreover, users must find a good balance between the breadth of the taxonomy, or average number of children of intermediate nodes, and its depth, or levels of specialization and the granularity of taxonomy leaves.

The ontology master plays an important role here, supported by numerous available resources (such as WordNet²⁵ and EuroVoc²⁶).

The UPON Lite approach involves three disjoint sub-hierarchies, one for each OPAL kind. Therefore, when users specialize a concept, as in, say, an object, its more specific concepts cannot likewise become an actor or a process.

For these challenges, a social approach is highly advisable, along the lines of a folksonomy.^{13,20}

Step 4. Predication Level

This step is similar to a database design activity, as it concentrates on the proper-

ties that, in the domain at hand, characterize the relevant entities. Users generally identify atomic properties (AP) and complex properties (CP). The former can be seen as printable data fields (such as unit price), and the latter exhibit an internal structure and have components (such as address composed of, say, street, city, postal code, and state). Finally, if a property refers to other entities (such as a customer referred to in an invoice) it is called a reference property (RP). In a relational database, an RP is represented by a foreign key. The resulting predicate hierarchy is organized with the entity at the top, then a property hierarchy below it, where nodes are tagged with CP, AP, and RP.

Continuing to use a spreadsheet, a user would build a table (see Table 4) where the first column reports the entities and the second the property name. In case of CP, the following columns on the right report the property components; in case of RP the referred entities are reported. Further information (such as cardinality constraints) can be added; for example, one invoice is sent to one and only one customer, who in turn may receive several invoices.

Challenges. Several decisions must be made in this step, starting with the granularity in representing properties. For instance, address can be a complex property, as covered earlier in this article, or can be an AP, where the

Table 3. Taxonomy excerpt with three specialization levels.

Top concept	First-level specialization	Second-level specialization
Business document	Invoice	
	Payment	Delayed payment
	Purchase order	
	Request for quotation	
Customer	Golden customer	
	Silver customer	

Table 4. Excerpt from a predication hierarchy.

Entity	Property	Sub-Property/Reference	Typing	Constraints
Invoice	Unit price [AP]		Currency value	
	Address [CP]	Street and number, city, state, Zip Code		
	Consignee [RP]	Customer		(1..1)
Customer	Name		String	
	Pending invoices	Invoice		(0..N)
	...			

whole address is encoded as one string. Likewise, an RP can be substituted by an AP if users adopt a “relational database” approach, viewing the property as the foreign key of the referenced entity (such as `customer` can be represented by `customer_code`). Other important points are represented by the typing of the AP (such as `string`, `integer`, and `Boolean`) and the cardinality constraints, or how many values a property can assume. Since UPON Lite is mainly for domain experts, typing may be too technical, such decisions can be delayed to a successive refinement cycle (mainly delegated to ontology engineers).

Step 5. Parthood Level

This step concentrates on the “architectural” structure of business entities, or parts of composite entities, whether objects, processes, or actors, by eliciting their decomposition hierarchy (or part-whole hierarchy). To this end a user would analyze the structure and components an entity exhibits, creating the hierarchy based on the `partOf` (inverse `hasPart`) relationship.

This hierarchy is particularly important in engineering and manufacturing and, more generally, when dealing with complex entities. For instance, a bill of material is a hierarchical decomposition of a product into its parts, subparts, and so on, until reaching elementary components, not further decomposable, representing the leaves of the decomposition tree, or, more precisely, a directed graph, generally acyclic.

Parthood can also be applied to immaterial entities (such as a regulation subdivided into sections and articles or a process subdivided into sub-processes and activities). In sub-processes and activities, users can enrich the model with other relations (such as “precedence” and “sync”), a subject beyond the scope of this article.

Challenges. In certain cases, users may have difficulty deciding if a hierarchical relation is `ISA` or `PartOf`. If we

consider price and then unit price, it is not evident if unit price is a special case of the former or part of it. Such a relationship is highly dependent on business and organizational choices. At an abstract level, we may say `ISA` is more suited, but if we have, say, an invoice where price means final price and is obtained through multiplying a quantity (number of pieces) by unit price, then the latter is indeed a component of the former. Another problem for users is to distinguish a part from a property. For instance, an invoice has a footer that may report the tax and the final price. From a structural point of view (such as when printing an invoice) the footer is considered a component of the invoice, but from the information point of view it can be considered a CP holding structured information. In general, usage context determines the relationship, and, eventually, social validation provides the final interpretation, with a central role for the ontology master.

Step 6. Ontology Level

Using the outcomes of the previous five steps, ontology engineers can proceed to build the final artifact—the ontology. It gathers all the knowledge collected in those steps, in particular the concepts in Step 2 and the semantic relations elicited in three steps: a taxonomy (Step 3), further enriched by the predication (Step 4), and parthood (Step 5), as required. In addition, it is also possible to formally represent constraints (such as typing and cardinality constraints) and the needed domain-specific relations (such as `provides(Supplier, Product)`). Many of these relations can be obtained through “upgrading” the RP introduced in Step 4. To continue with the tabular approach, domain relations are represented in a new table with three columns, with an implicit orientation left to right (see Table 5).

The final step in implementing a “full-fledged” ontology is encoding it

through a formal language like the Resource Description Framework (RDF) and OWL.¹⁰ This is a technical task that requires the direct intervention of ontology engineers. However, a number of technical innovations aim to facilitate it to further increase end-user involvement. For instance, the semantic suite Anzo Enterprise (from Cambridge Semantics)²⁷ provides a plug-in for Excel that enables Excel spreadsheets to be mapped to an ontology, transforming the related data into RDF. Also along these lines is support (though less extensive) provided for the popular Protégé ontology system by the MappingMaster OWL plugin.²⁸

A crucial aspect of the developed ontology is its quality, which is evaluated according to some predefined criteria. UPON Lite adheres to such an approach based on four perspectives: syntactic, semantic, pragmatic, and social.³ In UPON Lite the syntactic and social quality of the produced ontology is addressed through the stepwise approach and social collaboration in knowledge encoding based on a shared online spreadsheet. Semantic quality concerns the absence of contradictions, a property that can be checked through specially designed software (such as the reasoners available with Protégé, including RacerPro,²⁹ Pellet,³⁰ and FaCT++³¹), once the ontology is imported through the MappingMaster plugin. The pragmatic quality of the ontology is guaranteed through the close involvement of end users in the whole ontology-building process.

Challenges. The first challenge concerns the choice of how expressive ontology engineers want the ontology to be. A minimal option is to include the taxonomy and the reference properties in the form of domain-specific relations. They can also add a number of constraints (such as typing and cardinality constraints). The second challenge concerns encoding the ontology in formal terms. Here, the role of the ontology engineer is central due to the high level of expertise required by this step.

Related Work and Evaluation

Among the relevant methodologies of ontology engineering in the literature (such as Methontology,⁸ On-To-Knowledge,¹⁸ and DILIGENT¹⁹), none was expressly conceived for rapid ontology prototyping; each has a dif-

Table 5. Domain-specific relations (excerpt).

Entity	Relation	Entity
Supplier	provides	Product
Invoice	paidBy	Client
Product	providedBy	Supplier
...	...	...

ferent purpose and scope, aiming at development of industrial-strength ontologies. For comparative studies of relevant ontology-engineering methods see De Nicola et al.⁷ and Chimienti et al.⁴ In 2013, GOSPL⁶ was proposed as a collaborative ontology-engineering methodology aimed at building hybrid ontologies, or carrying informal and formal concepts. Finally, the NeON methodology¹⁶ was conceived for developing ontology networks, introducing two different ontology-network-life-cycle models—Waterfall and Iterative-Incremental. They are more sophisticated and complex than UPON Lite and designed for ontology engineers; only DILIGENT and GOSPL explicitly address collaboration between domain experts and ontology engineers.

UPON Lite has been developed over the past 15 years through constant experimentation in research and industrial projects, as well as in a number of university courses. Beyond experimental evaluation, we also carried out a comparative assessment against existing ontology-engineering methodologies, using an evaluation method conceived for rapid ontology engineering based on 10 key features:

Social and collaborative aspects of ontology development. Considering the extent social and collaborative processes are included in the methodology;

Domain expert orientation. Referring to the extent the methodology allows domain experts to build and maintain an ontology without support of ontology engineers;

Cost efficiency. Concerning the focus of the methodology on cost reduction;

Supporting tools. Referring to the extent the methodology suggests tools to ease ontology development;

Adaptability. Referring to whether the methodology is flexible enough to be adopted in different industrial applications;

Reusability. Referring to the extent the methodology considers the possibility of reusing existing resources;

Stepwise and cyclic approach. Representing how much the methodology is based on an incremental cyclic process, avoiding a rigid “waterfall” linear model;

Natural language. Referring to the

Table 6. Ontology-engineering methodologies compared.

	(F.1)	(F.2)	(F.3)	(F.4)	(F.5)	(F.6)	(F.7)	(F.8)	(F.9)	(F.10)	Ranking*
UPON Lite	H	H	H	H	H	H	H	H	M	M	9.3
Diligent	H	H	M	H	H	H	M	H	M	H	9.0
UPON	L	M	L	M	H	H	H	H	M	L	7.0
NeON_{wf}	M	M	H	M	M	H	L	M	L	L	6.3
GOSPL	H	M	L	M	L	H	M	H	L	L	6.3
NeON_{in}	M	L	L	M	M	H	H	M	L	M	6.3
OntoKnowledge	L	L	L	M	H	M	H	L	L	H	6.0
Methontology	L	L	M	M	M	M	M	H	M	L	6.0

*Computed as normalized average, with H = 3, M = 2, and L = 1

Figure 2. Excerpt from a Google Docs UPON Lite spreadsheet.

	A	B	C	D
	Term	Synonyms	Kind	Description [source]
1	Delivery address	Shipping address	Complex property	Location to which goods are to be sent [1].
2	Invoice	Bill	Object	Itemized list of goods shipped, usually specifying the price and the terms of sale [2].
3	Postal address	Address	Complex property	Information that locates and identifies a specific address, as defined by the postal service [3].
4	Purchasing conditions	Purchase terms and conditions	Object	Conditions related to the transaction and the trade [4].
5	Purchase order	PO	Object	Commercial document issued by a buyer to a seller, indicating types, quantities, and agreed prices for products or services the seller will provide to the buyer [5].
6	Customer	Client	Actor	One who purchases a commodity or service [2].
7	Invoicing	Issuing invoice	Process	Making or issuing an invoice for goods or services [6].
8	Purchasing	Buying	Process	Acquisition of something for payment [6].

extent the methodology uses natural languages resources, processing techniques, and tools;

Documentation. Concerning production of supporting documentation and the extent an intermediate artifact can be considered a valuable documentation; and

Organizational structure. Referring to the extent project management methods are included in the methodology.

We defined them along the lines of other comparison methods in the literature, with an orientation toward rapid ontology engineering. Table 6 refers to the NeON methodology as NeON_{wf} for the waterfall engineering model and NeON_{in} for the iterative-incremental model. The comparative evaluation for rapid ontology engineering shows UPON Lite and DILIGENT outperform

the other methodologies, with a slight edge to UPON Lite.

Conclusion

We have proposed UPON Lite, an ontology-engineering methodology based on a lean, incremental process conceived to enhance the role of end users and domain experts without specific ontology-engineer expertise. Aiming to support rapid prototyping of trial ontologies, UPON Lite is a derivation of the full-fledged UPON Methodology. UPON Lite is characterized by three main aspects of ontology engineering: a user-centered approach conceived to be easily adopted by non-ontology experts, thus minimizing the role of ontology engineers; a socially oriented approach, where multiple stakeholders play a central role; and an intuitive ontology-

engineering process organized in six steps, supported by a “familiar” tool like a spreadsheet. Here, the spreadsheet tables are represented in cursory form; see Figure 2 for an actual excerpt from a Google Docs³² spreadsheet we used in the running example.

UPON Lite has been used in industrial scenarios and university courses since 2001, producing more than 20 ontologies involving from a few domain experts up to 100, where non-ontology experts formed the great majority of ontology teams. The mean time of the ontology-building process varied from a week (in university courses) to a few months in industrial projects (not including maintenance). Since 2008, the methodology has been adopted by two European Union projects: Collaboration and Interoperability for Networked Enterprises in an Enlarged European Union (COIN) and Business Innovation in Virtual Enterprise Environments (BIVÉE). COIN developed a trial ontology for the Andalusia Aeronautic cluster for a furniture ecosystem for the Technology Institute on Furniture, Wood, Packaging and related industries in Spain, and for the robotics sector, with Loccioni, in Italy. A national Italian project, E-Learning Semantico per l'Educazione continua nella medicina (ELSE), developed a trial ontology for lifelong education of medical doctors in the domain of osteoporosis. The feedback, collected through interviews and working meetings, covered various aspects of the methodology, from usability to efficiency and adoptability to flexibility; the results are encouraging. The field experiences in all these projects reflect the feasibility of stakeholders and end users producing good (trial) ontologies in a short time. Furthermore, the direct involvement of domain experts reduced the need for interaction with ontology engineers, as required by traditional methodologies, even for small ontology changes.

Involving communities of practice helps reduce the time and cost of rapid ontology prototyping. The UPON Lite stepwise approach has proved beneficial for the learning curve of domain experts new to the methodology, allowing them to quickly learn the process and its intuitive outcomes, including lexicon, glossary, taxonomy, predication, and parthood.

The UPON Lite approach advocates

use of a social media platform and an online, shared spreadsheet as key tools; it also suggests other tools for supporting the methodology's first steps, including text mining, gloss extractors, and MappingMaster, can be used to automatically import spreadsheet content into an ontology editor (such as Protégé), thus producing an OWL ontology.

We performed a comparative evaluation of UPON Lite against the most popular ontology-engineering methodologies, as well as in the context of rapid development of a “lightweight” ontology. The results of the comparative assessment show UPON Lite offers the best solution.

In the future we intend to work on systematic monitoring of the adoption and use of UPON Lite in different domains, focusing on the problems that will emerge during the ontology-engineering process and how they should be addressed, considering the complexity of the ontology and the number of stakeholders involved.

Acknowledgment

This work is partially supported by European project BIVÉE grant number FP7-ICT-285746. C

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. DBpedia: A nucleus for a Web of open data. In *Proceedings of the Sixth International Semantic Web Conference and Second Asian Semantic Web Conference* (Busan, Korea, Nov. 11–15). Springer-Verlag, Berlin, Heidelberg, Germany, 2007, 722–735.
2. Barbagallo, A., De Nicola, A., and Missikoff, M. eGovernment ontologies: Social participation in building and evolution. In *Proceedings of the 43rd Hawaii International Conference on System Sciences* (Honolulu, HI, Jan. 5–8). IEEE Computer Society, 2010, 1–10.
3. Burton-Jones, A., Storey V.C., Sugumaran, V., and Ahluwalia P. A semiotic metrics suite for assessing the quality of ontologies. *Data & Knowledge Engineering* 55, 1 (Oct. 2005), 84–102.
4. Chimienti, M., Dassisti, M., De Nicola, A., and Missikoff M. Evaluation of ontology building methodologies: A method based on balanced scorecards. In *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, J.L.G. Dietz, Ed. (Madeira, Portugal, Oct. 6–8). INSTICC Press, 2009, 141–146.
5. D'Antonio, F., Missikoff, M., and Taglino, F. Formalizing the OPAL eBusiness ontology design patterns with OWL. In *Proceedings of the Third International Conference on Interoperability for Enterprise Software and Applications* (Madeira, Portugal, Mar. 27–30). Springer, London, U.K., 2007, 345–356.
6. Debruyne, C., Tran, T.-K., and Meersman, R. Grounding ontologies with social processes and natural language. *Journal on Data Semantics* 2, 2-3 (June 2013), 89–118.
7. De Nicola, A., Missikoff, M., and Navigli, R. A software engineering approach to ontology building. *Information Systems* 34, 2 (Apr. 2009), 258–275.
8. Fernández-López, M., Gómez-Pérez, A., and Juristo, N. Methontology: From ontological art towards ontological engineering. In *Proceedings of the AAAI Spring Symposium Series* (Stanford, CA, Mar. 24–26). AAAI Press, Menlo Park, CA, 1997, 33–40.
9. Keet, C.M. and Artale, A. Representing and reasoning

- over taxonomy of part-whole relationships. *Applied Ontology* 3, 1-2 (Jan. 2008), 91–110.
10. McGuinness, D.L. and van Harmelen, F. *OWL 2 Web Ontology Language Overview*. W3C Recommendation. W3C, Cambridge, MA, 2003; <http://www.w3.org/TR/owl2-overview/>
11. Noy, N.F. and McGuinness, D.L. *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford University, Stanford, CA, 2001; http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
12. Parveen, A., Habib, A., and Sarwar, S. Scope and limitation of electronic voting system. *International Journal of Computer Science and Mobile Computing* 2, 5 (May 2013), 123–128.
13. Pink, D.H. Folksonomy. *New York Times* (Dec. 11, 2005); http://www.nytimes.com/2005/12/11/magazine/11ideas1-21.html?_r=0
14. Sclano, F. and Velardi, P. TermExtractor: A Web application to learn the common terminology of interest groups and research communities. In *Proceedings of the Seventh Terminology and Artificial Intelligence Conference* (Sophia Antipolis, France, Oct. 8–9). Presses Universitaires de Grenoble, 2007, 85–94.
15. Spyns, P., Tang, Y., and Meersman, R. An ontology engineering methodology for DOGMA. *Applied Ontology* 3, 1-2 (Jan. 2008), 13–39.
16. Suárez-Figueroa, M.C., Gómez-Pérez, A., and Fernández-López, M. The NeOn methodology for ontology engineering. In *Ontology Engineering in a Networked World*. Springer, Berlin, Heidelberg, Germany, 2012, 9–34.
17. Sugumaran, V. and Storey, V.C. The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM Transactions on Database Systems* 31, 3 (Sept. 2006), 1064–1094.
18. Sure, Y., Staab, S., and Studer, R. On-To-Knowledge Methodology (OTKM). In *Handbook on Ontologies*, S. Staab and R. Studer, Eds. Springer, Berlin, Heidelberg, Germany, 2004, 117–132.
19. Tempich, C., Simperl, E., Luczak, M., Studer, R., and Pinto, H.S. Argumentation-based ontology engineering. *IEEE Intelligent Systems* 22, 6 (Nov.-Dec. 2007), 52–59.
20. Van Damme, C., Hepp, M., and Siorpaes, K. Folksonomy: An integrated approach for turning folksonomies into ontologies. In *Proceedings of the Workshop Bridging the Gap Between Semantic Web and Web 2.0 of the Fourth European Semantic Web Conference* (Innsbruck, Austria, June 7), 2007, 57–70.
21. Velardi, P., Cucchiarelli, A., and Petit M. A taxonomy learning method and its application to characterize a scientific Web community. *IEEE Transactions on Knowledge and Data Engineering* 19, 2 (Feb. 2007), 180–191.

Web References

22. Universal Business Language; <https://www.oasis-open.org/committees/ubl/>
23. AlchemyAPI; <http://www.alchemyapi.com/>
24. Open Calais; <http://www.opencalais.com/>
25. WordNet; <http://wordnet.princeton.edu/>
26. EuroVoc; <http://eurovoc.europa.eu/>
27. Anzo; <http://www.cambridgesemantics.com/>
28. Protégé; <http://protege.cim3.net/cgi-bin/wiki.pl?MappingMaster>
29. RacerPro; <http://franz.com/agraph/racer/>
30. Pellet; <http://clarkparsia.com/pellet/>
31. FaCT++; <http://owl.man.ac.uk/factplusplus/>
32. Google Docs; <https://docs.google.com/>

Antonio De Nicola (antonio.denicola@enea.it) is a researcher in the Laboratory for the Analysis and Protection of Critical Infrastructures of the Smart Energy Division at the Italian National Agency for New Technologies, Energy, and Sustainable Economic Development, Rome, Italy, and a Ph.D. candidate in computer science, control, and geoinformation at the University of Tor Vergata, Rome, Italy.

Michele Missikoff (michele.missikoff@cnr.it) is an adjunct professor at the University of International Studies of Rome and an associate researcher at the Italian National Research Council, Semantic Technology Lab at the Institute for Cognitive Sciences and Technologies, Rome, Italy.

Copyright held by authors.
Publication rights licensed to ACM. \$15.00

Systor2016

**The 9th ACM International
Systems and Storage Conference**

June 6 – 8 • Haifa, Israel

We invite you to submit original and innovative papers, covering all aspects of computer systems technology, such as file and storage technology; operating systems; distributed, parallel, and cloud systems; security; virtualization; and fault tolerance, reliability, and availability. SYSTOR 2016 accepts full papers, short papers, and posters.

Paper submission deadline: March 11, 2016

Poster submission deadline: April 8, 2016

Program chairs

Mark Silberstein, Technion
Emmett Witchel, University of Texas

General chair

Katherine Barabash, IBM Research

Posters chair

Anna Levin, IBM Research

Steering committee head

Michael Factor, IBM Research

Steering committee

Ethan Miller, University of California
Santa Cruz

Liuba Shrira, Brandeis University

Dan Tsafir, Technion

Dalit Naor, IBM Research

Erez Zadok, Stony Brook University

www.systor.org/2016/

Sponsored by



SIGOPS

In cooperation with



usenix
THE ADVANCED
COMPUTING SYSTEMS
ASSOCIATION



Platinum sponsor



Gold sponsors



Sponsors



What does it mean to be secure?

BY BOAZ BARAK

Hopes, Fears, and Software Obfuscation

COMPUTER PROGRAMS ARE arguably the most complex objects ever constructed by humans. Even understanding a 10-line program (such as the one depicted in Figure 1) can be extremely difficult. The complexity of programs has been the bane (as well as the boon) of the software industry, and taming it has been the objective of many efforts in industry and academia. Given this, it is not surprising that both theoreticians and practitioners have been trying to “harness this complexity for good” and use it to protect sensitive information and computation. In its most general form this is known as *software obfuscation*, and it is the topic of this article.

In a certain sense, any cryptographic tool such as encryption or authentication can be thought of as harnessing complexity for security, but with software obfuscation people have been aiming for something far more ambitious: a way to transform *arbitrary* programs into an “inscrutable” or *obfuscated* form. By this we do not mean reverse engineering the program should be cumbersome but rather it should be *infeasible*, in the same way that recovering the

plaintext of a secure encryption cannot be performed using any reasonable amount of resources.

Obfuscation, if possible, could be the cryptographer’s “master tool”—as we will see it can yield essentially any crypto application one can think of. Therefore, it is not surprising that both practitioners and theoreticians have been trying to achieve it for a great while. However, over the years many practical attempts at obfuscation (such as the DVD Content Scrambling System) had been broken (for example, Green¹⁸ and Jacob et al.¹⁹) Indeed in 2001, in work with Impagliazzo, Goldreich, Rudich, Sahai, Vadhan, and Yang,³ we proved that achieving such a secure transformation is *impossible*. So, why isn’t this the shortest survey in *Communications* history?

The key issue is what does it mean to be “secure.” In our 2001 work, we proved impossibility of a notion of security for such “obfuscating compilers,” which we termed as *virtual black box security* and will describe more formally next. While virtual black-box is arguably the most natural notion of security for obfuscators, it is not the only one. Indeed, in the same work³ we suggested a weaker notion called *indistinguishability obfuscation* or IO for short. We had no idea if IO can be achieved, nor if it is useful for many of the intended applications. But in 2013 Garg et al.¹³ used some recent cryptographic advances to give a *candidate construction* of obfuscators satisfying the IO

» key insights

- Software obfuscation can potentially yield the “ultimate cryptographic tool” enabling heretofore out of reach applications for security and privacy.
- For a long time, we did not even have a theoretical approach to constructing obfuscation, and had some reasons to suspect it does not exist.
- Exciting new work finally puts forward such an approach, but much additional research will be needed before it reaches sufficient efficiency and security for practice.

```
def isprime(n):  
    return all(p % i != 0 for i in range(2, n-1))  
  
def Goldbach(n):  
    return any((isprime(p) and isprime(n-p))  
              for p in range(2, n-1))
```

```
n = 4  
while True:  
    if not Goldbach(n): break  
    n += 2  
print "Hello world!"
```

```
def isprime(p):  
    return all(p % i != 0 for i in range(2, p-1))  
  
def Goldbach(n):  
    return any((isprime(p) and isprime(n-p))  
              for p in range(2, n-1))
```

```
n = 4  
while True:  
    if not Goldbach(n): break  
    n += 2  
print "Hello world!"
```

definition. Moreover, their work, and many followup works, have shown this weaker notion is actually extremely useful, and can recover many (though not all) the applications of virtual black-box obfuscators. These applications include some longstanding cryptographic goals that before Garg et al.'s work seemed far out of reach, and so the cryptographic community is justifiably excited about these new developments, with many papers and several publicly funded projects devoted to exploring obfuscation and its applications.

What is an indistinguishability obfuscator? How is it useful? and What do I mean by a “candidate construction?” Read the rest of this article to find out.

Obfuscating Compilers and Their Potential Applications

Obfuscation research is in an “embryonic stage” in the sense that so far we only have theoretical proofs of concept that are extremely far from practical efficiency. Even with the breakneck pace of research on this topic, it may take years, if not decades, until such obfuscators can be deployed at scale, and as we will see, beyond the daunting practical issues there are some fundamental

theoretical challenges we must address as well. Thus, while eventually one might hope to obfuscate large multi-part programs, in this article we focus on the task of obfuscating a single function, mapping an input to an output without any side effects (though there is recent research on obfuscating more general computational models). Similarly, given that the overhead in translating a program from one (Turing complete) programming language to another pales in comparison to the current inefficiencies, for the purposes of this article we can imagine that all programs are represented in some simple canonical way.

An *obfuscating compiler* is an algorithm that takes as input a program P and produces a functionally equivalent program P' . So far, this does not rule out the compiler that simply outputs its input unchanged, but we will want the program P' to be “inscrutable” or “obfuscated.” Defining this requirement formally takes some care, and as we will see, there is more than one way to do so. Our guiding principle is that the output program P' will not reveal any more information about the input P than is necessarily revealed by the fact the two

programs are functionally equivalent. To take two extreme examples, if you wrote a program P that outputs your credit card number when given the number 0 as input, then no matter how obfuscated P' is, it will still reveal the same information. In contrast, if the program P contained the credit number as a comment, with no effect on its functionality, then the obfuscated version P' should not reveal it. Of course, we want an obfuscator compiler to do much more than strip all comments. Specifically, we say a compiler transforming P to P' is *virtual black-box secure* if for any attacker A that learns some piece of information x from P' , x could have been learned by simply treating P' (or P) as a *black box* and querying it on various inputs and observing the outputs. More formally, the definition requires that for every (polynomial-time) algorithm A , there is another polynomial time algorithm S (known as a *simulator* in crypto parlance) such that the random variables $A(P')$ and S^P are computationally indistinguishable, where $A(P')$ denotes the output of A given the code of the obfuscated program P' , while S^P denotes the output of S given black-box (i.e., input/output) access to P (or to the functionally equivalent P').

Let us imagine that we had a practically efficient obfuscating compiler that satisfied this notion of virtual black-box security. What would we use it for? The most obvious application is software protection—publishing an obfuscated program P' would be the equivalent of providing a physical “black box” that can be executed but not opened up and understood. But there are many other applications. For example, we could use an obfuscator to design a “selective decryption scheme”^a—a program that would contain inside it a decryption key d but would only decrypt messages satisfying very particular criteria. For example, suppose that all my email was encrypted, and I had an urgent project that needed attention while I was on vacation. I could write a program P , such as the one of Figure 2, that given an encrypted email message as

^a The technical name for this notion is *functional encryption*.⁵

Figure 1. The following Python program prints “Hello world!” if and only if Goldbach’s conjecture is false.

```
def isprime(p):
    return all(p % i for i in range(2,p-1))

def Goldbach(n):
    return any( (isprime(p) and isprime(n-p))
               for p in range(2,n-1))

n = 4
while True:
    if not Goldbach(n): break
    n+= 2
print "Hello world!"
```

Figure 2. A “selective decryption” program. Black-box access to this program enables a user to decrypt only messages that match some particular pattern, and hence obfuscating such a program can be used to obtain a functional encryption scheme.

```
def DecryptEmail(EncryptedMsg):
    SecretKey = "58ff29d6ad1c33a00d0574fe67e53998"
    m = Decrypt(EncryptedMsg, SecretKey)
    if m.find("Foosball table")>=0: return m
    return "Sorry Yael, this email is private"
```

input, uses my secret decryption key to decrypt it, checks if it is related to this project and if so outputs the plaintext message. Then, I could give my colleague an obfuscated version P' of P , without fearing she could reverse engineer the program, learn my secret key and manage to decrypt my other email messages as well.

There are many more applications for obfuscation. The example of functional encryption could be vastly generalized. In fact, *almost any cryptographic primitive you can think of* can be fairly directly derived from obfuscation, starting from basic primitives such as public key encryption and digital signatures to fancier notions such as multiparty secure computation, fully homomorphic encryption, zero knowledge proofs, and their many variants. There are also applications to obfuscation that a priori seem to have nothing to do with cryptography; for example, one can use it to design autonomous agents that would participate on your behalf in digital transactions such as electronic auctions, or to publish patches for software vulnerabilities without worrying that attackers could learn the vulnerabilities by reverse engineering the patch.

So, virtual black-box obfuscation is wonderful, but does it exist? This is what we set to find out in 2001, and as already mentioned, our answer was negative. Specifically, we showed the existence of *inherently unobfuscatable functions*—this is a program P whose source code can be recovered from *any* functionally equivalent program P' though curiously it *cannot* be efficiently recovered using only black-box access to P .

In the intervening years, cryptography has seen many advances, in particular achieving constructions of some of the cryptographic primitives that were envisioned as potential applications of obfuscation, most notably *fully homomorphic encryption*¹⁴ (see the accompanying sidebar). In particular, in 2012 Garg, Gentry and Halevi¹² put forward a candidate construction for an object they called “cryptographic multilinear maps,” and which in this article I will somewhat loosely refer to as a “homomorphic quasi encryption” scheme. Using



Obfuscation research is in an “embryonic stage” in the sense that so far we only have theoretical proofs of concept that are extremely far from practical efficiency.



this object, Garg et al.¹³ showed a candidate construction of a general-purpose *indistinguishability obfuscators*.^b

Indistinguishability Obfuscators

An *indistinguishability obfuscator* (IO) hones in on one property of virtual black box obfuscators. Suppose P and Q are two functionally equivalent programs. It is not difficult to verify that virtual black-box security implies an attacker should not be able to tell apart the obfuscation P' of P from the obfuscation Q' of Q . Indistinguishability obfuscation requires only this property to hold.

Indistinguishability obfuscators were first defined in our original paper,³ where we noted this notion is weak enough to avoid our impossibility result, but we did not know whether or not it can be achieved. Indeed, a priori, one might think that indistinguishable obfuscators capture the “worst of both worlds.” On one hand, while the relaxation to IO security does allow to avoid the impossibility result, such obfuscators still seem incredibly difficult to construct. For example, assuming Goldbach’s Conjecture is correct, the IO property implies the obfuscation of the Goldbach(n) subroutine of the program in Figure 1 should be indistinguishable from the obfuscation of the function that outputs True on every even $n > 2$; designing a compiler that would guarantee this seems highly non-trivial. On the other hand, it is not immediately clear that IO is useful for concrete applications. For example, if we consider the “selective decryption” example mentioned previously, it is unclear that the IO guarantee means that obfuscating the program P that selectively decrypts particular messages would protect my secret key. After all, to show it does, it seems we would need to show there is a functionally equivalent program P' that does not leak the key (and hence by the IO property, since P and P' must have indistinguishable obfuscations, the obfuscation of P would protect the key

^b Even prior to the works by Garg et al.^{12, 13} there were papers achieving virtual black-box obfuscation for very restricted families of functions. In particular, independently of Garg et al.¹³, Brakerski and Rothblum⁷ used Garg et al.’s¹² construction to obtain virtual black-box obfuscation for functions that can be represented as *conjunctions* of input variables or their negations.

as well). But if we knew of such a P' , why did not we use it in the first place?

It turns out both these intuitions are (probably) wrong, and that in some sense IO may capture the “best of both worlds.” First, as I mentioned, despite the fact that it seems so elusive, Garg et al.¹³ did manage to give a candidate construction of indistinguishability obfuscators. Second,¹³ managed to show that IO is also *useful* by deriving functional encryption from it, albeit in a less direct manner. This pattern has repeated itself several times since, with paper after paper showing that many (though not all) of the desirable applications of virtual black-box obfuscation can be obtained (using more work) via IO. Thus, indistinguishability obfuscation is emerging as a kind of “master tool” or “hub” of cryptography, from which a great many of our other tools can be derived (see Figure 3).

Now all that is left is to find out how do we construct this wonderful object, and what is this caveat of “candidate construction” I keep mentioning?

I will start with describing the construction and later turn to discussing the caveat. Unfortunately, the construction is rather complex. This is both in the colloquial sense of being complicated to describe (not to mention implement) and in the computational complexity sense of requiring very large (though still polynomial) space and time resources. Indeed, this complexity is the main reason these constructions are still at the moment theoretical “proof of concepts,” as opposed to practical compilers. The only implementation of obfuscators I know of at the time of this writing was by Apon et al.,¹ and their obfuscation blows up a circuit of 160R gates to 31GB. (The main source of inefficiency arises from the constructions of “homomorphic quasi-encryption schemes” described here.) That said, making these schemes more efficient is the object of an intensive research effort, and I am sure we will see many improvements in the coming years. It is a testament to the excitement of this field that in the short time after the first candidate construction of IO, there are already far more works than I can mention that use IO for exciting applications, study its security or efficiency, consider different notions of obfuscation, and more.



The astute reader might notice that fully homomorphic encryption is an immediate consequence of (virtual black-box) obfuscation combined with any plain-old encryption.



While I will not be able to describe the actual construction, I do hope to give some sense into the components that go into it, and the rather subtle questions that arise in exploring its security.

Fully Homomorphic Encryption and “Quasi-Encryption”

In 2009, Craig Gentry rocked the world of cryptography by presenting a construction for *fully homomorphic encryption scheme*. What is this object? Recall that a traditional encryption scheme is composed of two functions: the encryption operation Enc —mapping the secret plaintexts into the “scrambled” cyphertexts—and the decryption operation Dec that performs the inverse of Enc (and requires the secret decryption key to compute). A fully homomorphic encryption supports two additional operations $\oplus$, $\otimes$ which correspond to “multiplying” and “adding” ciphertexts. Specifically they satisfy the equations for every $a, b \in \{0, 1\}$:

$$\text{Enc}(a) \oplus \text{Enc}(b) = \text{Enc}(a + b \pmod{2})$$

$$\text{Enc}(a) \otimes \text{Enc}(b) = \text{Enc}(ab \pmod{2})$$

Since

$$\text{NOT } a = 1 + a \pmod{2}$$

$$a \text{ AND } b = ab \pmod{2}$$

$$a \text{ OR } b = a + b + ab \pmod{2}$$

these two operations allow us to compute from encryptions $\text{Enc}(a_1), \dots, \text{Enc}(a_n)$ the value $\text{Enc}(P(a_1, \dots, a_n))$ given any program P that maps $\{0, 1\}^n$ to $\{0, 1\}$. Note that crucially, the $\otimes$ and $\oplus$ operations do *not* require knowledge of the secret key to be computed (indeed, otherwise they would be trivial to implement by writing

$$c \oplus c' = \text{Enc}(\text{Dec}(c) + \text{Dec}(c') \pmod{2}) \quad (1)$$

$$c \otimes c' = \text{Enc}(\text{Dec}(c) \text{Dec}(c') \pmod{2}) \quad (2)$$

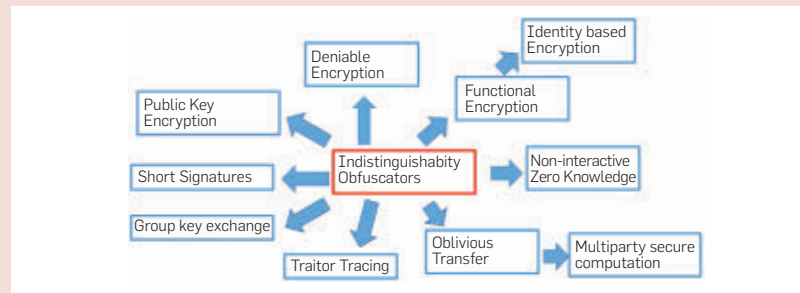
Fully homomorphic encryption was first envisioned in 1978 by Rivest et al.,²³ but they gave no constructions and for many years it was not at all clear whether the existence of such operations is compatible with security until Gentry¹⁴ came up with the first plausibly secure construction. Rivest et al.²³ were motivated by client-server applications (now known as “cloud computing”). Indeed, one can see that such an encryption scheme could be very useful in this setting, where for

example a client could send to the server an encryption $\text{Enc}(a) = \text{Enc}(a_1) \cdots \text{Enc}(a_n)$ of its private data a , so the server could use the $\oplus$ and $\otimes$ operations to compute some complicated program P on this encryption and return $\text{Enc}(P(a))$ to the client, without ever learning anything about a .

The astute reader might notice that fully homomorphic encryption is an immediate consequence of (virtual black-box) obfuscation combined with any plain-old encryption. Indeed, if secure obfuscation existed then we could implement $\oplus$ and $\otimes$ by obfuscating their trivial programs (1) and (2). One might hope that would also work in the other direction—perhaps we could implement obfuscation using a fully homomorphic encryption. Indeed, let F be the “program interpreter” function that takes as input a description of the program P and a string a and maps them to the output $P(a)$. Perhaps we could obfuscate the program P by publishing an encryption $P' = \text{Enc}(P)$ of the description of P via a fully homomorphic encryption. The hope would be we could use P' to evaluate P on input a by encrypting a and then invoking F on the encrypted values $\text{Enc}(P)$ and $\text{Enc}(a)$ using the homomorphic operations. However, a moment's thought shows if we do that, we would not get the value $P(a)$ but rather the *encryption* of this value. Thus P' is not really a functionally equivalent form of P , as (unless one knows the decryption key) access to P' does not allow us to compute P on chosen inputs. Indeed, while fully homomorphic encryption does play a part in the known constructions of obfuscation, they involve many other components as well.

In some sense, the problem with using a fully homomorphic encryption scheme is it is “too secure.” While we can perform various operations on ciphertexts, without knowledge of the secret key we do not get any information at all about the plaintexts, while obfuscation is all about the “controlled release” of particular information on the secret code of the program P . Therefore, the object we need to construct is what I call a “fully homomorphic quasi-encryption” which is a riff on an encryption scheme that is in some sense less secure but more

Figure 3. In two years since the candidate construction, Indistinguishability Obfuscation is already emerging as a “hub” for cryptography, implying (when combined with one-way functions) a great many other cryptography primitives.



versatile than a standard fully homomorphic encryption.^c

Homomorphic quasi-encryption. A “fully homomorphic quasi-encryption scheme” has the same Enc , Dec , $\otimes$ and $\oplus$ operations as a standard fully homomorphic encryption scheme, but also an additional operation $\stackrel{?}{=}$ that satisfies that $\text{Enc}(a) \stackrel{?}{=} \text{Enc}(b)$ is true if $a = b$ and equals false otherwise. Moreover, instead of $\{0, 1\}$, the plaintexts will be in $\{0, 1, \dots, p-1\}$ for some very large prime p (of a few thousand digits or so), and the $\oplus$ and $\otimes$ operations are done modulo p instead of modulo 2. Note that a quasi-encryption scheme is less secure than standard encryption, in the sense the $\stackrel{?}{=}$ operation allows an attacker to perform tasks, such as discovering two ciphertexts correspond to the same plaintext, that are infeasible in a secure standard encryption. Indeed, the notion of security for a quasi-encryption scheme is rather subtle and is very much still work in progress.^d A necessary condition is one should not be able to recover a from $\text{Enc}(a)$ but this is in no way sufficient. For now, let us say the quasi-encryption scheme is secure if an attacker cannot learn anything about the plaintexts beyond

what could be learned by combining the $\otimes$, $\oplus$ and $\stackrel{?}{=}$ operations.^e

One example of a “partially homomorphic quasi-encryption” scheme is *modular exponentiation*. That is, given some numbers g, q such that $g^p = 1 \pmod{q}$, we can obtain a quasi-encryption scheme supporting only $\oplus$ (and not $\otimes$) by defining $\text{Enc}(a) = g^a \pmod{q}$ for $a \in \{0, \dots, p-1\}$ (with Dec its inverse—i.e., the discrete log modulo q).^f We define $c \oplus c' = cc' \pmod{q}$, which indeed satisfies that $g^a \oplus g^{a'} = g^{a+a'}$, and define $\stackrel{?}{=}$ to simply check if the two ciphertexts are equal. Modular exponentiation has been the workhorse of cryptography since Diffie and Hellman (following a suggestion of John Gill) used it as a basis for their famous key exchange protocol.¹¹ In 2000 Joux²⁰ suggested (using different language) to use exponentiation over elliptic curve groups that support the so called “Weil and Tate pairings” to extend this quasi-encryption scheme to support a single multiplication. Surprisingly even a single multiplication turns out to be extremely useful and a whole sub-area of cryptography, known as “pairing-based cryptography,” is devoted to using these partially homomorphic quasi-encryptions for a great many applications. But the grand challenge of this area has been to obtain *fully homomorphic quasi-encryption*⁶ (or in their language a *multi-linear map*, as opposed

^c This is a non-standard name used for this exposition; the technical name is a *cryptographic multilinear map* or a *graded encoding scheme*.

^d In fact, the same paper Garg et al.¹² proposing the first candidate construction for such a quasi-encryption scheme also gave an attack showing their scheme does *not* satisfy some natural security definitions, and followup works extended this attack to other settings. Finding a candidate construction meeting a clean security definition that suffices for indistinguishability obfuscation is an important open problem.

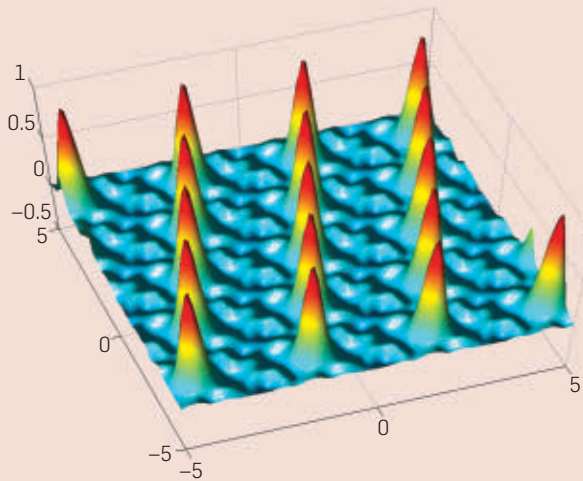
^e As an aside, a similar notion to quasi-encryption, without homomorphism, is known as *deterministic encryption* and is used for tasks such as performing SQL queries on encrypted data bases (for example, see Popa et al.²¹).

^f The Dec operation is not efficiently computable, but this turns out not to be crucial for many of the applications.

Lattice-based cryptography and fully homomorphic encryption

While the integer factoring problem is perhaps the most well known mathematical basis for cryptosystems, many recent constructions, including those used by fully homomorphic encryption and obfuscation, use computational problems related to *integer lattices*.

The fundamental observation behind these problems is that classical linear algebraic algorithms such as Gaussian elimination are incredibly *brittle* in the sense they cannot handle even slight amounts of noise in their data. One concrete instantiation of this observation is Regev's *Learning With Errors* (LWE) conjecture²² that there is no efficient algorithm that can recover a secret random vector $x \in \{0, \dots, p-1\}^n$ given noisy linear equations on x (such as, a random matrix A and the vector $y = Ax + e \pmod{p}$ where e is a random error vector of small magnitude). This has been shown to be essentially equivalent to the question of trying to “error correct” a vector in $\mathbb{R}^n$ that sampled from a distribution that is very close to, but not exactly contained in, a discrete subspace (that is, a *Lattice*) of $\mathbb{R}^n$.



The LWE problem turns out to be an even more versatile basis for cryptography than discrete log and integer factoring and it has been used as a basis for a great many cryptographic schemes. It also has the advantage that, unlike factoring and discrete log, it is not known to be breakable even by *quantum* computers.

We now give a very rough sketch of how LWE can be used to obtain a fully homomorphic encryption scheme, following the paper.¹⁶ See Gentry's excellent survey¹⁵ for an accessible full description of this scheme. Gentry et al.'s¹⁶ scheme is the following candidate encryption: the secret key is some vector $s \in \{0, \dots, p-1\}^n$, and to encrypt the message $\lambda \in \{0, \dots, p-1\}$ we generate a random matrix A such that $As = \lambda s \pmod{p}$. Note this scheme is obviously homomorphic—if $As = \lambda s \pmod{p}$ and $A's = \lambda's \pmod{p}$ then $(A + A')s = (\lambda + \lambda')s \pmod{p}$ and $(AA')s = \lambda\lambda's \pmod{p}$. Unfortunately, it is also obviously insecure—using Gaussian elimination we can recover s from sufficiently many encryptions of zero. Gentry et al.¹⁶ fix this problem by adding *noise* to these encryptions, hence fooling the Gaussian elimination algorithm. Managing the noise so it does not blow up too much in the homomorphic operations requires delicate care and additional ideas, and this is the reason why Gentry called his survey “computing on the edge of Chaos.”

to the *bi-linear* pairing). An exciting approach toward this grand challenge was given by the work of Garg et al.¹² On a very high level, they showed how one can modify a fully homomorphic encryption scheme to support the $\stackrel{?}{=}$ operation by publishing some partial information on the secret decryption key, that at least as far as we know, only allows to check for plaintext-equality of ciphertexts without

revealing any additional information. The main challenge remaining is that the security of their scheme has yet to be proven (and in fact we have yet to even find the right *definitions* for security). I discuss this issue in more detail later. But even with this caveat their work is still a wonderful achievement, and provides cryptography with a candidate construction for one of the most versatile tools with which

one can achieve a great many cryptographic objectives.

From quasi-encryption to obfuscation.

As mentioned, the construction of obfuscation from a fully homomorphic quasi-encryption is rather complicated, but I will attempt to outline some of the ideas behind it. I will not even try to argue about the *security* of the obfuscation construction, but rather simply give some hints of how one might use the quasi-encryption scheme to represent a program P in a form that at least intuitively seems very hard to understand. At a high level the obfuscation of a program P consists simply of the quasi-encryptions (which we will call *encodings*) of N numbers $a_1, \dots, a_N$. To make this a valid representation, we must supply a way to compute $P(x)$ from these encodings for every input x . The idea is every input x would correspond to some formula f_x involving additions and multiplications modulo p such that $P(x) = 0$ if and only if $f_x(a_1, \dots, a_m) = 0 \pmod{p}$. Since we can test the latter condition using the $\oplus$, $\otimes$ and $\stackrel{?}{=}$ operations, we can find out if $P(x) = 0$ or $P(x) = 1$. This results in an obfuscation of programs with one bit of output, but can be generalized to handle programs with larger outputs.

How do we construct this magical mapping of inputs to formulas? We cannot present it fully here, but can describe some of the tools it uses. One component is the naive approach described earlier of constructing an obfuscation scheme from a fully homomorphic encryption. As we noted, this approach does not work because it only allows us to compute the output of the program in encrypted form, but it does essentially reduce the task of obfuscating an *arbitrary* function to the task of obfuscating the *decryption function* of the concrete encryption scheme. The crucial property for us is this decryption function can be computed via a *logarithmic depth circuit*. This allows us to use some of the insights obtained in the study of logarithmic depth circuits (which had been developed toward obtaining circuit lower bounds, without envisioning any cryptographic or other practical applications whatsoever). In particular, Barrington⁴ proved the following beautiful but seemingly

completely abstract result in 1986 (see also Figure 4):

THEOREM 1. *If $F : \{0, 1\}^n \rightarrow \{0, 1\}$ is a function computable by a log-depth circuit, then there exists a sequence of $m = \text{poly}(n)$ 5×5 matrices $A_1, \dots, A_m$ with entries in $\{0, 1\}$ and a mapping $x \mapsto \pi_x$ from $\{0, 1\}^n$ into the set of permutations of $[m']$ such that for every $x \in \{0, 1\}^n$*

$$F(x) = \left(\prod_{i=1}^{m'} A_{\pi_x(i)} \right)_{1,1}. \quad (3)$$

(That is, $F(x)$ is equal to the top left element of the product of matrices according to the order π_x .)

This already suggests the following method for obfuscation: If we want to obfuscate the decryption function F , then we construct the corresponding matrices $A_1, \dots, A_m$, encode all $N = 25m$ of their entries (which we will call $a_1, \dots, a_N$), and then define for every x the formula f_x to be the right-hand side of (3). This is a valid representation of the program P , since by using the homomorphic properties of the quasi-encryption we can compute from the N encodings of numbers the value of $F(x)$ (and hence, by combining this with our previous idea, also the value of $P(x)$) for every input x . However, it is not at all clear this representation does not leak additional information about the function. For example, how can we be sure we cannot recover the secret decryption key by multiplying the matrices in some different order?

Indeed, the actual obfuscation scheme of Garg et al.¹³ is more complicated and uses additional randomization tricks (as well as a more refined variant of quasi-encryption schemes called *graded encoding*) to protect against such attacks. Using these tricks, we were able to show in work with Garg et al.² (also see Brakerski and Rothblum⁸) that it is not possible to use the $\oplus$, $\otimes$ and $\stackrel{?}{=}$ operations to break the obfuscation. This still does not rule out the possibility of an attacker using the raw bits of the encoding (which is in fact what is used in the³ impossibility result) but it is a promising sign.

“Postmodern” Cryptography

So far I have avoided all discussion of the *security* of homomorphic

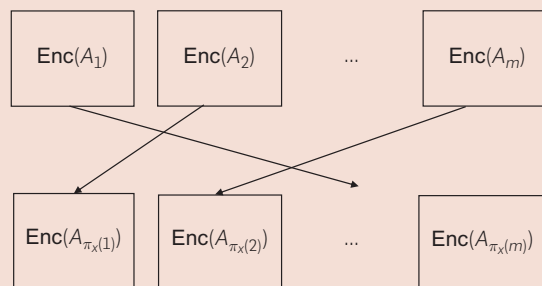
quasi-encryption schemes and obfuscation and indeed their status is significantly subtler than other cryptographic primitives such as encryption and digital signatures. To understand these issues, it is worthwhile to take a step back and look at the question of security for cryptographic schemes in general. The history of cryptography is littered with the figurative corpses of cryptosystems believed secure and then broken, and sometimes with the actual corpses of people (such as Mary, Queen of Scots) that have placed their faith in these cryptosystems. But something changed in modern times. In 1976 Diffie and Hellman¹¹ proposed the notion of *public key cryptography* and gave their famous Diffie–Hellman key exchange protocol, which was followed soon after by the RSA cryptosystem.²⁴ In contrast to cryptosystems such as Enigma, the description of these systems is simple and completely public. Moreover, by being *public key systems*, they give more information to potential attackers, and since they are widely deployed on a scale more massive than ever before, the incentives to break them are much higher. Indeed, it seems reasonable to estimate that the amount of manpower and computer cycles invested in cryptanalysis of these schemes today every year dwarves all the cryptanalytic efforts in pre-1970 human history. And yet (to our knowledge) they remain unbroken.

How can this be? I believe the answer lies in a fundamental shift from “security through obscurity” to “security through simplicity.” To understand

this consider the question of how could the relatively young and unknown Diffie and Hellman (and later Rivest, Shamir and Adleman) convince the world they have constructed a secure public key cryptosystem, an object so paradoxical that most people would have guessed could not exist (and indeed a concept so radical that Merkle’s first suggestion of it was rejected as an undergraduate project in a coding theory course). The traditional approach toward establishing something like that was “security through obscurity”—keep all details of the cryptosystem secret and have many people try to cryptanalyze it in-house, in the hope that any weakness would be discovered by them before it is discovered by your adversaries. But this approach was of course not available to Diffie and Hellman, working by themselves without many resources, and publishing in the open literature.

Of course the best way would have been to prove a mathematical theorem that breaking their system would necessarily take a huge number of operations. Thanks to the works of Church, Turing, and Gödel, we now know that this statement can in fact be phrased as a precise mathematical assertion. However, this assertion would in particular imply that $P \neq NP$ and hence proving it seems way beyond our current capabilities. Instead, what Diffie and Hellman did (aided by Ralph Merkle and John Gill) was to turn to “security by simplicity”—base their cryptosystem on a simple and well-studied mathematical problem, such as inverting modular exponentiation

Figure 4. We use Barrington’s Theorem to encode a program F computable by a logarithmic depth circuit into a sequence of m matrices $A_1, \dots, A_m$ and publish the quasi-encryptions of these matrices’ entries. Every input x corresponds to a permutation π_x such that if we multiply the matrices in this order, the top left element in the resulting matrix will equal $F(x)$.



or factoring integers, that has been investigated by mathematicians for ages for reasons having nothing to do with cryptography. More importantly, it is plausible to conjecture there simply *does not exist* an efficient algorithm to solve these clean well-studied problems, rather than it being the case that such an algorithm has not been found yet due to the problem's cumbersomeness and obscurity. Later papers, such as the pioneering works of Goldwasser and Micali,¹⁷ turned this into a standard paradigm and ushered in the age of *modern cryptography*, whereby we use precise *definitions* of security for our very intricate and versatile cryptosystems and then *reduce* the assertion that they satisfy these definitions into the conjectured hardness of a handful of very simple and well-known mathematical problems.

I wish I could say the new obfuscation schemes are in fact secure assuming integer factoring, computing discrete logarithm, or another well-studied problem (such as the LWE problem mentioned in the sidebar) is computationally intractable. Unfortunately, nothing like that is known. At the moment, our only arguments for the security of the constructions of the homomorphic quasi-encryption and indistinguishability obfuscator constructions is (as of this writing) we do not know how to break them. Since so many potential crypto applications rely on these schemes one could worry that we are entering (to use a somewhat inaccurate and overloaded term) a new age of “post-modern cryptography” where we still have precise definition of security, but need to assume an ever growing family of conjectures to prove that our schemes satisfy those definitions. Indeed, following the initial works of Garg et al.^{12,13} there have been several attacks on their schemes showing limitations on the security notions they satisfy (for example, see Coron et al.⁹ and Coron¹⁰) and it is not inconceivable that by the time this article appears they would be broken completely.

While this suggests the possibility all the edifices built on obfuscation and quasi-encryption could crumble

as a house of cards, the ideas behind these constructions seem too beautiful and profound for that to be the case. Once cryptographers have tasted the “promised land” of the great many applications enabled by IO, there is every hope that (as they have so many times before) they would rise to this challenge and manage to construct indistinguishability obfuscators and quasi-encryption based on a single well-studied conjecture, thus placing these objects firmly within the paradigm of modern cryptography. Indeed, this is the focus of an intensive research effort. More than that, one could hope by following the path these constructions lead us and “going boldly where no man has gone before,” we cryptographers will get new and fundamental insights on what is it that separates the easy computational problems from the hard ones.

Acknowledgments

Thanks to Dan Boneh, Craig Gentry, Omer Paneth, Amit Sahai, Brent Waters and the anonymous *Communications* reviewers for helpful comments on previous versions of this article. Thanks to Oded Regev for providing me with the figure for the “Learning with Errors” problem. 

References

1. Apon, D., Huang, Y., Katz, J., Malozemoff, A.J. Implementing cryptographic program obfuscation. Cryptology ePrint Archive, Report 2014/779, 2014. <http://eprint.iacr.org/>.
2. Barak, B., Garg, S., Kalai, Y.T., Paneth, O., Sahai, A. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014, pp. 221–238.
3. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K. On the (im) possibility of obfuscating programs. *J. ACM* 59, 2 (2012), 6. Preliminary version in *CRYPTO* 2001.
4. Barrington, D.A.M. Bounded-width polynomial-size branching programs recognize exactly those languages in NC¹. *J. Comput. Syst. Sci.* 38, 1 (1989), 150–164. Preliminary version in *STOC* 1986.
5. Boneh, D., Sahai, A., Waters, B. Functional encryption: A new vision for public-key cryptography. *Commun. ACM* 55, 11 (2012), 56–64.
6. Boneh, D., Silverberg, A. Applications of multilinear forms to cryptography. *Contemp. Math.* 324, 1 (2003), 71–90. Preliminary version posted on eprint on 2002, see <https://eprint.iacr.org/2002/080>.
7. Brakerski, Z., Rothblum, G.N. Obfuscating conjunctions. In *CRYPTO*, 2013, 416–434.
8. Brakerski, Z., Rothblum, G.N. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, 2014, 1–25.
9. Coron, J., Gentry, C., Halevi, S., Lepoint, T., Maji, H.K., Miles, E., Raykova, M., Sahai, A., Tibouchi, M. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *Proceedings of the Advances in Cryptology –*

CRYPTO 2015 – 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16–20, 2015, Part I, 2015, 247–266.

10. Coron, J.-S. Cryptanalysis of GGH15 multilinear maps. Cryptology ePrint Archive, Report 2015/1037, 2015. <http://eprint.iacr.org/>.
11. Diffie, W., Hellman, M.E. New directions in cryptography. *IEEE Trans. Inform. Theory* 22, 6 (1976), 644–654.
12. Garg, S., Gentry, C., Halevi, S. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013. See also *CRYPTOLOGY ePrint Archive*, Report 2012/610.
13. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013, 40–49.
14. Gentry, C. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009, 169–178.
15. Gentry, C. Computing on the edge of Chaos: Structure and randomness in encrypted computation. *Proceedings of the 2014 International Congress of Mathematicians (ICM)*, 2014. Also available online at <http://eprint.iacr.org/2014/610>.
16. Gentry, C., Sahai, A., Waters, B. Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013, 75–92.
17. Goldwasser, S., Micali, S. Probabilistic encryption. *J. Comput. Syst. Sci.* 28, 2 (1984), 270–299. Preliminary version in *STOC* 1982.
18. Green, M. Cryptographic obfuscation and ‘unhackable’ software, 2014. Blog post. Available at: <http://blog.cryptographyengineering.com/2014/02/cryptographic-obfuscation-and.html>.
19. Jacob, M., Boneh, D., Felten, E. Attacking an obfuscated cipher by injecting faults. In *Digital Rights Management*. Springer, 2003, 16–31.
20. Joux, A. A one round protocol for tripartite Diffie-Hellman. *J. Cryptol.* 17, 4 (2004), 263–276. Preliminary version in *ANTS* 2000.
21. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H. CryptDB: Processing queries on an encrypted database. *Commun. ACM* 55, 9 (2012), 103–111.
22. Regev, O. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM* 55, 6 (2009). Preliminary version in *STOC* 2005.
23. Rivest, R.L., Adleman, L., Dertouzos, M.L. On data banks and privacy homomorphisms. *Found. Secure Comput.* 4, 11 (1978), 169–180.
24. Rivest, R.L., Shamir, A., Adleman, L.M. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.

Boaz Barak (info@boazbarak.org) is the Gordon McKay Professor of Computer Science at the Harvard John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA. This article was written while he was at Microsoft Research.

Copyright held by author.
Publication rights licensed to ACM. \$15.00.



Watch the author discuss his work in this exclusive *Communications* video. <http://cacm.acm.org/videos/hopes-fears-and-software-obfuscation>

research highlights

P. 98

**Technical
Perspective
STACKing Up
Undefined Behaviors**

By John Regehr

P. 99

A Differential Approach to Undefined Behavior Detection

By Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek,
and Armando Solar-Lezama

P. 107

**Technical
Perspective
Taming the
Name Game**

By David Forsyth

P. 108

Learning to Name Objects

By Vicente Ordonez, Wei Liu, Jia Deng, Yejin Choi,
Alexander C. Berg, and Tamara L. Berg

Technical Perspective

STACKing Up Undefined Behaviors

By John Regehr

ANY COMPUTER SYSTEM must make trade-offs between freedoms retained by the system and guarantees made to the system's users. Designers attempt to balance conflicting goals, such as throughput and ease of use. Programming languages must make these trade-offs too. For example, a language with built-in garbage collection often retains the freedom to move objects around in memory, making it difficult to share objects with other processes or with hardware devices.

C and C++ are based on an extreme set of trade-offs: In these languages, a wide variety of hard-to-avoid program behaviors, such as signed integer overflow and out-of-bounds array references, are "undefined behaviors." No guarantees at all are made to a program that executes an undefined behavior. The languages' heavy reliance on undefined behaviors stems from C's obsolete philosophy of "trust the programmer" and also from pragmatic efforts by standards committees to encompass a wide variety of implementations. Bugs arising from undefined behaviors are difficult to prevent and during the last few decades they have led to a huge number of exploitable vulnerabilities in security-critical computer programs.

In 2009, a researcher found the Linux kernel contained code that dereferenced a pointer before checking if it was null. The C compiler was then able to effectively perform the following analysis:

Case 1: The pointer is not null, rendering the null check unnecessary.

Case 2: The pointer is null. Since this is undefined behavior, the compiler does not have any obligation to consider this case.

It is easy to see that neither case requires the null check, which the compiler failed to emit, resulting in an exploitable vulnerability in the kernel. This bug was considered to be pernicious since the source code contained the necessary null pointer check but the compiled binary code did not.

In the following paper, Wang et al. recognized this Linux bug was a member of a broader class of bugs. They hypothesized that any time the compiler is able to delete code by using reasoning based on undefined behavior, the program being compiled probably contains a bug. Their tool, STACK, detects this kind of "unstable code," and it has been used to find many bugs in important applications.

Although much effort had previously been put into detecting undefined behaviors, STACK's design point is interesting and new. First, a tool that warns about every instance of dead code is useless, because dead code is common and is often benign. STACK's differential approach to detecting unstable code permits it to focus on the special kind of code that is dead only because of undefined behavior. Empirically, C and C++ developers have a very difficult time finding unstable code by hand. Second, STACK makes no attempt to warn about undefined behaviors that do not lead to unstable code. While this may at first glance appear to be a limitation, in practice it means a large fraction of STACK's defect reports are useful to developers. Finally, STACK's model for undefined behavior is not tied to any particular compiler. Rather, STACK generates queries about undefined behavior that are passed to an automated theorem prover, enabling it to detect code that is unstable even if no C or C++ compiler can yet remove it.

The computing community is working hard to purge bugs arising from undefined behaviors from our huge installed base of C and C++. These languages were used to implement nearly all of our most safety-critical and security critical programs. More novel approaches, such as STACK, are needed. 

John Regehr (regehr@cs.utah.edu) is a professor in the School of Computing at the University of Utah, Salt Lake City.

Copyright held by author.

COMMUNICATIONS APPS

Access the
latest issue,
past issues,
BLOG@CACM,
News, and
more.



Available for iPad,
iPhone, and Android



Available for iOS,
Android, and Windows

[http://cacm.acm.org/
about-communications/
mobile-apps](http://cacm.acm.org/about-communications/mobile-apps)



Association for
Computing Machinery

A Differential Approach to Undefined Behavior Detection

By Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek, and Armando Solar-Lezama

Abstract

This paper studies undefined behavior arising in systems programming languages such as C/C++. Undefined behavior bugs lead to unpredictable and subtle systems behavior, and their effects can be further amplified by compiler optimizations. Undefined behavior bugs are present in many systems, including the Linux kernel and the Postgres database. The consequences range from incorrect functionality to missing security checks.

This paper proposes a formal and practical approach, which finds undefined behavior bugs by finding “unstable code” in terms of optimizations that leverage undefined behavior. Using this approach, we introduce a new static checker called STACK that precisely identifies undefined behavior bugs. Applying STACK to widely used systems has uncovered 161 new bugs that have been confirmed and fixed by developers.

1. INTRODUCTION

The specifications of many programming languages designate certain code fragments as having *undefined behavior* (Section 2.3 in Ref.¹⁸). For instance, in C “use of a nonportable or erroneous program construct or of erroneous data” leads to undefined behavior (Section 3.4.3 in Ref.²³); a comprehensive list of undefined behavior is available in the C language specification (Section J.2 in Ref.²³).

One category of undefined behavior is simply programming mistakes, such as buffer overflow and null pointer dereference. The other category is nonportable operations, the hardware implementations of which often have subtle differences. For example, when signed integer overflow or division by zero occurs, a division instruction traps on x86 (Section 3.2 in Ref.²²), while it silently produces an undefined result on PowerPC (Section 3.3.8 in Ref.³⁰). Another example is shift instructions: left-shifting a 32-bit one by 32 bits produces zero on ARM and PowerPC, but one on x86; however, left-shifting a 32-bit one by 64 bits produces zero on ARM, but one on x86 and PowerPC.

By designating certain programming mistakes and non-portable operations as having undefined behavior, the specifications give compilers the freedom to generate instructions that behave in arbitrary ways in those cases, allowing compilers to generate efficient and portable code without extra checks. For example, many higher-level programming languages (e.g., Java) have well-defined handling (e.g., runtime exceptions) on buffer overflow, and the compiler would need to insert extra bounds checks for memory access operations. However, the C/C++ compiler does *not* need to insert bounds checks, as out-of-bounds cases are undefined.

It is the programmer's responsibility to avoid undefined behavior.

According to the C/C++ specifications, programs that invoke undefined behavior can have arbitrary problems. As one summarized, “permissible undefined behavior ranges from ignoring the situation completely with unpredictable results, to having demons fly out of your nose.”⁴⁵ But what happens in practice? The rest of this paper will show that modern compilers increasingly exploit undefined behavior to perform aggressive optimizations; with these optimizations many programs can produce surprising results that programmers did not anticipate.

2. RISKS OF UNDEFINED BEHAVIOR

One risk of undefined behavior is that a program will observe different behavior on different hardware architectures, operating systems, or compilers. For example, a program that performs an oversized left-shift will observe different results on ARM and x86 processors. As another example, consider a simple SQL query:

```
SELECT ((-9223372036854775808)::int8)/(-1);
```

This query caused signed integer overflow in the Postgres database server, which on a 32-bit Windows system did not cause any problems, but on a 64-bit Windows system caused the server to crash, due to the different behavior of division instructions on the two systems.⁴⁴

In addition, compiler optimizations can amplify the effects of undefined behavior. For example, consider the pointer overflow check `buf + len < buf` shown in Figure 1, where `buf` is a pointer and `len` is a positive integer. The programmer's intention is to catch the case when `len` is so large that `buf + len` wraps around and bypasses the first check in Figure 1. We have found similar checks in a number of systems, including the Chromium browser, the Linux kernel, and the Python interpreter.⁴⁴

While this check appears to work with a flat address space, it fails on a segmented architecture (Section 6.3.2.3 in Ref.³²). Therefore, the C standard states that an overflowed pointer is undefined (Section 6.5.6 in Ref.^{23(p8)}), which allows gcc to simply assume that no pointer overflow ever occurs on *any* architecture. Under this assumption, `buf + len` must be larger than `buf`, and thus the “overflow” check always evaluates to

The original version of this paper is entitled “Towards Optimization-Safe Systems: Analyzing the Impact of Undefined Behavior” and was published in the *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP'13)*.⁴⁴

false. Consequently, gcc removes the check, paving the way for an attack to the system.¹⁷

As another example, Figure 2 shows a mild defect in the Linux kernel, where the programmer incorrectly placed the dereference `tun->sk` before the null pointer check `!tun`. Normally, the kernel forbids access to page zero; a null `tun` pointing to page zero causes a kernel oops at `tun->sk` and terminates the current process. Even if page zero is made accessible (e.g., via `mmap` or some other exploits^{24, 38}), the check `!tun` would catch a null `tun` and prevent any further exploits. In either case, an adversary should *not* be able to go beyond the null pointer check.

Unfortunately, when gcc first sees the dereference `tun->sk`, it concludes that the pointer `tun` must be non-null, because the C standard states that dereferencing a null pointer is undefined (Section 6.5.3 in Ref.²³). Since `tun` is non-null, gcc further determines that the null pointer check

Figure 1. A pointer overflow check found in several code bases. The code becomes vulnerable as gcc optimizes away the second `if` statement.¹⁷

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;
if (buf + len >= buf_end)
    return; /* len too large */
if (buf + len < buf)
    return; /* overflow, buf+len wrapped around */
/* write to buf[0..len-1] */
```

Figure 2. A null pointer dereference vulnerability (CVE-2009-1897) in the Linux kernel, where the dereference of pointer `tun` is before the null pointer check. The code becomes exploitable as gcc optimizes away the null pointer check.¹³

```
struct tun_struct *tun = ...;
struct sock *sk = tun->sk;
if (!tun)
    return POLLERR;
/* write to address based on tun */
```

is unnecessary and eliminates the check, making a privilege escalation exploit possible that would not otherwise be.¹³

To further understand how compiler optimizations exploit undefined behavior, we conduct a study using six real-world examples in the form of sanity checks, as shown in the top row of Figure 3. All of these checks may evaluate to *false* and become dead code under optimizations, because they invoke undefined behavior. We will use them to test existing compilers next.

- The check $p + 100 < p$ resembles Figure 1.
- The null pointer check `!p` with an earlier dereference resembles Figure 2.
- The check $x + 100 < x$ with a signed integer x caused a harsh debate in gcc's bugzilla.⁵
- The check $x^+ + 100 < 0$ tests whether optimizations perform more elaborate reasoning; x^+ is known to be positive.
- The shift check `!(1 << x)` was intended to catch a large shifting amount x , from a patch to the `ext4` file system.⁶
- The check `abs(x) < 0`, intended to catch the most negative value (i.e., -2^{n-1}), tests whether optimizations understand library functions.⁷

We chose 12 well-known C/C++ compilers to see what they do with the above code examples: 2 open-source compilers (gcc and clang) and 10 recent commercial compilers (HP's aCC, ARM's armcc, Intel's icc, Microsoft's msvc, AMD's open64, PathScale's pathcc, Oracle's suncc, TI's TMS320C6000, Wind River's Diab compiler, and IBM's XL C compiler). For every code example, we test whether a compiler optimizes the check into *false*, and if so, we find the lowest optimization level $-On$ at which it happens. The result is shown in Figure 3.

We further use gcc and clang to study the evolution of optimizations, as the history is easily accessible. For gcc, we chose the following representative versions that span more than a decade:

Figure 3. Optimizations of unstable code in popular compilers. This includes gcc, clang, aCC, armcc, icc, msvc, open64, pathcc, suncc, TI's TMS320C6000, Wind River's Diab compiler, and IBM's XL C compiler. In the examples, p is a pointer, x is a signed integer, and x^+ is a positive signed integer. In each cell, " On " means that the specific version of the compiler optimizes the check into *false* and discards it at optimization level n , while " $-$ " means that the compiler does not discard the check at any level.

	<code>if (p + 100 < p)</code>	<code>*p; if (!p)</code>	<code>if (x + 100 < x)</code>	<code>if (x⁺ + 100 < 0)</code>	<code>if (!(1 << x))</code>	<code>if (abs(x) < 0)</code>
gcc-2.95.3	-	-	01	-	-	-
gcc-3.4.6	-	02	01	-	-	-
gcc-4.2.1	00	-	02	-	-	02
gcc-4.9.1	02	02	02	02	-	02
clang-1.0	01	-	-	-	-	-
clang-3.4	01	-	01	-	01	-
aCC-6.25	-	-	-	-	-	03
armcc-5.02	-	-	02	-	-	-
icc-14.0.0	-	02	01	02	-	-
msvc-11.0	-	01	-	-	-	-
open64-4.5.2	01	-	02	-	-	02
pathcc-1.0.0	01	-	02	-	-	02
suncc-5.12	-	03	-	-	-	-
ti-7.4.2	00	-	00	02	-	-
windriver-5.9.2	-	-	00	-	-	-
xlC-12.1	03	-	-	-	-	-

- gcc 2.95.3, the last 2.x, released in 2001;
- gcc 3.4.6, the last 3.x, released in 2006;
- gcc 4.2.1, the last GPLv2 version, released in 2007 and still widely used in BSD systems;
- gcc 4.9.1, released in 2014.

For comparison, we chose two versions of clang, 1.0 released in 2009, and 3.4 released in 2014.

We can see that exploiting undefined behavior to eliminate code is common among compilers, not just in recent gcc versions as some programmers have claimed.²⁶ Even gcc 2.95.3 eliminates $x + 100 < x$. Some compilers eliminate code that gcc does not (e.g., clang on $1 < x$).

These optimizations can lead to baffling results even for veteran C programmers, because code unrelated to the undefined behavior gets optimized away or transformed in unexpected ways. Such bugs lead to spirited debates between compiler developers and practitioners that use the C language but do not adhere to the letter of the official C specification. Practitioners describe these optimizations as “make no sense”⁴⁰ and merely the compiler’s “creative reinterpretation of basic C semantics.”²⁶ On the other hand, compiler writers argue that the optimizations are legal under the specification; it is the “broken code”⁵ that programmers should fix. Worse yet, as compilers evolve, new optimizations are introduced that may break code that used to work before; as we show in Figure 3, many compilers have become more aggressive over the past 20 years with such optimizations.

3. CHALLENGES OF UNDEFINED BEHAVIOR DETECTION

Given the wide range of problems that undefined behavior can cause, what should programmers do about it? The naïve approach is to require programmers to carefully read and understand the C language specification, so that they can write careful code that avoids invoking undefined behavior. Unfortunately, as we demonstrate in Section 2, even experienced C programmers do not fully understand the intricacies of the C language, and it is exceedingly difficult to avoid invoking undefined behavior in practice.

Since optimizations often amplify the problems due to undefined behavior, some programmers (such as the Postgres developers) have tried reducing the compiler’s optimization level, so that aggressive optimizations do not take advantage of undefined behavior bugs in their code. As we see in Figure 3, compilers are inconsistent about the optimization levels at which they take advantage of undefined behavior, and several compilers make undefined behavior optimizations even at optimization level zero (which should, in principle, disable all optimizations).

Runtime checks can be used to detect certain undefined behaviors at runtime; for example, gcc provides an `-ftrapv` option to trap on signed integer overflow, and clang provides an `-fsanitize=undefined` option to trap several more undefined behaviors. There have also been attempts at providing a more “programmer-friendly” refinement of C,^{14, 29} which has less undefined behavior, though in general it remains unclear how to outlaw undefined behavior from

the specification without incurring significant performance overhead.^{14, 42}

Certain static-analysis and model checkers identify classes of bugs due to undefined behavior. For example, compilers can catch some obvious cases (e.g., using gcc’s `-Wall`), but in general this is challenging (Part 3 in Ref.²⁷); tools that find buffer overflow bugs¹¹ can be viewed as finding undefined behavior bugs, because referencing a location outside of a buffer’s range is undefined behavior. See Section 6 for a more detailed discussion of related work.

4. APPROACH: FINDING DIVERGENT BEHAVIOR

Ideally, compilers would generate warnings for developers when an application invokes undefined behavior, and this paper takes a static analysis approach to finding undefined behavior bugs. This boils down to deciding, for each operation in the program, whether it can be invoked with arguments that lead to undefined behavior. Since many operations in C can invoke undefined behavior (e.g., signed integer operations, pointer arithmetic), producing a warning for every operation would overwhelm the developer, so it is important for the analysis to be precise. Global reasoning can precisely determine what values an argument to each operation can take, but it does not scale to large programs.

Instead of performing global reasoning, our goal is to find local invariants (or likely invariants) on arguments to a given operation. We are willing to be incomplete: if there are not enough local invariants, we are willing to not report potential problems. On the other hand, we would like to ensure that every report is likely to be a real problem.¹

The local likely invariant that we exploit in this paper has to do with unnecessary source code written by programmers. By “unnecessary source code” we mean dead code, unnecessarily complex expressions that can be transformed into a simpler form, etc. We expect that all of the source code that programmers write should either be necessary code, or it should be clearly unnecessary; that is, it should be clear from local context that the code is unnecessary, without relying on subtle semantics of the C language. For example, the programmer might write `if (0) { ... }`, which is clearly unnecessary code. However, our likely invariant tells us that programmers would never write code like `a = b << c; if (c >= 32) { ... }`, where `b` is a 32-bit integer. The `if` statement in this code snippet is unnecessary code, because `c` could never be 32 or greater due to undefined behavior in the preceding left-shift. The core of our invariant is that programmers are unlikely to write such subtly unnecessary code.

To formalize this invariant, we need to distinguish “live code” (code that is always necessary), “dead code” (code that is always unnecessary), and “unstable code” (code that is subtly unnecessary). We do this by considering the different possible interpretations that the programmer might have for the C language specification. In particular, we consider `C` to be the language’s official specification, and `C'` to be the specification that the programmer believes `C` has. For the purposes of this paper, `C'` differs from `C` in which operations lead to undefined behavior. For example, a programmer might expect shifts to be well-defined for all possible arguments; this is one such possible `C'`. In other words, `C'` is

a relaxed version of the official C, by assigning certain interpretations to operations that have undefined behavior in C.

Using the notion of different language specifications, we say that a piece of code is *live* if, for every possible C' , the code is necessary. Conversely, a piece of code is *dead* if, for every possible C' , the code is unnecessary; this captures code like `if (0) { ... }`. Finally, a piece of code is *unstable* if, for some C' variants, it is unnecessary, but in other C' variants, it is necessary. This means that two programmers that do not precisely understand the details of the C specification might disagree about what the code is doing. As we demonstrate in the rest of this paper, this heuristic often indicates the presence of a bug.

Building on this invariant, we can now detect when a program is likely invoking undefined behavior. In particular, given an operation o in a function f , we compute the set of unnecessary code in f under different interpretations of undefined behavior at o . If the set of unnecessary code is the same for all possible interpretations, we cannot say anything about whether o is likely to invoke undefined behavior. However, if the set of unnecessary code varies depending on what undefined behavior o triggers, this means that the programmer wrote unstable code. However, by our assumption, this should never happen, and we conclude that the programmer was likely thinking they're writing live code, and simply did not realize that o would trigger undefined behavior for the *same* set of inputs that are required for the code to be live.

5. THE STACK TOOL

To find undefined behavior bugs using the above approach, we built a static analysis tool called STACK. In practice, it is difficult to enumerate and consider all possible C' variants. Thus, to build a practical tool, we pick a single variant, called C^* . C^* defines a null pointer that maps to address zero, and wrap-around semantics for pointer and integer arithmetic.³¹ We believe this captures the common semantics that programmers (mistakenly) believe C provides. Although our C^* deals with only a subset of undefined behaviors in the C specification, a different C^* could capture other semantics that programmers might implicitly assume, or handle undefined behavior for other operations that our C^* does not address.

STACK relies on an optimizer $\mathcal{O}$ to implicitly flag unnecessary code. STACK's $\mathcal{O}$ eliminates dead code and performs expression simplifications under the semantics of C and C^* , respectively. For code fragment e , if $\mathcal{O}$ is *not* able to rewrite e under neither semantics, STACK considers e as “live code”; if $\mathcal{O}$ is able to rewrite e under both semantics, e is “dead code”; if $\mathcal{O}$ is able to rewrite e under C but not C^* , STACK reports it as “unstable code.”

Since STACK uses just two interpretations of the language specification (namely, C and C^*), it might miss bugs that could arise under different interpretations. For instance, any code eliminated by $\mathcal{O}$ under C^* would never trigger a warning from STACK, even if there might exist another C' which would not allow eliminating that code. STACK's approach could be extended to support multiple interpretations to address this potential shortcoming.

5.1. A definition of unstable code

We now give a formal definition of unstable code. A code fragment e is a statement or expression at a particular source location in program $\mathcal{P}$. If the compiler can transform the fragment e in a way that would change $\mathcal{P}$'s behavior under C^* but not under C, then e is unstable code.

Let $\mathcal{P}[e/e']$ be a program formed by replacing e with some fragment e' at the same source location. When is it legal for a compiler to transform $\mathcal{P}$ into $\mathcal{P}[e/e']$, denoted as $\mathcal{P} \rightsquigarrow \mathcal{P}[e/e']$? In a language specification without undefined behavior, the answer is straightforward: it is legal if for every input, both $\mathcal{P}$ and $\mathcal{P}[e/e']$ produce the same result. In a language specification *with* undefined behavior, the answer is more complicated; namely, it is legal if for every input, one of the following is true:

- both $\mathcal{P}$ and $\mathcal{P}[e/e']$ produce the same results without invoking undefined behavior, or
- $\mathcal{P}$ invokes undefined behavior, in which case it does not matter what $\mathcal{P}[e/e']$ does.

Using this notation, we define unstable code below.

DEFINITION 1 (UNSTABLE CODE). *A code fragment e in program $\mathcal{P}$ is unstable w.r.t. language specifications C and C^* iff there exists a fragment e' such that $\mathcal{P} \rightsquigarrow \mathcal{P}[e/e']$ is legal under C but not under C^* .*

For example, for the sanity checks listed in Figure 3, a C compiler is entitled to replace them with *false*, as this is legal according to the C specification, whereas a hypothetical C^* compiler cannot do the same. Therefore, these checks are unstable code.

5.2. Algorithms for identifying unstable code

The above definition captures what unstable code is, but does not provide a way of finding unstable code, because it is difficult to reason about how an entire program will behave. As a proxy for a change in program behavior, STACK looks for code that can be transformed by some optimizer $\mathcal{O}$ under C but not under C^* . In particular, STACK does this using a two-phase scheme:

1. run $\mathcal{O}$ without taking advantage of undefined behavior, which captures optimizations under C^* ; and
2. run $\mathcal{O}$ again, this time taking advantage of undefined behavior, which captures (more aggressive) optimizations under C.

If $\mathcal{O}$ optimizes extra code in the second phase, we assume the reason $\mathcal{O}$ did not do so in the first phase is because it would have changed the program's semantics under C^* , and so STACK considers that code to be unstable.

STACK's optimizer-based approach to finding unstable code will miss unstable code that a specific optimizer cannot eliminate in the second phase, even if there exists some optimizer that could. This approach will also generate false reports if the optimizer is not aggressive enough in eliminating code in the first phase. Thus, one challenge in STACK's

design is coming up with an optimizer that is sufficiently aggressive to minimize these problems.

In order for this approach to work, **STACK** requires an optimizer that can selectively take advantage of undefined behavior. To build such optimizers, we formalize what it means to “take advantage of undefined behavior” in Section 5.2.1, by introducing the *well-defined program assumption*, which captures C’s assumption that programmers never write programs that invoke undefined behavior. Given an optimizer that can take explicit assumptions as input, **STACK** can turn on (or off) optimizations based on undefined behavior by supplying (or not) the well-defined program assumption to the optimizer. We build two aggressive optimizers that follow this approach: one that eliminates unreachable code (Section 5.2.2) and one that simplifies unnecessary computation (Section 5.2.3).

Well-defined program assumption. We formalize what it means to take advantage of undefined behavior in an optimizer as follows. Consider a program with input $\mathbf{x}$. Given a code fragment e , let $R_e(\mathbf{x})$ denote its *reachability condition*, which is *true* iff e will execute under input $\mathbf{x}$; and let $U_e(\mathbf{x})$ denote its *undefined behavior condition*, or *UB condition* for short, which indicates whether e exhibits undefined behavior on input $\mathbf{x}$, as summarized in Figure 4.

Both $R_e(\mathbf{x})$ and $U_e(\mathbf{x})$ are boolean expressions. For example, given a pointer dereference $*p$ in expression e , one UB condition $U_e(\mathbf{x})$ is $p = \text{NULL}$ (i.e., causing a null pointer dereference).

Intuitively, in a well-defined program to dereference pointer p , p must be non-null. In other words, the negation of its UB condition, $p \neq \text{NULL}$, must hold whenever the expression executes. We generalize this below.

DEFINITION 2 (WELL-DEFINED PROGRAM ASSUMPTION). A code fragment e is well-defined on an input $\mathbf{x}$ iff executing e never triggers undefined behavior at e :

$$R_e(\mathbf{x}) \rightarrow \neg U_e(\mathbf{x}). \quad (1)$$

Furthermore, a program is well-defined on an input iff every fragment of the program is well-defined on that input, denoted as Δ :

$$\Delta(\mathbf{x}) = \bigwedge_{e \in \mathcal{P}} R_e(\mathbf{x}) \rightarrow \neg U_e(\mathbf{x}). \quad (2)$$

Eliminating unreachable code. The first algorithm identifies unstable statements that can be eliminated (i.e., $\mathcal{P} \rightsquigarrow \mathcal{P}[e/\emptyset]$ where e is a statement). For example, if reaching a statement requires triggering undefined behavior, then that statement must be unreachable. We formalize this below.

THEOREM 1 (ELIMINATION). In a well-defined program $\mathcal{P}$, an optimizer can eliminate code fragment e , if there is no input $\mathbf{x}$ that both reaches e and satisfies the well-defined program assumption $\Delta(\mathbf{x})$:

$$\nexists \mathbf{x} : R_e(\mathbf{x}) \wedge \Delta(\mathbf{x}). \quad (3)$$

The boolean expression $R_e(\mathbf{x}) \wedge \Delta(\mathbf{x})$ is referred as the elimination query.

PROOF. Assuming $\Delta(\mathbf{x})$ is *true*, if the elimination query $R_e(\mathbf{x}) \wedge \Delta(\mathbf{x})$ always evaluates to *false*, then $R_e(\mathbf{x})$ must be *false*, meaning that e must be unreachable. One can then safely eliminate e . $\square$

Consider Figure 2 as an example. There is one input `tun` in this program. To pass the earlier `if` check, the reachability condition of the `return` statement is `!tun`. There is one UB condition `tun = NULL`, from the pointer dereference `tun->sk`, the reachability condition of which is *true*. As a result, the elimination query $R_e(\mathbf{x}) \wedge \Delta(\mathbf{x})$ for the `return` statement is:

$$!tun \wedge (\text{true} \rightarrow \neg(tun = \text{NULL})).$$

Clearly, there is no `tun` that satisfies this query. Therefore, one can eliminate the `return` statement.

With the above definition it is easy to construct an algorithm to identify unstable code due to code elimination (see Figure 5). The algorithm first removes unreachable fragments without the well-defined program assumption, and then warns against fragments that become unreachable with this assumption. The latter are unstable code.

Figure 4. Examples of C/C++ code fragments and their undefined behavior conditions. We describe their sufficient (though not necessary) conditions under which the code is undefined (Section J.2 in Ref.²³). Here p, p', q are n -bit pointers; x, y are n -bit integers; a is an array, the capacity of which is denoted as `ARRAY_SIZE(a)`; op_s refers to binary operators $+, -, *, /$ over signed integers; x_∞ means to consider x as infinitely ranged; `NULL` is the null pointer; `alias(p, q)` predicates whether p and q point to the same object.

Code fragment	Sufficient condition	Undefined behavior
Core language:		
$p + x$	$p_\infty + x_\infty \notin [0, 2^n - 1]$	Pointer overflow
$*p$	$p = \text{NULL}$	Null pointer dereference
$x \text{ op}_s y$	$x_\infty \text{ op}_s y_\infty \notin [-2^{n-1}, 2^{n-1} - 1]$	Signed integer overflow
$x / y, x \% y$	$y = 0$	Division by zero
$x \ll y, x \gg y$	$y < 0 \vee y \geq n$	Oversized shift
$a[x]$	$x < 0 \vee x \geq \text{ARRAY_SIZE}(a)$	Buffer overflow
Standard library:		
<code>abs(x)</code>	$x = -2^{n-1}$	Absolute value overflow
<code>memcpy(dst, src, len)</code>	$ \text{dst} - \text{src} < \text{len}$	Overlapping memory copy
use q after <code>free(p)</code>	<code>alias(p, q)</code>	Use after free
use q after $p' := \text{realloc}(p, \dots)$	<code>alias(p, q) $\wedge$ $p' \neq \text{NULL}$</code>	Use after realloc

Simplifying unnecessary computation. The second algorithm identifies unstable expressions that can be optimized into a simpler form (i.e., $\mathcal{P} \rightsquigarrow \mathcal{P}[e/e']$ where e and e' are expressions). For example, if evaluating a boolean expression to *true* requires triggering undefined behavior, then that expression must evaluate to *false*. We formalize this below.

THEOREM 2 (SIMPLIFICATION). *In a well-defined program $\mathcal{P}$, an optimizer can simplify expression e with another e' , if there is no input $\mathbf{x}$ that evaluates $e(\mathbf{x})$ and $e'(\mathbf{x})$ to different values, while both reaching e and satisfying the well-defined program assumption $\Delta(\mathbf{x})$:*

$$\exists e' \nexists \mathbf{x} : e(\mathbf{x}) \neq e'(\mathbf{x}) \wedge R_e(\mathbf{x}) \wedge \Delta(\mathbf{x}). \quad (4)$$

The boolean expression $e(\mathbf{x}) \neq e'(\mathbf{x}) \wedge R_e(\mathbf{x}) \wedge \Delta(\mathbf{x})$ is referred as the simplification query.

PROOF. Assuming $\Delta(\mathbf{x})$ is *true*, if the simplification query $e(\mathbf{x}) \neq e'(\mathbf{x}) \wedge R_e(\mathbf{x}) \wedge \Delta(\mathbf{x})$ always evaluates to *false*, then either $e(\mathbf{x}) = e'(\mathbf{x})$, meaning that they evaluate to the same value; or $R_e(\mathbf{x})$ is *false*, meaning that e is unreachable. In either case, one can safely replace e with e' . $\square$

Simplification relies on an oracle to propose e' for a given expression e . Note that there is no restriction on the proposed expression e' . In practice, it should be simpler than the original e since compilers tend to simplify code. STACK currently implements two oracles:

- Boolean oracle: propose *true* and *false* in turn for a boolean expression, enumerating possible values.
- Algebra oracle: propose to eliminate common terms on both sides of a comparison if one side is a subexpression of the other. It is useful for simplifying nonconstant expressions, such as proposing $y < 0$ for $x + y < x$, by eliminating x from both sides.

As an example, consider simplifying $p + 100 < p$ using the boolean oracle, where p is a pointer. For simplicity assume its reachability condition is *true*. From Figure 4, the UB condition of $p + 100$ is $p_\infty + 100_\infty \notin [0, 2^n - 1]$. The boolean oracle first proposes *true*. The corresponding simplification query is:

$$(p + 100 < p) \neq \text{true} \\ \wedge \text{true} \wedge (\text{true} \rightarrow \neg(p_\infty + 100_\infty \notin [0, 2^n - 1])).$$

Clearly, this is satisfiable. The boolean oracle then proposes

Figure 5. The elimination algorithm. It reports unstable code that becomes unreachable with the well-defined program assumption.

```

1: procedure ELIMINATE( $\mathcal{P}$ )
2:   for all  $e \in \mathcal{P}$  do
3:     if  $R_e(\mathbf{x})$  is UNSAT then
4:       REMOVE( $e$ ) ▷ trivially unreachable
5:     else
6:       if  $R_e(\mathbf{x}) \wedge \Delta(\mathbf{x})$  is UNSAT then
7:         REPORT( $e$ )
8:         REMOVE( $e$ ) ▷ unstable code eliminated
```

false. This time the simplification query is:

$$(p + 100 < p) \neq \text{false} \\ \wedge \text{true} \wedge (\text{true} \rightarrow \neg(p_\infty + 100_\infty \notin [0, 2^n - 1])).$$

Since there is no pointer p that satisfies this query, one can fold $p + 100 < p$ into *false*.

With the above definition it is straightforward to construct an algorithm to identify unstable code due to simplification (see Figure 6). The algorithm consults an oracle for every possible simpler form e' for expression e . Similar to elimination, it warns if it finds e' that is equivalent to e only with the well-defined program assumption.

5.3. Implementation

We implemented STACK using the LLVM compiler framework²⁸ and the Boolector solver.⁴ STACK consists of approximately 4000 lines of C++ code. To make the tool scale to large code bases, STACK implements an approximate version of the algorithms described in Section 5.2. Interested readers can refer to our SOSP paper for details.⁴⁴

STACK focuses on identifying unstable code by exploring two basic optimizations, elimination because of unreachability and simplification because of unnecessary computation. It is possible to exploit the well-defined program assumption in other forms. For example, instead of discarding code, some optimizations reorder instructions and produce unwanted code due to memory aliasing⁴¹ or data races,³ which STACK does not implement.

STACK implements two oracles, boolean and algebra, for proposing new expressions for simplification. One can extend it by introducing new oracles.

5.4. Main results

From July 2012 to March 2013, we periodically applied STACK to systems software written in C/C++, including OS kernels, virtual machines, databases, multimedia encoders/decoders, language runtimes, and security libraries. Based on STACK's bug reports, we submitted patches to the corresponding developers. The developers confirmed and fixed 161 new bugs.

We also applied STACK to all 17,432 packages in the Debian Wheezy archive as of March 24, 2013. STACK checked 8575 of them that contained C/C++ code. Building and analyzing these packages took approximately 150 CPU-days on

Figure 6. The simplification algorithm. It asks an oracle to propose a set of possible e' , and reports if any of them is equivalent to e with the well-defined program assumption.

```

1: procedure SIMPLIFY( $\mathcal{P}$ , oracle)
2:   for all  $e \in \mathcal{P}$  do
3:     for all  $e' \in \text{PROPOSE}(\text{oracle}, e)$  do
4:       if  $e(\mathbf{x}) \neq e'(\mathbf{x}) \wedge R_e(\mathbf{x})$  is UNSAT then
5:         REPLACE( $e, e'$ )
6:         break ▷ trivially simplified
7:       if  $e(\mathbf{x}) \neq e'(\mathbf{x}) \wedge R_e(\mathbf{x}) \wedge \Delta(\mathbf{x})$  is UNSAT then
8:         REPORT( $e$ )
9:         REPLACE( $e, e'$ )
10:        break ▷ unstable code simplified
```

Intel Xeon E7-8870 2.4 GHz processors. For 3471 (40%) out of these 8575 packages, STACK issued at least one warning.

The results show that undefined behavior is widespread, and that STACK is useful for identifying undefined behavior. Please see our paper for more complete details.⁴⁴

6. RELATED WORK

To the best of our knowledge, we present the first definition and static checker to find unstable code, but we build on several pieces of related work. In particular, earlier surveys^{25, 35, 42} and blog posts^{27, 33, 34} collect examples of unstable code, which motivated us to tackle this problem. We were also motivated by related techniques that can help with addressing unstable code, which we discuss next.

6.1. Testing strategies

Our experience with unstable code shows that in practice it is difficult for programmers to notice certain critical code fragments disappearing from the running system as they are silently discarded by the compiler. Maintaining a comprehensive test suite may help catch “vanished” code in such cases, though doing so often requires a substantial effort to achieve high code coverage through manual test cases. Programmers may also need to prepare a variety of testing environments as unstable code can be hardware- and compiler-dependent.

Automated tools such as KLEE⁹ can generate test cases with high coverage using symbolic execution. These tools, however, often fail to model undefined behavior correctly. Thus, they may interpret the program differently from the language standard and miss bugs. Consider a check $x + 100 < x$, where x is a signed integer. KLEE considers $x + 100$ to wrap around given a large x ; in other words, the check catches a large x when executing in KLEE, even though gcc discards the check. Therefore, to detect unstable code, these tools need to be augmented with a model of undefined behavior, such as the one we proposed in this paper.

6.2. Optimization strategies

We believe that programmers should avoid undefined behavior. However, overly aggressive compiler optimizations are also responsible for triggering these bugs. Traditionally, compilers focused on producing fast and small code, even at the price of sacrificing security, as shown in Section 2. Compiler writers should rethink optimization strategies for generating secure code.

Consider $x + 100 < x$ with a signed integer x again. The language standard does allow compilers to consider the check to be *false* and discard it. In our experience, however, it is unlikely that the programmer intended the code to be removed. A programmer-friendly compiler could instead generate efficient overflow checking code, for example, by exploiting the overflow flag available on many processors after evaluating $x + 100$. This strategy, also allowed by the language standard, produces more secure code than discarding the check. Alternatively, the compiler could produce warnings when exploiting undefined behavior in a potentially surprising way.⁸

Currently, gcc provides several options to alter the compiler’s assumptions about undefined behavior, such as

- `-fwrapv`, assuming signed integer wraparound for addition, subtraction, and multiplication;
- `-fno-strict-overflow`, assuming pointer arithmetic wraparound in addition to `-fwrapv`; and
- `-fno-delete-null-pointer-checks`,³⁷ assuming unsafe null pointer dereferences.

These options can help reduce surprising optimizations, at the price of generating slower code. However, they cover an incomplete set of undefined behavior that may cause unstable code (e.g., no options for shift or division). Another downside is that these options are specific to gcc; other compilers may not support them or interpret them in a different way.⁴²

6.3. Checkers

Many existing tools can detect undefined behavior as listed in Figure 4. For example, gcc provides the `-ftrapv` option to insert runtime checks for signed integer overflows (Section 3.18 in Ref.³⁶); IOC¹⁵ (now part of clang’s sanitizers¹²) and KINT⁴³ cover a more complete set of integer errors; Saturn¹⁶ finds null pointer dereferences; several dedicated C interpreters such as kcc¹⁹ and Frama-C¹⁰ perform checks for undefined behavior. See Chen et al.’s survey¹¹ for a summary.

In complement to these checkers that directly target undefined behavior, STACK finds unstable code that becomes dead due to undefined behavior. In this sense, STACK can be considered as a generalization of Engler et al.’s inconsistency cross-checking framework.^{16, 20} STACK, however, supports more expressive assumptions, such as pointer and integer operations.

As explored by existing checkers,^{2, 21, 39} dead code is an effective indicator of likely bugs. STACK finds undefined behavior bugs by finding *subtly* unnecessary code under different interpretations of the language specification.

6.4. Language design

Language designers may reconsider whether it is necessary to declare certain constructs as undefined behavior, since reducing undefined behavior in the specification is likely to avoid unstable code. One example is left-shifting a signed 32-bit one by 31 bits. This is undefined behavior in C (Section 6.5.7 in Ref.²³), even though the result is consistently `0x80000000` on most modern processors. The committee for the C++ language standard has assigned well-defined semantics to this operation in the latest specification.²⁹

7. SUMMARY

This paper demonstrates that undefined behavior bugs are much more prevalent than was previously believed, that they lead to a wide range of significant problems, that they are often misunderstood by system programmers, and that many popular compilers already perform unexpected optimizations, leading to misbehaving or vulnerable systems. We introduced a new approach for identifying undefined behavior, and developed a static checker, STACK, to help system programmers identify and fix bugs. We hope that compiler writers will also rethink optimization strategies against undefined behavior. Finally, we hope this paper

encourages language designers to be careful with using undefined behavior in the language specification. Almost every language allows a developer to write programs that have undefined meaning according to the language specification. This research indicates that being liberal with what is undefined can lead to subtle bugs. All of STACK's source code is publicly available at <http://css.csail.mit.edu/stack/>.

Acknowledgments

We thank Xavier Leroy for helping improve this paper, and many others for their feedback on earlier papers.^{42, 44} This research was supported by the DARPA Clean-slate design of Resilient, Adaptive, Secure Hosts (CRASH) program under contract \#N66001-10-2-4089, and by NSF award CNS-1053143. 

References

- Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., Engler, D. A few billion lines of code later: Using static analysis to find bugs in the real world. *Commun. ACM* 53, 2 (Feb. 2010), 66–75.
- Blackshear, S., Lahiri, S. Almost-correct specifications: A modular semantic framework for assigning confidence to warnings. In *Proceedings of the 2013 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* (Seattle, WA, Jun. 2013), 209–218.
- Boehm, H.-J. Threads cannot be implemented as a library. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* (Chicago, IL, Jun. 2005), 261–268.
- Brummayer, R., Biere, A. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (York, UK, Mar. 2009), 174–177.
- Bug 30475 – `assert(int+100 > int)` optimized away, 2007. http://gcc.gnu.org/bugzilla/show_bug.cgi?id=30475.
- Bug 14287 – `ext4: fixpoint divide exception at ext4_fill_super`, 2009. https://bugzilla.kernel.org/show_bug.cgi?id=14287.
- Bug 49820 – explicit check for integer negative after `abs` optimized away, 2011. http://gcc.gnu.org/bugzilla/show_bug.cgi?id=49820.
- Bug 53265 – warn when undefined behavior implies smaller iteration count, 2013. http://gcc.gnu.org/bugzilla/show_bug.cgi?id=53265.
- Cadar, C., Dunbar, D., Engler, D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation (OSDI)* (San Diego, CA, Dec. 2008).
- Canet, G., Cuoq, P., Monate, B. A value analysis for C programs. In *Proceedings of the 9th IEEE International Working Conference on Source Code Analysis and Manipulation* (Edmonton, Canada, Sept. 2009), 123–124.
- Chen, H., Mao, Y., Wang, X., Zhou, D., Zeldovich, N., Kaashoek, M.F. Linux kernel vulnerabilities: State-of-the-art defenses and open problems. In *Proceedings of the 2nd Asia-Pacific Workshop on Systems* (Shanghai, China, Jul. 2011).
- Clang Compiler User's Manual: Controlling Code Generation*, 2014. <http://clang.llvm.org/docs/UsersManual.html#controlling-code-generation>.
- Corbet, J. Fun with NULL pointers, part 1, July 2009. <http://lwn.net/Articles/342330/>.
- Cuoq, P., Flatt, M., Regehr, J. Proposal for a friendly dialect of C, Aug. 2014. <http://blog.regehr.org/archives/1180>.
- Dietz, W., Li, P., Regehr, J., Adve, V. Understanding integer overflow in C/C++. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)* (Zurich, Switzerland, Jun. 2012), 760–770.
- Dillig, I., Dillig, T., Aiken, A. Static error detection using semantic inconsistency inference. In *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* (San Diego, CA, Jun. 2007), 435–445.
- Dougherty, C.R., Seacord, R.C. C compilers may silently discard some wraparound checks. Vulnerability note VU#162289, US-CERT, 2008. <http://www.kb.cert.org/vuls/id/162289>, original version: <http://www.isps.org/render.html?it=9100>, also known as CVE-2008-1685.
- Ellison, C., Roşu, G. *Defining the Undefinedness of C*. Technical report, University of Illinois, Apr. 2012. <http://hdl.handle.net/2142/30780>.
- Ellison, C., Roşu, G. An executable formal semantics of C with applications. In *Proceedings of the 39th ACM Symposium on Principles of Programming Languages (POPL)* (Philadelphia, PA, Jan. 2012), 533–544.
- Engler, D., Chen, D.Y., Hallem, S., Chou, A., Chelf, B. Bugs as deviant behavior: A general approach to inferring errors in systems code. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)* (Chateau Lake Louise, Banff, Canada, Oct. 2001), 57–72.
- Hoenicke, J., Leino, K.R.M., Podelski, A., Schäf, M., Wies, T. It's doomed; we can prove it. In *Proceedings of the 16th International Symposium on Formal Methods (FM)* (Eindhoven, the Netherlands, Nov. 2009), 338–353.
- Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2: Instruction Set Reference, A–Z*, Jan. 2013.
- ISO/IEC 9899:2011, Programming languages – C*, Dec. 2011.
- Jack, B. Vector rewrite attack: Exploitable NULL pointer vulnerabilities on ARM and XScale architectures. White paper, Juniper Networks, May 2007.
- Krebbbers, R., Wiedijk, F. Subtleties of the ANSI/ISO C standard. Document N1639, ISO/IEC, Sept. 2012.
- Lane, T. Anyone for adding `-fwrapv` to our standard CFLAGS? Dec. 2005. <http://www.postgresql.org/message-id/1689.1134422394@sss.pgh.pa.us>.
- Lattner, C. What every C programmer should know about undefined behavior, May 2011. <http://blog.livm.org/2011/05/what-every-c-programmer-should-know.html>.
- Lattner, C., Adve, V. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO)* (Palo Alto, CA, Mar. 2004), 75–86.
- Miller, W.M. C++ standard core language defect reports and accepted issues, issue 1457: Undefined behavior in left-shift, Feb. 2012. http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_defects.html#1457.
- Power ISA Version 2.06 Revision B, Book I: Power ISA User Instruction Set Architecture*, Jul. 2010.
- Ranise, S., Tinelli, C., Barrett, C. QF_BV logic, Jun. 2013. http://smtlib.cs.uiowa.edu/logics/QF_BV.smt2.
- Rationale for International Standard – Programming Languages – C*, Apr. 2003.
- Regehr, J. A guide to undefined behavior in C and C++, Jul. 2010. <http://blog.regehr.org/archives/213>.
- Regehr, J. Undefined behavior consequences contest winners, Jul. 2012. <http://blog.regehr.org/archives/767>.
- Seacord, R.C. Dangerous optimizations and the loss of causality, Feb. 2010. <https://www.securecoding.cert.org/confluence/download/attachments/40402999/Dangerous+Optimizations.pdf>.
- Stallman, R.M., the GCC Developer Community. *Using the GNU Compiler Collection for GCC 4.8.0*. GNU Press, Free Software Foundation, Boston, MA, 2013.
- Teo, E. [PATCH] add `-fno-delete-null-pointer-checks` to gcc CFLAGS, Jul. 2009. <https://lists.ubuntu.com/archives/kernel-team/2009-July/006609.html>.
- Tinnes, J. Bypassing Linux NULL pointer dereference exploit prevention (`mmap_min_addr`), Jun. 2009. <http://blog.cr0.org/2009/06/bypassing-linux-null-pointer.html>.
- Tomb, A., Flanagan, C. Detecting inconsistencies via universal reachability analysis. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis* (Minneapolis, MN, Jul. 2012), 287–297.
- Torvalds, L. Re: [patch] CFS scheduler, -v8, May 2007. <https://lkml.org/lkml/2007/5/7/213>.
- Tourrilhes, J. Invalid compilation without `-fno-strict-aliasing`, Feb. 2003. <https://lkml.org/lkml/2003/2/25/270>.
- Wang, X., Chen, H., Cheung, A., Jia, Z., Zeldovich, N., Kaashoek, M.F. Undefined behavior: What happened to my code? In *Proceedings of the 3rd Asia-Pacific Workshop on Systems* (Seoul, South Korea, Jul. 2012).
- Wang, X., Chen, H., Jia, Z., Zeldovich, N., Kaashoek, M.F. Improving integer security for systems with KInt. In *Proceedings of the 10th Symposium on Operating Systems Design and Implementation (OSDI)* (Hollywood, CA, Oct. 2012), 163–177.
- Wang, X., Zeldovich, N., Kaashoek, M.F., Solar-Lezama, A. Towards optimization-safe systems: Analyzing the impact of undefined behavior. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP)* (Farmington, PA, Nov. 2013), 260–275.
- Woods, J.F. Re: Why is this legal? Feb. 1992. <http://groups.google.com/group/comp.std.c/msg/dfef367547684b>.

Xi Wang ([xi]@cs.washington.edu), University of Washington, Seattle, WA.

Nickolai Zeldovich, M. Frans Kaashoek, and Armando Solar-Lezama ([nickolai, kaashoek, asolar]@csail.mit.edu), Massachusetts Institute of Technology, Cambridge, MA.

Technical Perspective

Taming the Name Game

By David Forsyth

VISION SYSTEMS GIVE animals two, somewhat different, kinds of information. The first is a model of the world they see. Our vision systems tell us where free space is (and so, where we could move); what is big and what is small; and what is smooth and what is scratchy.

Research in computer vision has now produced very powerful reconstruction methods. These methods can recover rich models of complex worlds from images and from video, and have had tremendous impact on everyday life. If you have seen a CGI film, you have likely seen representations recovered by one of these methods.

The second is a description of the world in terms of objects at a variety of levels of abstraction. Our vision systems can tell us that something is an animal; that it is a cat; and that it is the neighbor's cat. Computer vision has difficulty mimicking all these skills. We have really powerful methods for classifying images based on two technologies. First, given good feature vectors, modern classifiers—functions that report a class, given a feature vector, and that are learned from data—are very accurate. Second, with appropriate structural choices, one can learn to construct good features—this is the importance of convolutional neural networks. These methods apply to detection, too. One detects an object by constructing a set of possible locations for that object, then passing them to a classifier. Improvements in image classification and detection are so frequent that one can only keep precise track of the current state of the art by haunting ArXiv.

There remains a crucial difficulty: What should a system report about an image? It is likely a bad idea to identify each object in the image because there are so many and most do not matter (say, the bolt that holds the left front leg to your chair). So a system should report mainly objects that are important. The system now needs

to choose a name for each object that it reports. Many things can have the same name, because a world that consists entirely of wholly distinct things is too difficult to deal with. But the same thing could have many names, and choosing the best name becomes an issue. For example, when you see a swan, being told it is a “bird” is not particularly helpful (chickens are quite widely eaten), and you would probably expect better than “Cygnus olor,” too, because it might be Colombianus. But when they see a serval, people who have not encountered one before would feel their vision system was doing its job if it reported “fairly large cat.”

Psychologists argue there are basic-level categories that identify the best name for a thing. The choice of a basic-level category for a thing seems to be driven by its shape and appearance. For example, a sparrow and a wren might be in one basic-level bird category, and an ostrich and a rhea would be together in a different one. From the practical point of view, this idea is difficult to use because there is not

In the following paper, the authors offer a method to determine a basic-level category name for an object in an image.

The term one uses should be natural—something people tend to say.

much data about what the basic-level category for particular objects is.

In the following paper, the authors offer a method to determine a basic-level category name for an object in an image. The term one uses should be natural—something people tend to say. For example, one could describe a “King penguin” as such, or as a “seabird,” a “bird” or an “animal;” but “penguin” gives a nice balance between precision and generality, and is what most people use. The authors show how to use existing linguistic datasets to score the naturalness of a term. The term one uses should also tend to be correct for the image being described. More general terms are more likely to be correct (one can label pretty much anything “entity”). The authors show how to balance the likely correctness of a term, using a confidence score, with its naturalness.

Another strategy to identify basic-level categories is to look at terms people actually use when describing images. The authors look at captioned datasets to find nouns that occur often. They represent images using a set of terms, produced by one classifier. They then build another classifier to predict commonly occurring nouns from the first set. They require that most terms make no contribution to the predicted noun by enforcing sparsity in the second classifier; as a result, they can see what visual terms tend to produce which nouns (as Figure 6 illustrates, the noun “tree” is produced by a variety of specialized terms to do with vegetation, shrubbery, and so on). The result is an exciting link between terms of art commonly used in computer vision, and the basic categories of perceptual psychology. 

David Forsyth (daf@illinois.edu) is a professor in the computer science department at the University of Illinois, Urbana.

Copyright held by author.

Learning to Name Objects

By Vicente Ordonez, Wei Liu, Jia Deng, Yejin Choi, Alexander C. Berg, and Tamara L. Berg

Abstract

We have seen remarkable recent progress in computational visual recognition, producing systems that can classify objects into thousands of different categories with increasing accuracy. However, one question that has received relatively less attention is “what labels should recognition systems output?” This paper looks at the problem of predicting category labels that mimic how human observers would name objects. This goal is related to the concept of entry-level categories first introduced by psychologists in the 1970s and 1980s. We extend these seminal ideas to study human naming at large scale and to learn computational models for predicting entry-level categories. Practical applications of this work include improving human-focused computer vision applications such as automatically generating a natural language description for an image or text-based image search.

1. INTRODUCTION

Computational visual recognition is beginning to work. Although far from solved, algorithms for analyzing images have now advanced to the point where they can recognize or localize thousands of object categories with reasonable accuracy.^{3, 14, 24, 25} While we could predict any one of many relevant labels for an object, the question of “What *should* I actually call it?” is becoming important for large-scale visual recognition. For instance, if a classifier were lucky enough to get the example in Figure 1 correct, it might output *Cygnus Colombianus*, while most people would probably simply say *swan*. Our goal is to learn models to map from specific, encyclopedic terms (*Cygnus Colombianus*) to how people might refer to a given object (*swan*).

These learned mappings could add a new type of structure to hand-built linguistic resources, such as WordNet.⁹ WordNet enumerates a large set of English nouns augmented by relationships, including hyperonymy (*is-a* connections) linking more general categories, for example, passerine, to more specific categories, for example, firebird (a firebird is a kind of passerine). Our models might learn that an image of a firebird is more likely to be described by the term “bird” instead of a more technical term like “passerine.” When combined with a computer vision system that attempts to recognize many very specific types of objects in a particular image, our models allow mapping to the words people are likely to use for describing the depicted objects. For end-user applications, these types of outputs may be more useful than the outputs of very accurate but overly specific visual categorization systems. This is especially relevant for human computer interaction mediated by text—for instance, in text-based image search.

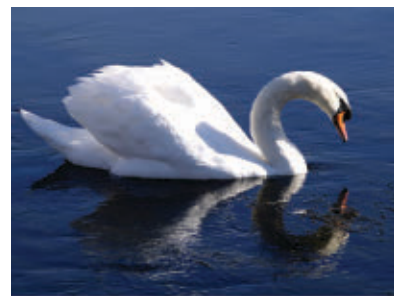
Our work is inspired by previous research on *basic and entry-level categories* formulated by psychologists, including Rosch²³ and Kosslyn.¹³ Rosch defines *basic-level categories*

as those categories at the highest level of generality whose members still share many common attributes and have fewer distinctive attributes. An example of a basic level category is *bird* where most instances share attributes like having feathers, wings, and beaks. Subordinate, more specific categories, such as *American Robin* will have members that share even more attributes such as shape, color, and size. Super-ordinate, more general categories, such as *animal* have members that share fewer attributes and demonstrate more variability. Rosch studied basic level categories through human experiments, for example, asking people to enumerate common attributes for a given category.

The work of Jolicoeur et al.¹³ further studied the way people identify categories, defining the notion of *entry-level categories*. Entry-level categories are essentially the categories that people will naturally use to identify objects. The more prototypical an object the more likely it will have its entry point at the basic-level category. For less typical objects the entry point might be at a lower level of abstraction. For example an *American robin* or a *penguin* are both members of the same basic-level *bird* category. However, the *American robin* is more prototypical, sharing many features with other birds and thus its entry-level category coincides with its basic-level category of *bird*, while a *penguin* would be identified at a lower level of abstraction (see Figure 2).

Thus, while objects are members of many categories—for

Figure 1. Example translation from a WordNet based object category prediction to what people might call the depicted object.



Recognition Prediction

What Should I Call It?

Cygnus Colombianus



Swan

The original version of this paper is entitled “From Large Scale Image Categorization to Entry-Level Categories” and was published in *International Conference on Computer Vision*, December 2013, IEEE/CVF. A later version of this paper is entitled “Predicting Entry-Level Categories” and was submitted to *International Journal of Computer Vision – Marr Prize Special Issue*. November 2014, Springer.

Figure 2. An American Robin is a more prototypical type of bird hence its entry-level category coincides with its basic-level category while for penguin which is a less prototypical example of bird, the entry-level category is at a lower level of abstraction.



Superordinates: animal, vertebrate
Basic Level: bird
Entry Level: bird
Subordinates: American robin



Superordinates: animal, vertebrate
Basic Level: bird
Entry Level: penguin
Subordinates: Chinstrap penguin

example, Mr. Ed is a palomino, but also a horse, an equine, an odd-toed ungulate, a placental mammal, a mammal, and so on—most people looking at Mr. Ed would tend to call him a *horse*, his entry level category (unless they are fans of the show). Our paper focuses on the problem of object naming in the context of *entry-level categories*. We consider two related tasks: (1) learning a mapping from *fine-grained/encyclopediaic categories*—for example, leaf nodes in WordNet⁹—to what people are likely to call them (*entry-level categories*) and (2) learning to map from outputs of thousands of noisy computer vision classifiers/detectors evaluated on an image to what a person is likely to call depicted objects.

Evaluations show that our models can effectively emulate the naming choices of humans. Furthermore, we show that using noisy computer vision estimates for image content, our system can output words that are significantly closer to human annotations than either the raw visual classifier predictions or the results of using a state of the art hierarchical classification system⁶ that can output object labels at varying levels of abstraction, from very specific terms to very general categories.

1.1. Outline

The rest of this paper is organized as follows. Section 2 presents a summary of related work. Section 3 introduces a large-scale image categorization system based on deep convolutional neural network (CNN) activations. In Section 4, we learn translations between input linguistic concepts and entry-level concepts. In Section 5, we propose two models that can take an image as input and predict entry-level concepts for the depicted objects. Finally, in Sections 6 and 7 we provide experimental evaluations and conclusions.

2. RELATED WORK

Questions about *entry-level categories* are directly relevant to recent work on generating natural language descriptions for images.^{8, 11, 15, 16, 19, 21} In these papers, the goal is to automatically produce natural language that describes the content of an image or video. We attack one specific facet of this problem, how to name objects in images in a human-like manner. Previous approaches that construct image descriptions directly from computer vision predictions often result

in unnatural constructions, for example, “Here we see one TV-monitor and one window.”¹⁵ Other methods handle naming choices indirectly by sampling human written text written about other visually similar objects.^{16, 17}

On a technical level, our work is related to recent work from Deng et al.⁶ that tries to “hedge” predictions of visual content by *optimally* backing off in the WordNet semantic hierarchy. For example, given a picture of a *dog*, a noisy visual predictor might easily mistake this for a *cat*. Therefore, outputting a more general prediction, for example, *animal*, may sometimes be better for overall performance in cases of visual ambiguity. One key difference is that our approach uses a reward function over the WordNet hierarchy that is non-monotonic along paths from the root to leaves, as it is based on word usage patterns, rather than perplexity. Another difference is that we make use of recent convolutional neural network based features for our underlying visual classifiers.¹² Our approach also allows mappings to be learned from a WordNet leaf node, *l*, to natural word choices that are not along a path from *l* to the root, “entity.” In evaluations, our results significantly outperform the “hedging” technique⁶ because although optimal for maximizing classification accuracy, it is not optimal with respect to how people describe image content.

Our work is also related to the growing challenge of harnessing the ever increasing number of pre-trained recognition systems, thus avoiding always “starting from scratch” in developing new applications. With the advent of large labeled datasets of images, including ImageNet⁵ with over 15,000,000 labeled images for a subset of the WordNet hierarchy, a large amount of compute effort has been dedicated to building vision based recognition systems. It would be wasteful not to take advantage of the CPU weeks,^{10, 14} months,^{4, 6} or even millennia¹⁸ invested in developing and training such recognition models. However, for any specific end user application, the categories of objects, scenes, and attributes labeled in a particular dataset may not be the most useful predictions. One benefit of our work can be seen as exploring the problem of translating the outputs of a vision system trained with one vocabulary of labels (WordNet leaf nodes) to labels in a new vocabulary (commonly used visually descriptive nouns).

Our proposed methods take into account several sources of structure and information: the structure of WordNet, frequencies of word use on the web,² outputs of a large-scale visual recognition system,¹² and large amounts of paired image and text data. In particular, we make use of the SBU Captioned Photo Dataset²¹ which contains 1 million images with natural language captions as a source of natural image naming patterns. By incorporating all of these resources, we are able to study entry-level categories at a much larger scale than in previous settings.

2.1. Challenges of predicting entry-level categories

At first glance, the task of finding the entry-level categories may seem like a linguistic problem of finding a *hypernym* of any given word. Although there is a considerable conceptual connection between entry-level categories and hypernyms, there are two notable differences:

1. Although “*bird*” is a hypernym of both “*penguin*,” and

“sparrow,” “bird” may be a good entry-level category for “sparrow,” but not for “penguin.” This phenomenon, that some members of a category are more prototypical than others, is discussed in *Prototype Theory*.²³

2. Entry-level categories are not confined by (inherited) hypernyms, in part because encyclopedic knowledge is different from common sense knowledge. For example “rhea” is not a kind of “ostrich” in the strict taxonomical sense. However, due to their visual similarity, people generally refer to a “rhea” as an “ostrich.” Adding to the challenge is that although extensive, WordNet is neither complete nor practically optimal for our purpose. For example, according to WordNet, “kitten” is not a kind of “cat,” and “tulip” is not a kind of “flower.”

In fact, both of the above points have a connection to visual information of objects, as visually similar objects are more likely to belong to the same entry-level category. In this work, we present the first extensive study that (1) characterizes entry-level categories in the context of translating encyclopedic visual categories to natural names that people commonly use, and (2) provides approaches that infer entry-level categories from a large-scale image corpus, guided by semantic word knowledge.

3. A LARGE-SCALE IMAGE CATEGORIZATION SYSTEM

We take advantage of recent advances in deep learning based visual features for training a large number of visual classifiers for leaf-node object categories. In particular, we use the pre-trained CNN model from the Caffe framework¹² based on the model from Krizhevsky et al.,¹⁴ trained on 1000 imagenet categories from the ImageNet Large Scale Visual Recognition Challenge 2012. This model consists of a feed-forward neural network with multiple layers, each with different levels of connectivity between units in contiguous layers. The last few layers in the network consist of fully connected layers, where all the units in a given layer are connected to all the units in the subsequent layer. The output layer of the network consists of 1000 units corresponding to each category for the classification task. Donahue et al.⁷ showed that the activations of some of the intermediate layers, particularly the fully connected layers before the output layer, were a useful generic image representation for a variety of other recognition tasks.

We similarly compute the 4096 activations in the last fully connected layer of this network and use these as features to train a linear SVM for each of 7404 leaf level categories in ImageNet. We also use a validation set to calibrate the output scores of each SVM using Platt scaling.²² These 7404 visual classifiers will be used to predict image content either directly (Section 5.1) or to train entry-level visual predictors (Sections 4.2 and 5.2).

4. TRANSLATING ENCYCLOPEDIA CONCEPTS TO ENTRY-LEVEL CONCEPTS

Our first goal toward understanding how people name objects, is to learn mappings between encyclopedic concepts (ImageNet leaf categories, e.g., *Chlorophyllum molybdites*) and concepts that are more *natural* (e.g., mushroom). In Section 4.1, we present an approach that relies on the WordNet

hierarchy and frequencies of words in a web scale corpus. In Section 4.2, we follow an approach that uses visual recognition models learned on a paired image-caption dataset.

4.1. Language-based translation

As a baseline, we first consider a translation approach that relies only on language-based information: the hierarchical semantic tree from WordNet⁹ and text statistics from the Google Web 1T corpus.² We posit that the frequencies of terms computed from massive amounts of text on the web reflect the “naturalness” of concepts. We use the n-gram counts of the Google Web 1T corpus² as a proxy for naturalness. Specifically, for a synset w , we quantify naturalness as, $\phi(w)$, the log of the count for the most commonly used synonym in w . As possible translation concepts for a given category, v , we consider all nodes, w in v ’s inherited hypernym structure (all of the synsets along the WordNet path from w to the root).

We define a translation function for a category v , $\tau(v, \lambda)$, that maps v to a new node w , such that w maximizes the trade-off between naturalness, $\phi(w)$, and semantic proximity, $\psi(w, v)$, measuring the distance between node v and node w in the WordNet hypernym structure:

$$\tau(v, \lambda) = \arg \max_w [\phi(w) - \lambda \psi(w, v)], w \in \Pi(v), \quad (1)$$

where $\Pi(v)$ is the set of (inherited) hypernyms from v to the root, including v . For instance given an input category $v = \text{King penguin}$ we consider all categories along its set of inherited hypernyms, for example, *penguin*, *seabird*, *bird*, *animal* (see Figure 3). An ideal prediction for this concept would be *penguin*. We use line search to find the optimal λ , which controls how much we care about naturalness versus semantic proximity, based on a held out set of subordinate-category, entry-level category pairs $D = (x_i, y_i)$ collected using crowdsourcing to maximize the number of correct translations predicted by our model:

$$\Phi(D, \lambda) = \sum_i \mathbb{1}[\tau(x_i, \lambda) = y_i], \quad (2)$$

where $\mathbb{1}[\cdot]$ is the indicator function. We show the relationship between λ and translation accuracy, $\Phi(D, \lambda)$, in Figure 4, where

Figure 3. Our first categorical translation model uses the WordNet hierarchy to find an hypernym that is close to the leaf node concept (semantic distance) and has a large naturalness score based on its n-gram frequency. The green arrows indicate the ideal category that would correspond to the entry-level category for each leaf-node in this sample semantic hierarchy.

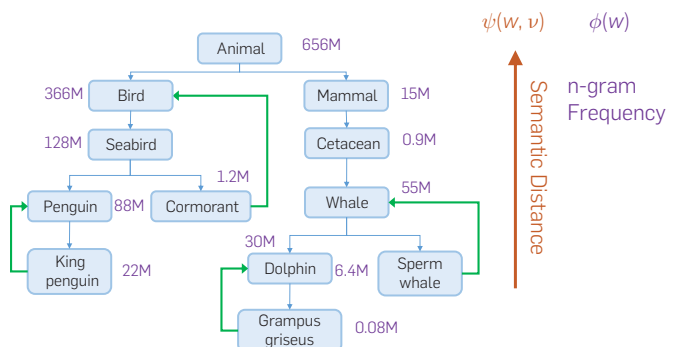
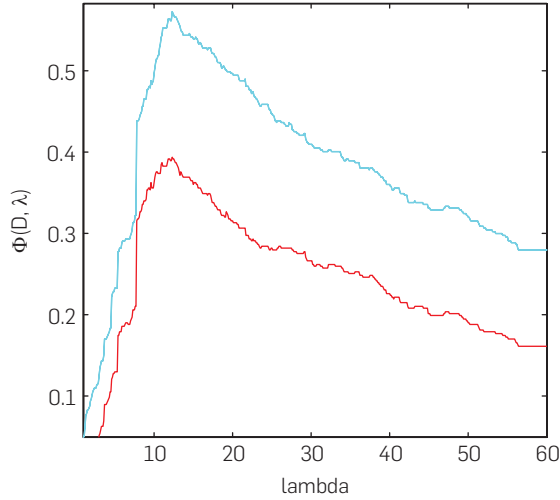


Figure 4. Relationship between parameter λ and translation accuracy, $\Phi(D, \lambda)$, evaluated on the most agreed human label (red) or any human label (cyan).



the red line shows accuracy for predicting the word used by the most people for a synset, while the cyan line shows the accuracy for predicting any word used by a labeler for the synset. As we increase λ , $\Phi(D, \lambda)$ increases initially and then decreases as too much generalization or specificity reduces the naturalness of the predictions. For example, generalizing from *grampus griseus* to *dolphin* is good for “naturalness,” but generalizing all the way to “entity” decreases “naturalness.”

Our experiment also supports the idea that *entry-level categories* lie at a level of abstraction where there is a discontinuity. Going beyond this level of abstraction suddenly makes our predictions considerably worse. Rosch²³ indeed argues in the context of basic level categories that basic cuts in categorization happen precisely at these discontinuities where there are bundles of information-rich functional and perceptual attributes.

4.2. Visual-based translation

Next, we try to make use of pre-trained visual classifiers to improve translations between input concepts and entry-level concepts. For a given leaf synset, v , we sample a set of $n = 100$ images from ImageNet. For each image, i , we predict some potential entry-level nouns, N_i , using pre-trained visual classifiers that we will further describe in Section 5.2. We use the union of this set of labels $N = N_1 \cup N_2 \dots \cup N_n$ as keyword annotations for synset v and rank them using a *term frequency-inverse document frequency* (TFIDF) information retrieval measure. This ranking measure promotes labels that are predicted frequently for our set of 100 images, while decreasing the importance of labels that are predicted frequently in all our experiments across different categories. We pick the most highly ranked noun for each node, v , as its entry-level categorical translation.

We show a comparison of the output of this approach with our Language-based Translation approach and mappings provided by human annotators in Table 1. We explain the collection of human annotations in the evaluation section (Section 6.1).

Table 1. Translations from ImageNet leaf node synset categories to entry-level categories using our automatic approaches from Sections 4.1 (left) and 4.2 (center) and crowd-sourced human annotations (right).

Input concept	Language-based translation	Visual-based translation	Human translation
Cactus wren	Bird	Bird	Bird
Buzzard, Buteo buteo	Hawk	Hawk	Hawk
Whinchat, Saxicola rubetra	Chat	Bird	Bird
Weimaraner	Dog	Dog	Dog
Numbat, banded anteater, anteater	Anteater	Dog	Anteater
Rhea, Rhea americana	Bird	Grass	Ostrich
Conger, conger eel	Eel	Fish	Fish
Merino, merino sheep	Sheep	Sheep	Sheep
Yellowbelly marmot, rockchuck	Marmot	Male	Squirrel
Snorkeling, snorkel diving	Swimming	Sea turtle	Snorkel

5. PREDICTING ENTRY-LEVEL CONCEPTS FOR IMAGES

In Section 4, we proposed models to translate between one linguistic concept, for example, *grampus griseus*, to a more natural object name, for example, *dolphin*. Our objective in this section is to explore methods that can take an image as input and predict entry-level labels for the depicted objects. The models we propose are: (1) a method that combines “naturalness” measures from text statistics with direct estimates of visual content computed at leaf nodes and inferred for internal nodes (Section 5.1) and (2) a method that learns visual models for entry-level category prediction directly from a large collection of images with associated captions (Section 5.2).

5.1. Linguistically guided naming

In our first image prediction method, we estimate image content for an image, I , using the pre-trained visual models described in Section 3. These models predict the presence or absence of 7404 leaf level visual categories in the ImageNet (WordNet) hierarchy. Following the “hedging” approach,⁶ we compute estimates of visual content for internal nodes in the hierarchy by accumulating all predictions below a node:

$$f(v, I) = \begin{cases} \hat{f}(v, I), & \text{if } v \text{ is an leaf node,} \\ \sum_{v' \in Z(v)} \hat{f}(v', I), & \text{if } v \text{ is an internal node,} \end{cases} \quad (3)$$

where $Z(v)$ is the set of all leaf nodes under node v and $\hat{f}(v, I)$ is the output of a platt-scaled decision value from a linear SVM trained to recognize category v . Similar to our approach in Section 4.1, we define for every node in the ImageNet hierarchy

a trade-off function between “naturalness” ϕ (n -gram counts) and specificity $\tilde{\psi}$ (relative position in the WordNet hierarchy):

$$\gamma(v, \hat{\lambda}) = [\phi(w) - \hat{\lambda}\tilde{\psi}(w)], \quad (4)$$

where $\phi(w)$ is computed as the log counts of the nouns and compound nouns in the text corpus from the *SBU Captioned Dataset*,²¹ and $\tilde{\psi}(w)$ is an upper bound on $\psi(w, v)$ from Equation (1) equal to the maximum height in the WordNet hierarchy for node w . We parameterize this trade-off by $\hat{\lambda}$.

For entry-level category prediction for images, we would like to maximize both “naturalness” and visual content estimates. For example, text based “naturalness” will tell us that both *cat* and *swan* are good entry-level categories, but a confident visual prediction for *Cygnus Colombianus* for an image tells us that *swan* is a much better entry-level prediction than *cat* for that image.

Therefore, for an input image, we want to output a set of concepts that have a large prediction for both “naturalness” and content estimate score. For our experiments we output the top K WordNet synsets with the highest f_{nat} scores:

$$f_{nat}(v, I, \hat{\lambda}) = f(v, I) \gamma(v, \hat{\lambda}). \quad (5)$$

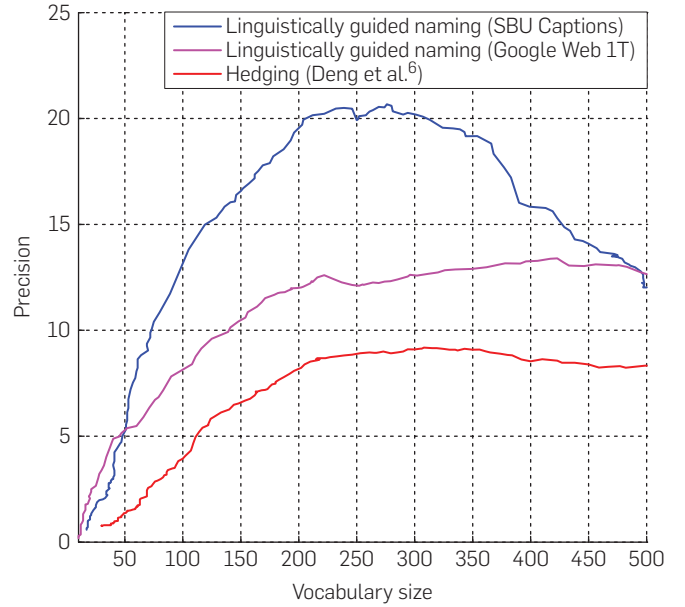
As we change $\hat{\lambda}$ we expect similar behavior to our concept translations (Section 4.1), tuning $\hat{\lambda}$ to control the degree of specificity while trying to preserve “naturalness.” We compare our framework to the “hedging” technique⁶ for different settings of $\hat{\lambda}$. For a side by side comparison we modify hedging to output the top K synsets based on their scoring function. Here, the working vocabulary is the unique set of predicted labels output for each method on this test set. Results demonstrate (Figure 5) that under different parameter settings we *consistently* obtain much higher levels of precision for predicting entry-level categories than hedging.⁶ We also obtain an additional gain in performance over our previous work²⁰ by incorporating dataset-specific text-statistics from the *SBU Captioned Dataset* rather than the more generic *Google Web 1T* corpus.

5.2. Visually guided naming

In the previous section, we rely on WordNet structure to compute estimates of image content, especially for internal nodes. However, this is not always a good measure of content prediction because: (1) The WordNet hierarchy doesn’t encode knowledge about some semantic relationships between objects (i.e., functional or contextual relationships), (2) Even with the vast coverage of 7404 ImageNet leaf nodes we are missing models for many potentially important entry-level categories that are not at the leaf level.

As one alternative, we can directly train models for entry-level categories from data where people have provided entry-level labels—in the form of nouns present in visually descriptive image captions. We postulate that these nouns represent examples of entry-level labels because they have been naturally annotated by people to describe what is present in an image. For this task, we leverage the SBU Captioned Photo Dataset²¹ which contains 1 million captioned images. We transform this dataset into a set $D = \{X^{(i)}, Y^{(i)} \mid X^{(i)} \in \mathbf{X}, Y^{(i)} \in \mathbf{Y}\}$, where $\mathbf{X} = [0-1]^s$ is a vector of estimates of visual content for

Figure 5. Relationship between average precision agreement and working vocabulary size (on a set of 1000 images) for the hedging method (red) and our Linguistically guided naming method using text statistics from the generic Google Web 1T dataset (magenta) and from the SBU Caption Dataset (Section 5.1). We use $K = 5$ to generate this plot and a random set of 1000 images from the SBU Captioned Dataset.



$s = 7404$ ImageNet leaf node categories and $\mathbf{Y} = [0, 1]^d$ is a set of binary output labels for d target categories. Input content estimates are provided by the deep learning based SVM predictions (described in Section 3).

For training our d target categories, we obtain labels Y from the million captions by running a POS-tagger¹ and defining $Y^{(j)} = \{y_{ij}\}$ such that:

$$y_{ij} = \begin{cases} 1, & \text{if caption for image } j \text{ has noun } i, \\ 0, & \text{if otherwise.} \end{cases} \quad (6)$$

The POS-tagger helps clean up some word sense ambiguity due to polysemy, by only selecting those instances where a word is used as a noun. The number of target categories, d , is determined experimentally from data by learning models for the most frequent nouns in this dataset. This provides us with a target vocabulary that is both likely to contain entry-level categories (because we expect entry-level category nouns to commonly occur in our visual descriptions) and to contain sufficient images for training effective recognition models. We use up to 10,000 images for training each model. Since we are using human labels from real-world data, the frequency of words in our target vocabulary follows a power-law distribution. Hence we only have a very large amount of training data for a few most commonly occurring noun concepts. Specifically, we learn linear SVMs followed by Platt scaling for each of our target concepts. We keep $d = 1169$ of the best performing models. Our scoring function f_{svm} for a target concept v_i is then:

$$f_{svm}(v_i, I, \theta_i) = \frac{1}{1 - \exp(a_i \theta_i^\top X + b_i)}, \quad (7)$$

where θ_i are the model parameters for predicting concept v_i ,

and a_i and b_i are Platt scaling parameters learned for each target concept v_i on a held out validation set.

$$R(\theta_i) = \frac{1}{2} \|\theta_i\|^2 + c \sum_{j=1}^{|D|} \max(0, 1 - y_{ij} \theta_i^\top X^{(j)})^2. \quad (8)$$

We learn the parameters θ_i by minimizing the squared hinge-loss with ℓ_1 regularization (Equation 8). The latter provides a natural way of modeling the relationships between the input and output label spaces that encourages sparseness (examples in Figure 6). We find $c = 0.01$ to yield good results for our problem and use this value for training all individual models.

One of the drawbacks of using the ImageNet hierarchy to aggregate estimates of visual concepts (Section 5.1) is that it ignores more complex relationships between concepts. Here, our data-driven approach to the problem implicitly discovers these relationships. For instance a concept like *tree* has co-occurrence relationships with various types of birds, and other animals that live on trees (see Figure 6).

Given this large dataset of images with noisy visual predictions and text labels, we manage to learn quite good predictors of high-level content, even for categories with relatively high intra-class variation (e.g., girl, boy, market, house).

6. EXPERIMENTAL EVALUATION

We evaluate results on our two proposed naming tasks—learning translations from encyclopedic concepts to entry-level concepts (Section 6.1), and predicting entry-level concepts for objects in images (Section 6.2).

6.1. Evaluating translations

We use Amazon Mechanical Turk to crowd source translations of ImageNet synsets into entry-level categories $D = \{x_i, y_i \mid x_i \text{ is a leaf node, } y_i \text{ is a word}\}$. Our experiments present users with a 2×5 array of images sampled from an ImageNet synset, x_i , and users are asked to provide a label, y_i , for the depicted concept. Results are obtained for 500 ImageNet synsets and aggregated across 8 users per task. We found agreement (measured as at least 3 of 8 users agreeing) among users for 447 of the 500 concepts, indicating that even though there are many potential labels for each synset (e.g., *Sarcophaga carnaria* could conceivably be labeled as fly,

dipterous insect, insect, arthropod, etc.) people have a strong preference for particular entry-level categories.

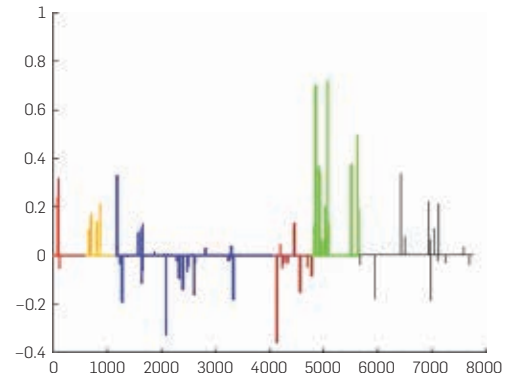
We show sample results from each of our methods to learn concept translations in Table 1. In some cases linguistics-based translation fails. For example, *whinchat* (a type of bird) translates to “chat” most likely because of the inflated counts for the most common use of “chat.” Our visual-based translation fails when it learns to weight context words highly, for example “snorkeling” → “water,” or “African bee” → “flower” even when we try to account for common context words using TFIDF. Finally, even humans are not always correct, for example “Rhea Americana” looks like an ostrich, but is not taxonomically one. Even for categories like “marmot” most people named it “squirrel.” Overall, our language-based translation (Section 4.1) agrees 37% of the time with human supplied translations and the visual-based translation (Section 4.2) agrees 33% of the time, indicating that translation learning is a non-trivial task. This experiment expands on previous studies in psychology.^{13, 23} Cheap and easy online crowdsourcing enables us to gather these labels for a much larger set of (500) concepts than previous experiments and to learn generalizations for a substantially larger set of ImageNet synsets.

6.2. Evaluating image entry-level predictions

We measure the accuracy of our proposed entry-level category image prediction methods by evaluating how well we can predict nouns freely associated with images by users on Amazon Mechanical Turk. Results are evaluated on a test set containing 1000 images selected at random from the million image dataset. We additionally collect annotations for another 2000 images so that we can tune trade-off parameters in our models. This test set is completely disjoint from the sets of images used for learning the pre-trained visual models. For each image, we instruct three users on MTurk to write down any nouns that are relevant to the image content. Because these annotations are free associations we observe a large and varied set with 3610 distinct nouns total in our evaluation sets. This makes noun prediction extremely challenging!

Figure 6. Entry-level category *tree* with its corresponding top weighted leaf node features after training an SVM on our noisy data, and a visualization of weights grouped by an arbitrary categorization of leaf nodes. Vegetation (green), birds (orange), instruments (blue), structures (brown), mammals (red), and others (black).

tree → iron tree, iron-tree, ironwood, ironwood tree
snag
European silver fir, Christmas tree, Abies alba
baobab, monkey-bread tree, Adansonia digitata
Japanese black pine, black pine, Pinus thunbergii
huisache, cassie, mimosa bush, sweet wattle, sweet acacia, scented wattle,
flame tree, Acacia farnesiana
feeder
bird feeder, birdfeeder, feeder
koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus
flying fox
damask
American basswood, American lime, Tilia americana



For evaluation, we measure how well we can predict all nouns associated with an image by Turkers (Figure 7a) and how well we can predict the nouns commonly associated by Turkers (assigned by at least two of three Turkers, Figure 7b). For reference we compute the precision of one human annotator against the other two and found that on our test set humans were able to predict what the previous annotators labeled with 0.35 precision when compared to the agreed set of nouns by Turkers.

Results show precision and recall for prediction on our test set, comparing: leaf node classification performance (flat classifier), the outputs of “hedging,”⁶ and our proposed entry-level category predictors (Linguistically guided, Section 5.1, and Visually guided, Section 5.2). Performance on the test set is admirable for this challenging task. On the two datasets we find the Visually guided naming model to perform better (Section 5.2) than the Linguistically guided naming (Section 5.1). In addition, we significantly outperform both leaf node classification and the “hedging”

technique.⁶ We show an image with sample output from our methods at $K = 5$ in Figure 8.

7. CONCLUSION

We have explored models for mapping encyclopedic concepts to entry-level concepts, and for predicting natural names for objects depicted in images. Results indicate that our inferred concept translations are meaningful and that our models provide a first step toward predicting entry-level categories—the nouns people use to name objects—depicted in images. These methods could be helpful for many different end-user applications that require recognition outputs that are useful for human consumption, including tasks related to description generation and image retrieval from complex text queries.

Acknowledgments

This work was partially supported by NSF Career Award #1444234 and NSF Award #1445409.

Figure 7. Precision-recall curve for different entry-level prediction methods when using the top K categorical predictions for $K = 1, 3, 5, 10, 15, 20, 50$. (a) An evaluation using the union of all human labels as ground truth and (b) using only the set of labels where at least two users agreed.

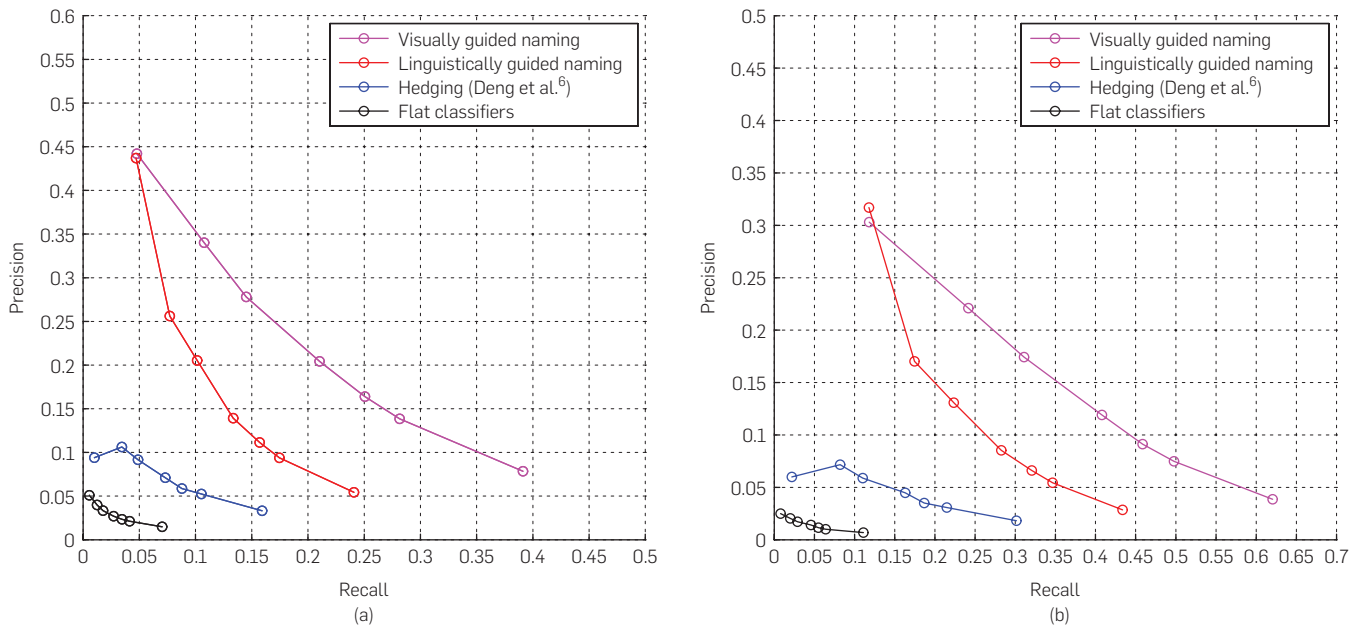


Figure 8. Category predictions for an example input image for a large-scale categorization system and our translated outputs using linguistically and visually guided models. The first column contains nouns associated with the image by people. We highlight in green the predicted nouns that were also mentioned by people. Note that *oast* is a type of farm building for drying hops and a *dacha* is a type of Russian farm building.

Input image	Human categorization (crowdsourcing)	Large-scale categorization system	Linguistically guided naming (our work)	Visually guided naming (our work)
	barn building fence house tree yard	corncrib oast farmhouse log cabin dacha	building house home tent tree	house barn wooden roof farm

References

1. Bird, S. Nltk: The natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions* (July 2006). Association for Computational Linguistics, Sydney, Australia, 69–72.
2. Brants, T., Franz, A. Web 1t 5-gram version 1. In *Linguistic Data Consortium (LDC)* (2006), Linguistic Data Consortium, Philadelphia.
3. Dean, T., Ruzon, M.A., Segal, M., Shlens, J., Vijayanarasimhan, S., Yagnik, J. Fast, accurate detection of 100,000 object classes on a single machine. In *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2013), 1814–1821.
4. Deng, J., Berg, A.C., Li, K., Li, F.-F. What does classifying more than 10,000 image categories tell us? In *European Conference on Computer Vision (ECCV)*, Daniilidis, Kostas and Maragos, Petros and Paragios, Nikos, eds. Volume 6315 of *Lecture Notes in Computer Science* (2010), Springer, Berlin, Heidelberg, 71–84.
5. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009 (June 2009), 248–255.
6. Deng, J., Krause, J., Berg, A.C., Fei-Fei, L. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012 (June 2012), 3450–3457.
7. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition, 2013. arXiv preprint arXiv:1310.1531.
8. Farhadi, A., Hejrati, M., Sadeghi, M.A., Young, P., Rashtchian, C., Hockenmaier, J., Forsyth, D. Every picture tells a story: Generating sentences for images. In *European Conference on Computer Vision (ECCV)*, Daniilidis, Kostas and Maragos, Petros and Paragios, Nikos, eds. Volume 6314 of *Lecture Notes in Computer Science* (2010), Springer, Berlin, Heidelberg, 15–29.
9. Fellbaum, C., ed. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, Massachusetts, 1998.
10. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.* 32, 9 (Sept 2010), 1627–1645.
11. Hodosh, M., Young, P., Hockenmaier, J. Framing image description as a ranking task: Data, models and evaluation metrics. *J. Artif. Int. Res.* 47, 1 (May 2013), 853–899.
12. Jia, Y. Caffe: An open source convolutional architecture for fast feature embedding, 2013. <http://caffe.berkeleyvision.org/>.
13. Jolicoeur, P., Gluck, M.A., Kosslyn, S.M. Pictures and names: Making the connection. *cognitive psychology. Cogn. Psychol.* 16, (1984), 243–275, 1984.
14. Krizhevsky, A., Sutskever, I., Hinton, G. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira and C.J.C. Burges and L. Bottou and K.Q. Weinberger, eds. (2012), Curran Associates, Inc., 1097–1105.
15. Kulkarni, G., Premraj, V., Ordonez, V., Dhar, S., Li, S., Choi, Y., Berg, A., Berg, T. Babytalk: Understanding and generating simple image descriptions. *IEEE Trans. Pattern Anal. Machine Intell.* 35, 12 (Dec 2013), 2891–2903.
16. Kuznetsova, P., Ordonez, V., Berg, A., Berg, T.L., Choi, Y. Collective generation of natural image descriptions. In *Association for Computational Linguistics (ACL)*, 2012.
17. Kuznetsova, P., Ordonez, V., Berg, T., Choi, Y. Treetalk: Composition and compression of trees for image descriptions. *Trans. Assoc. Comput. Linguist.* 2, 1 (2014), 351–362.
18. Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., Ng, A. Building high-level features using large scale unsupervised learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML12)*, John Langford and Joelle Pineau, eds. (Edinburgh, Scotland, GB, July 2012), Omnipress, New York, NY, USA, 81–88.
19. Mitchell, M., Han, X., Dodge, J., Mensch, A., Goyal, A., Berg, A., Yamaguchi, K., Berg, T., Stratos, K., Daumé, H. III. Midge: Generating image descriptions from computer vision detections. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics* (April 2012), Association for Computational Linguistics, Avignon, France, 747–756.
20. Ordonez, V., Deng, J., Choi, Y., Berg, A.C., Berg, T.L. From large scale image categorization to entry-level categories. In *2013 IEEE International Conference on Computer Vision (ICCV)* (Dec 2013), 2768–2775.
21. Ordonez, V., Kulkarni, G., Berg, T.L. Im2text: Describing images using 1 million captioned photographs. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, eds. (2011), Curran Associates, Inc., 1143–1151.
22. Platt, J.C. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers* (1999), MIT Press, 61–74.
23. Rosch, E. Principles of categorization. In *Cognition and Categorization*, E. Rosch and B.B. Lloyd, eds. (1978), 27–48.
24. Simonyan, K., Zisserman, A. Very deep convolutional networks for large-scale image recognition, Sept. 2014. arXiv preprint arXiv:1409.1556.
25. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. Going deeper with convolutions, Sept. 2014. arXiv preprint arXiv:1409.4842.

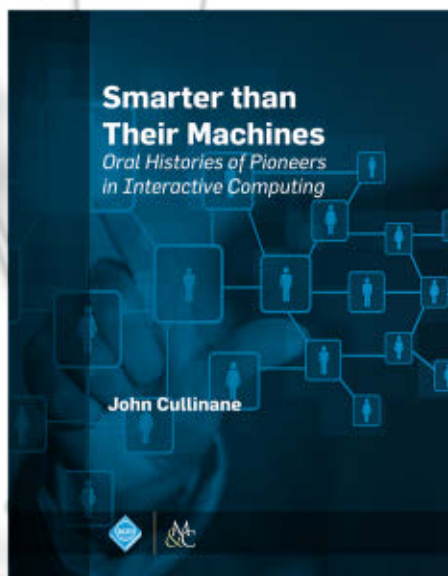
Vicente Ordonez (vicenteor@alienai.org), Allen Institute for Artificial Intelligence, Seattle, WA.

Wei Liu, Alexander C. Berg, and Tamara L. Berg ({wliu, aberg, tberg}@cs.unc.edu), Department of Computer Science, University of North Carolina at Chapel Hill, NC.

Jia Deng (jiadeng@umich.edu), Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI.

Yejin Choi (yejin@cs.washington.edu), Department of Computer Science and Engineering, University of Washington, Seattle, WA.

Copyright held by authors. Publication rights licensed to ACM. \$15.00.



A personal walk down the computer industry road. BY AN EYEWITNESS.

Smarter Than Their Machines: Oral Histories of the Pioneers of Interactive Computing is based on oral histories archived at the Charles Babbage Institute, University of Minnesota. These oral histories contain important messages for our leaders of today, at all levels, including that government, industry, and academia can accomplish great things when working together in an effective way.



ISBN: 978-1-62705-550-5 DOI: 110.1145/2663015
<http://books.acm.org>
<http://www.morganclaypoolpublishers.com/acm>

Are you looking for your next IT job?

Do you need Career Advice?

The **ACM Career & Job Center** offers ACM members a host of career-enhancing benefits:

- A **highly targeted focus** on job opportunities in the computing industry
- **Access to hundreds** of industry job postings
- Resume posting **keeping you connected** to the employment market while letting you maintain full control over your confidential information
- **Job Alert system** that notifies you of new opportunities matching your criteria
- **Career coaching** and guidance available from trained experts dedicated to your success
- **Free access** to a content library of the best career articles compiled from hundreds of sources, and much more!



Visit **ACM's Career & Job Center** at:
<http://jobs.acm.org>



Association for
Computing Machinery

Advancing Computing as a Science & Profession

The **ACM Career & Job Center** is the perfect place to begin searching for your next employment opportunity!

Visit today at **<http://jobs.acm.org>**

Lamar University
Department of Computer Science
Assistant Professor

Lamar University's Department of Computer Science seeks applications for a tenure-track **Assistant Professor** position beginning fall 2016. Applicants must have a PhD in Computer Science and a strong commitment to teaching and research in cyber-security. Lamar is an AA/EEO state supported university of approximately 15,000 students. It offers the B.S. and M.S. in Computer Science. There are 9 full-time faculty and 500 undergraduate and graduate majors. Review of applications will begin on March 1, 2016, and continue until the position is filled. **Apply at** <http://jobs.lamar.edu/postings/3118>

If you have additional questions, please address them to Stefan.Andrei@lamar.edu

Toyota Technological Institute (TTI)
Faculty Position Open

Toyota Technological Institute (TTI) has an opening for a professor (tenured or tenure-track), or an associate professor (tenured or tenure-track) in the Department of Advanced Science and Technology, Faculty of Engineering.

The successful candidate is expected to conduct not only quality research and education in the field of information science and technology specified below, but also to promote cooperative research and educational programs in collaboration with our sister institute, Toyota Technological Institute at Chicago that focuses on fundamental computer science.

Research field: Intelligent information processing including artificial intelligence and its application, communication and network systems, computer vision, cyber-physical systems, human machine interface.

Qualifications: The successful candidate must have a Ph.D degree (or equivalent), a record of outstanding research achievements, and the ability to conduct strong research programs in the specified area. The candidate is expected to teach mathematics and programming at the introductory level and machine learning, information theory and signal processing at the advanced level. The supervision of undergraduate and graduate students in their research programs is also required.

Starting date: October 2016, or at the earliest convenience

Documents:

- (1) Curriculum vitae
- (2) List of publications
- (3) Copies of 5 representative publications
- (4) Description of major accomplishments and future plans for research activities and education (3 pages)
- (5) Names of two references with e-mail addresses and phone numbers
- (6) Application form available from our website (<http://www.toyota-ti.ac.jp/english/employment/index.html>)

Deadline: April 15, 2016

Inquiry: Search Committee Chair,
Professor Tatsuo Narikiyo
(Tel) +81-52-809-1816,
(E-mail) n-tatsuo@toyota-ti.ac.jp

The above should be sent to:

Mr. Takashi Hirato
Administration Division
Toyota Technological Institute
2-12-1, Hisakata, Tempaku-ku
Nagoya, 468-8511 Japan
(Please write "Application for Intelligent Information Processing Position" in red on the envelope.)



ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to acmm mediasales@acm.org. Please include text, and indicate the issue/or issues where the ad will appear, and a contact name and number.

Estimates: An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. NO PROOFS can be sent. Classified line ads are NOT commissionable.

Rates: \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. The MINIMUM is six lines.

Deadlines: 20th of the month/2 months prior to issue date. For latest deadline info, please contact: acmm mediasales@acm.org

Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at: <http://jobs.acm.org>

Ads are listed for a period of 30 days.

For More Information Contact:
ACM Media Sales
at 212-626-0686 or
acmm mediasales@acm.org



Faculty positions in Electrical and Computer Engineering in Africa

The College of Engineering at Carnegie Mellon University, a world leader in information and communication technology, has extended its global reach into Africa. In 2012 we became the first U.S.-based research university offering on-site master's degrees in Africa at our base in Kigali, Rwanda.

Carnegie Mellon University in Rwanda is educating future leaders who will use their hands-on, experiential learning to advance technology innovation and grow the businesses that will transform Africa.

We are seeking highly qualified candidates to join our world-class faculty, who share in our vision of developing creative and technically strong engineers that will impact society. Faculty members are expected to collaborate with industry and deliver innovative, interdisciplinary graduate teaching and research programs.

Carnegie Mellon is seeking exceptional candidates who can deliver innovative, interdisciplinary graduate programs in these areas:

- Software engineering
- Mobile and cloud computing
- Communications and wireless networking
- Cybersecurity and privacy
- Embedded systems
- Energy systems
- Image and signal processing
- Data analytics
- Applications in healthcare, agriculture, finance and infrastructure
- Innovation and technology management

Candidates should possess a Ph.D. in a related discipline and an outstanding record in research, teaching and leadership.

-
- Please contact us at info@rwanda.cmu.edu for full application requirements.
 - Further information about CMU in Rwanda can be found at www.cmu.edu/rwanda.
 - Applications should be submitted by email to director@rwanda.cmu.edu.

Carnegie Mellon University
Rwanda

[CONTINUED FROM P. 120] **You also still teach undergraduate 101-level computing.**

I have for 50+ years. The course has, of course, morphed over the years, but it still gives me great pleasure to turn newbies on to the field.

You and your students, working with Ted Nelson, designed one of the first hypertext systems, HES, in the late 1960s. I understand you're now working on your seventh hypermedia system.

The idea is to let you gather information from a variety of sources—the Web, PowerPoint decks, Excel spreadsheets, Word documents—and live-link them to extracts on your unbounded 2D work space. You can group things, name those groups, hyperlink between notes and documents (where “hyperlinks” are first-class objects with metadata), hyperlink between hyperlinks, annotate the notes, documents, and hyperlinks. It's a very rich system for gathering and organizing information. The part we're going to start working on soon is figuring out how to crawl through all that data to make some sort of linear or even branching narratives. Prezi provides a simple example of what we have in mind.

You also have done a lot work in so-called post-WIMP (windows, icons, menus, and pointer devices) user interfaces, trying to create ways to go beyond the standard keyboard-and-mouse paradigm.

The WIMP user interface has many limitations. It typically isn't driven by speech, though there are now multiple ways of using speech to input raw text. It's two-dimensional, and there are many situations in which you simply don't walk around with a keyboard and mouse; for example, virtual and augmented reality.

Most people today think of what lies beyond WIMP as touch.

That's absolutely one of the powerful addenda you can have for a WIMP GUI. But if you look at how you use your smartphone, the finger is in many cases just a substitute for the mouse. You're still clicking on targets to select them, or using swipe, flick, and pinch-zoom gestures. So this by-

now “universal” gesture vocabulary is very limited.

But people are capable of learning dozens of gestures.

One of the earliest things our group did when tablet PCs came out around 2000 is create a scribble gesture. I don't even have to tell you how to do it—you scribble over something and that deletes it. We had a gesture for lassoing things, which is now available on some WIMP interfaces. Undo-redo is a simple back-and-forth gesture, etc.

But the dream of all of us is that you should not just use your hands but your voice, and the system should know where you're looking, and it should know much more about your intent and your overall context. MIT's Architecture Machine Group, a forerunner of the Media Lab, produced a video called “Put That There” of a system in the late '70s that was as inspirational as Sketchpad was for showing how such smart, multimodal UIs could work.

What are some of the applications for the interfaces you and your research group have built?

Most are educational, and all were designed by students and other researchers in my group. The Music Notepad lets you draw notes and play them back through a MIDI-driven synthesizer. MathPad lets you handwrite mathematics and manipulate and solve equations, draw diagrams, create simple 2D animations to show the workings of the system of equations, and so forth. With ChemPad, you can draw two-dimensional molecule diagrams and it will turn them into three-dimensional ball-and-stick diagrams that you can tumble and get various kinds of information about.

We're still working on various sketching applications. In fact, one of our sponsors, Adobe, has decided to include our latest sketching program, called Shaper, as a plugin for Adobe Illustrator.

Another application your group created is the Touch Art Gallery, or TAG.

TAG is a platform for inputting artworks digitized at high resolution and letting users explore and annotate them with familiar touch and pen (for precise drawing) gestures. From the

beginning, we specialized in handling the largest possible artworks, including what we believe to be the largest artwork ever digitized, the AIDS Memorial Quilt, which is 22 acres in size. It's been exhibited in pieces, but it's too big to really take in. Through TAG, you can use large touch displays and go all the way from an overview, where the screen is filled with almost indistinguishably small tiles, to zooming in on the details of an individual panel, which is the size of a grave.

Each panel contains not just fabric, but objects and mementos like photographs, toys, souvenirs ...

When you're dealing with something that is loaded with emotional meaning, your ability to interact with it not just visually but tactilely is very important. People want to touch art, and they can't in a museum; they can't even get close enough to really see the details. TAG is currently being used in an exhibition in Singapore by the Nobel Foundation, featuring the terms of Alfred Nobel's will, interactive tours of his life, his associates, and his factories and houses and, of course, a gallery of all 900 (Nobel) laureates.

In a sense, this work brings you back to your original interests in computer-driven displays and their use in human-computer interaction.

Indeed. I was always interested less in hardware and software than in the interaction. One of my earliest published papers, in 1966, was “Computer Driven Displays and their Use in Man-Machine Interaction.”

In France, computer science is called *informatique*. The emphasis is not on the computer, which is a machine, after all, but about what you do with the computer, which is manage and visualize information. It's been a fantastic journey to see computer graphics evolve from an arcane technology accessible to a handful of specialists to completely integral to the world's daily consumption and production of information, communication, entertainment and, increasingly, education. 

Leah Hoffmann is a technology writer based in Piermont, NY.

© 2016 ACM 0001-0782/16/03 \$15.00

Q&A

A Graphics and Hypertext Innovator

Andries van Dam on interfaces, interaction, and why he still teaches undergraduates.

ANDY VAN DAM has been on the faculty at Brown University for more than 50 years. A committed mentor and educator, he co-founded the university's computer science department and served as its first chairman; he still teaches undergraduates. His research has been formative to the field of interactive computer graphics—from the Hypertext Editing System (or HES, co-designed with Ted Nelson), which used interactive displays to create and visualize hypertext, to *Fundamentals of Computer Graphics*, a book he co-wrote with James Foley that later became the widely used reference book *Computer Graphics: Principles and Practice*.

You got your Ph.D.—one of the first formal computer science Ph.D.'s ever awarded—at the University of Pennsylvania.

I'd gone to Penn to do electronics engineering. The year I entered, the engineering school launched a new track in computer and information science. My officemate, Richard Wexelblat, and I took a course from Robert McNaughton, who was what we'd now call a theoretical computer scientist. It had a little bit of everything—from programming to automata theory. I fell in love, and decided to enter the new track.

How did you get into graphics?

I saw Ivan Sutherland's still-great movie about Sketchpad, which is one of the top half-dozen Ph.D. disserta-



tions in terms of impact it had—seeing it changed my life.

You are referring to the film that showcased the groundbreaking computer program Ivan Sutherland wrote in 1963 for his dissertation at the Massachusetts Institute of Technology (MIT)'s Lincoln Labs.

Exactly. This was the era of mainframes and the beginning of minicomputers, and Sketchpad introduced two important innovations.

First was interactivity. In those days, you had to use a keypunch to make a deck of 80-column punch cards or use a Teletype to create paper tape, and then feed them to the computer. When your task was run, the computer would grind, and then

eventually print something on fan-fold striped paper, typically that your program bombed, followed by a memory dump.

And instead of this painfully slow cycle of submit/resubmit, where you could get one or two runs a day on a mainframe serving an entire organization like a university, here's this guy sitting at an interactive display, manipulating what looks a bit like an organ console with lots of buttons, dials, and switches. With his left hand, he's playing on a panel of buttons, and with his right hand he's manipulating a light pen, and it looks he's drawing directly on the screen. With the push of a button, he causes the rough drawing to be straightened out in front of your eyes. He designs a circuit or a mechanical drawing in a matter of minutes. Parts can be replicated instantly. And that's the second important innovation: he is manipulating graphical diagrams directly instead of having to work through code and coordinates. I was awestruck.

After you got your Ph.D., in 1965, you went to Brown, where you have been ever since.

Brown has been my home for more than 50 years, in no small part because of its emphasis on undergraduate teaching. I'm grateful I'm in this field that's still booming, where students can get fascinating jobs and have significant impact. [CONTINUED ON P. 119]



Computing Reviews

20th ANNUAL BEST OF COMPUTING

Best Reviews Notable Books & Articles

The 20th Annual Best of Computing presents Computing Reviews' best reviews and the most interesting books and articles of the year.

The Best Reviews highlight the excellent contributions of the reviewers. Based on the recommendations from our reviewers, category editors, editors in chief of journals, and others in the computing community, the Notable Books & Articles list brings together influential items published in computing.

April 2016 Online & Print



Association for
Computing Machinery

ThinkLoud

www.computingreviews.com



fuse **DIS 2016**

Designing Interactive Systems

June 4 – 8

Brisbane Australia

 dis2016.org

 bit.ly/dis16

 [@DIS2016](https://twitter.com/DIS2016)



Association for
Computing Machinery



Queensland University of Technology
Brisbane Australia