

COMMUNICATIONS

CACM.ACM.ORG

02/2014 VOL.57 NO.02

ACM



Mars Code

Ready Technology

A New Type of
Mathematics?

Node at LinkedIn

Computation Takes Time,
But How Much?

Cognitive Implants

Association for
Computing Machinery





2014

ACM INTERNATIONAL CONFERENCE
ON INTERACTIVE EXPERIENCES FOR
TELEVISION AND ONLINE VIDEO

25-27 JUNE, 2014
NEWCASTLE UPON TYNE
UK

Paper Submissions by
3 February 2014

Workshop, Demo, WIP
DC, Grand Challenge
& Industrial
Submissions by
31 March 2014

Welcoming Submissions on
Content Production

Systems & Infrastructures

Devices & Interaction Techniques

Experience Design & Evaluation

Media Studies

Data Science & Recommendations

Business Models & Marketing

Innovative Concepts & Media Art

TVX2014.COM

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>**

Departments

- 5 **Letter from Chair of Education Board
Education, Always**
By Andrew McGettrick

- 7 **Letter from the President
Cognitive Implants**
By Vinton G. Cerf

- 9 **Letters to the Editor
Contribute More Than
Algorithmic Speculation**

- 10 **BLOG@CACM
Clarifying Human-Computer
Interaction**
Philip Guo teaches an undergrad
through the use of examples.

- 27 **Calendar**

- 115 **Careers**

Last Byte

- 120 **Puzzled
Lowest Number Wins**
By Peter Winkler

News



- 13 **A New Type of Mathematics?**
New discoveries expand
the scope of computer-assisted
proofs of theorems.
By Don Monroe

- 16 **Should Everybody Learn to Code?**
Not everyone needs coding skills,
but learning how to think like
a programmer can be useful in
many disciplines.
By Esther Shein

- 19 **Computational Photography
Comes into Focus**
Advances in computational
photography are making image
capture the starting point. The
technology is transforming the field.
By Samuel Greengard

- 22 **ACM Fellows Inducted**

Viewpoints

- 24 **Privacy and Security
Would Cybersecurity
Professionalization Help Address
the Cybersecurity Crisis?**
Evaluating the trade-offs involved
in cybersecurity professionalization.
*By Diana L. Burley, Jon Eisenberg,
and Seymour E. Goodman*

- 28 **Education
Establishing a Nationwide
CS Curriculum in
New Zealand High Schools**
Providing students, teachers,
and parents with a better
understanding of computer science
and programming.
By Tim Bell

- 31 **Inside Risks
An Integrated Approach
to Safety and Security Based
on Systems Theory**
Applying a more powerful new
safety methodology to security risks.
*By William Young and
Nancy G. Leveson*

- 36 **Code Vicious
Bugs and Bragging Rights**
It is not always size that matters.
By George V. Neville-Neil

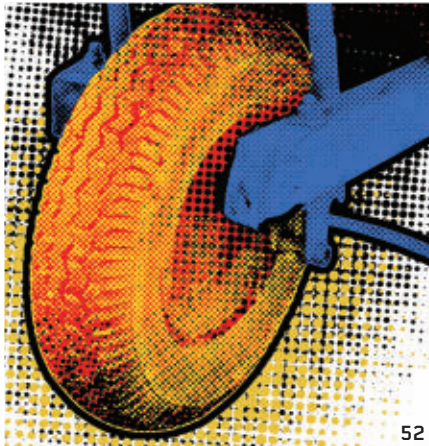
- 38 **Economic and Business Dimensions
Digital Platforms: When Is
Participation Valuable?**
Assessing the benefits and
challenges of knowledge spillovers.
*By Marco Ceccagnoli, Chris Forman,
Peng Huang, and D.J. Wu*

- 40 **Viewpoint
Ready Technology**
Fast-tracking emerging
business technologies.
By Stephen J. Andriole



Association for Computing Machinery
Advancing Computing as a Science & Profession

Practice



52

44 **Node at LinkedIn: The Pursuit of Thinner, Lighter, Faster**

A discussion with Kiran Prasad, Kelly Norton, and Terry Coatta.

52 **Center Wheel for Success**

“Not invented here” syndrome is not unique to the IT world.

By Poul-Henning Kamp

55 **Provenance in Sensor Data Management**

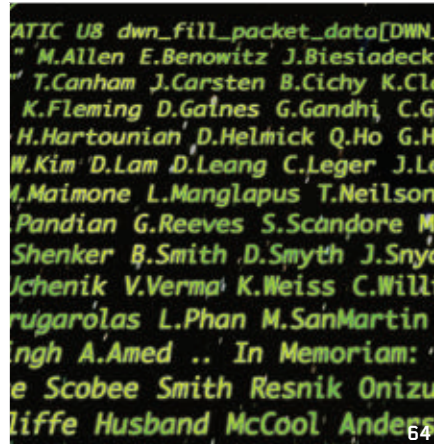
A cohesive, independent solution for bringing provenance to scientific research.

By Zachary Hensley, Jibonananda Sanyal, and Joshua New



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

Contributed Articles



64

64 **Mars Code**

Redundant software (and hardware) ensured Curiosity reached its destination and functioned as its designers intended.

By Gerard J. Holzmann

74 **Automatic Exploit Generation**

The idea is to identify security-critical software bugs so they can be fixed first.

By Thanassis Avgerinos, Sang Kil Cha, Alexandre Rebert, Edward J. Schwartz, Maverick Woo, and David Brumley

85 **Cryptography Miracles, Secure Auctions, Matching Problem Verification**

A solution to the persistent problem of preventing collusion in Vickrey auctions.

By Silvio Micali and Michael O. Rabin

**About the Cover:**

When the Curiosity rover landed on Mars in August 2012, it was a momentous feat for NASA and, in particular, the team responsible for the software that continues to control the rover some 350 million miles from Earth. This month's cover pays tribute to the unsung heroes of that landmark effort—the members

of the Flight Software Team. Look closely, you should be able to read their names. Cover illustration by Brian Greenberg/Andrij Borys Associates.

Review Articles



94

94 **Computation Takes Time, But How Much?**

Timing analysis for hard real-time systems.

By Reinhard Wilhelm and Daniel Grund

Research Highlights

106 **Technical Perspective****A New Spin on an Old Algorithm**

By Michael W. Mahoney

107 **Communication Costs of Strassen's Matrix Multiplication**

By Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz



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

John White

Deputy Executive Director and COO

Patricia Ryan

Director, Office of Information Systems

Wayne Graves

Director, Office of Financial Services

Russell Harris

Director, Office of SIG Services

Donna Cappel

Director, Office of Publications

Bernard Rous

Director, Office of Group Publishing

Scott E. Delman

ACM COUNCIL

President

Vinton G. Cerf

Vice-President

Alexander L. Wolf

Secretary/Treasurer

Vicki L. Hanson

Past President

Alain Chesnais

Chair, SGB Board

Erik Altman

Co-Chairs, Publications Board

Jack Davidson and Joseph Konstan

Members-at-Large

Eric Allman; Ricardo Baeza-Yates;

Radia Perlman; Mary Lou Soffa;

Eugene Spafford

SGB Council Representatives

Brent Hailpern; Andrew Sears;

David Wood

BOARD CHAIRS

Education Board

Andrew McGettrick

Practitioners Board

Stephen Bourne

REGIONAL COUNCIL CHAIRS

ACM Europe Council

Fabrizio Gagliardi

ACM India Council

Anand S. Deshpande, PJ Narayanan

ACM China Council

Jianguang Sun

PUBLICATIONS BOARD

Co-Chairs

Jack Davidson; Joseph Konstan

Board Members

Ronald F. Boisvert; Marie-Paule Cani;

Nikil Dutt; Roch Guerrin; Carol Hutchins;

Patrick Madden; Catherine McGeoch;

M. Tamer Ozsu; Mary Lou Soffa

ACM U.S. Public Policy Office

Cameron Wilson, Director

1828 L Street, N.W., Suite 800

Washington, DC 20036 USA

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

Computer Science Teachers Association

Chris Stephenson,

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

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

Editorial Assistant

Zarina Strakhan

Rights and Permissions

Deborah Cotton

Art Director

Andrij Borys

Associate Art Director

Margaret Gray

Assistant Art Directors

Mia Angelica Balaquiot

Brian Greenberg

Production Manager

Lynn D'Addesio

Director of Media Sales

Jennifer Ruzicka

Public Relations Coordinator

Virginia Gold

Publications Assistant

Emily Williams

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

acmhlp@cacm.org

Letters to the Editor

letters@cacm.acm.org

WEBSITE

<http://cacm.acm.org>

AUTHOR GUIDELINES

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

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 acmmidasales@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

eic@cacm.acm.org

NEWS

Co-Chairs

Marc Najork and William Pulleyblank

Board Members

Hsiao-Wuen Hon; Mei Kobayashi;

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;

Seymour Goodman; 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-Chairs

Stephen Bourne and George Neville-Neil

Board Members

Eric Allman; Charles Beeler; Bryan Cantrill;

Terry Coatta; Stuart Feldman; Benjamin Fried;

Pat Hanrahan; Tom Limoncelli;

Marshall Kirk McKusick; Erik Meijer;

Theo Schlossnagle; Jim Waldo

The Practice section of the CACM

Editorial Board also serves as

the Editorial Board of *ACM Queue*.

CONTRIBUTED ARTICLES

Co-Chairs

Al Aho and Georg Gottlob

Board Members

William Aiello; Robert Austin; Elisa Bertino;

Gilles Brassard; Kim Bruce; Alan Bundy;

Peter Buneman; Erran Carmel;

Andrew Chien; Peter Druschel;

Carlo Ghezzi; Carl Gutwin; James Larus;

Igor Markov; Gail C. Murphy; Shree Nayar;

Bernhard Nebel; Lionel M. Ni;

Sriram Rajamani; Marie-Christine Rousset;

Avi Rubin; Krishan Sabnani;

Fred B. Schneider; Abigail Sellen;

Ron Shamir; Yoav Shoham; Marc Snir;

Larry Snyder; Manuela Veloso;

Michael Vitale; Wolfgang Wahlster;

Hannes Werthner; Andy Chi-Chih Yao

RESEARCH HIGHLIGHTS

Co-Chairs

Azer Bestavros and Gregory Morrisett

Board Members

Martin Abadi; Sanjeev Arora; Dan Boneh;

Andrei Broder; Stuart K. Card; Jon Crowcroft;

Alon Halevy; Maurice Herlihy; Norm Jouppi;

Andrew B. Kahng; Xavier Leroy;

Mendel Rosenblum; David Salesin;

Guy Steele, Jr.; David Wagner;

Margaret H. Wright

WEB

Chair

James Landay

Board Members

Gene Golovchinsky; Marti Hearst;

Jason I. Hong; Jeff Johnson;

Wendy E. Mackay

ACM Copyright Notice

Copyright © 2014 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 \$100.

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 acmhlp@cacm.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



DOI:10.1145/2534706.2534707

Andrew McGettrick

Education, Always

In a recent issue of *Communications*, ACM President Vinton Cerf gave an excellent account of what ACM is doing to help reform K–12 education (August 2013, p. 7).

Moreover, he stressed ACM's position about the fundamental importance of computer science, that it should be regarded as a science on par with other sciences. Education initiatives are having a great impact within the U.S., but beyond the U.S. shores, others are watching and learning.

ACM's Education Board promotes computer science education at all levels and in all ways possible. In October, ACM Council approved the publication of the CS 2013 report—an exhaustive 10-year effort championed by ACM's Education Board and IEEE-Computer Society. The curriculum presents many new features, including an outward-facing view of the discipline, facilitating links with multidisciplinary work. It draws attention to the different platforms on which software resides and places considerable emphasis on security. Information Assurance and Security is deemed a new “knowledge area;” moreover, security considerations are to be embedded within the teaching of programming, software development, the human-computer interface activities, databases, networking, and other topic areas to better prepare students for the future.

CS 2013 is the latest in a series of curriculum guidance documents on computer science championed by both organizations; the respective co-chairs were Mehran Sahami from ACM and Steve Roach from the IEEE-CS. The report runs over 500 pages, largely because of considerable efforts invested in providing guidance (often in the

form of course exemplars) to a wide variety of interested parties. To view the full report, visit <http://www.acm.org/education/curricula-recommendations/>.

ACM's Education Board also responded to a recent request from the National Science Foundation to address how to best direct institutions of higher education on cyber security education as well as how to promote the need to incorporate this track into their courses. A report presenting the Board's suggestions is now available on ACM's Educational Activities website (<http://www.acm.org/education>).

Massive Open Online Courses (MOOCs) have been a major topic of discussion. Some commentators see these courses as having the capacity to bring about radical change to educational processes, whereas others hold a more conservative view. The Education Board, having a great interest in MOOCs, is sponsoring the first Learning at Scale (<http://learningatscale.acm.org>) conference to address MOOCs-related research issues and to help clarify and establish ACM's position on online learning. This conference is slated to take place in Atlanta in March 2014, just prior to and co-located with SIGCSE 2014.

As we look toward, and prepare for, the future of computing education, I am reminded of Doug Englebart's 1962 paper “Augmenting Human Intelligence: A Conceptual Framework.” There he puts forward the view that computing has a vital role to play, not in making users more intelligent, but in supporting their thinking and their analysis of problems. A subsequent pa-

per draws attention to the bootstrapping implications of this approach. In the research world disciplines like mathematics, physics, chemistry, medicine, and the humanities are all benefitting from great computational power, sophisticated modeling, and advances in data science. But related computational thinking needs to be woven more delicately into the fabric of general education so that students become more effective and efficient learners. Through this approach, all disciplines will benefit.

With information being readily available anywhere and at any time, and with great computing power also immediately accessible, education must respond and change. On current evidence, part of the solution is to pay far greater attention to interactive computing and graphics, simulation and modeling, search, analytics, and machine intelligence (feeding into business intelligence, among others).

Education is an ever-evolving domain, and ACM's Education Board has always been at the forefront leading the charge for change. Our goal is to best prepare future generations for a truly digital world. Computer education is the entryway to that world, and it is our job to make sure that door is never closed. 

Andrew McGettrick (andrew.mcgettrick@strath.ac.uk) is Professor Emeritus at the University of Strathclyde, Glasgow, Scotland, U.K. and the chair of ACM's Education Board and Education Council.

Copyright help by owners/author(s).



SCIENCE OF SECURITY

2013 Cybersecurity Paper Competition

Eligibility: Any paper that exhibits **outstanding contribution** to cybersecurity science and published from **Oct 1, 2012 - Dec 31, 2013**.

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.

Nominations close March 31, 2014.

Submit nominations to
<http://cps-vo.org/group/sos/papercompetition>

Hosted by the Research Directorate, National Security Agency



NSA

Rise Above the Ordinary

A career at NSA is no ordinary job. It's a profession dedicated to identifying and defending against threats to our nation. It's a dynamic career filled with challenging and highly rewarding work that you can't do anywhere else but NSA.

You, too, can rise above the ordinary. Whether it's producing valuable foreign intelligence or preventing foreign adversaries from accessing sensitive or classified national security information, you can help protect the nation by putting your intelligence to work.

NSA offers a variety of career fields, paid internships, co-op and scholarship opportunities.

Learn more about NSA and how your career can make a difference for us all.

**Excellent Career Opportunities
in Computer Science, Computer
Engineering and Related Fields**

KNOWINGMATTERS



Search NSA to Download



NSA
www.NSA.gov/Careers

APPLY TODAY

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

WHERE INTELLIGENCE GOES TO WORK®



Vinton G. Cerf

DOI:10.1145/2563407

Cognitive Implants

We are already well into the second month of 2014 and well on our way to the middle of the second decade of the 21st century. One hundred years ago, World War I was about

to start. Einstein's "annus mirabilis" papers were just nine years in the past. The first computers were about 25 years ahead, counting Conrad Zuse's 1938–1939 et. seq. work on the Z1 and Z2, especially, as seminal. Some 50 years ago—1964—marked the introduction of the IBM 360 computer. Roughly 40 years ago, the first paper on the Internet's core Transmission Control Protocol was published, the first hand-held mobile was being prototyped, and the Ethernet was invented. About 30 years ago the Internet was formally launched into operation and Apple announced the Macintosh. Circa 25 years ago, the World Wide Web was invented, the Mosaic Browser appears, and the so-called dot-com boom is poised to take off.

Every time I see calendar dates like 2014, I feel as if I have been transported by time machine into the future. It could not possibly be 2014 already! Isaac Asimov made some remarkably astute projections about 2014 in 1964,^a so what might he say today?

What we can reasonably see today is the emergence of a crude form of cognitive accessory that augments our remarkable, but in some ways limited, ability to think, analyze, evaluate, and remember. Just as readily available calculators seem to have eroded our ability to perform manual calculations, search engines have tended to become substitutes for basic human memory. The search engines of the Internet have become the moral equivalent of cogni-

tive implants. When I cannot think of someone's name or a fact (an increasingly common phenomenon), I find myself searching my email or just looking things up on the World Wide Web.

In effect, the Web is behaving like a big accessory that I use as if it were just a brain implant. Maybe by 2064 I will be able to access information just by thinking about it. Current mobiles, laptops, tablets, and Google Glass have audio interfaces that allow a user to voice requests for information and to cause transactions to take place. Whether we ever actually have the ability to connect our brains in some direct way to the Internet, it is clear we are fast approaching the ability to outfit computers (think "robots") with the ability to know about, perceive, and interact with the physical world.

It has been speculated that machine intelligence and adaptive programming will be the avenue through which computers will become increasingly cognizant of the world around them—increasingly behaving like self-aware systems. In addition to so-called "cyber-physical systems" that provide sensory input to computers and are expected to interact with the real world, an increasing degree of augmentation of our human sensory and cognitive capacity seems predictable. While we joke about memory upgrades or implants, search engines and the content of the Internet and World Wide Web act like exabyte memories that are reached through direct interaction with the computers that house them. Ray Kurzweil's virtuous, exponential

computing functionality and capacity growth predictions, even if overly bold in the short term, strike me as potential underestimates of what may be possible in 50 to 100 years.

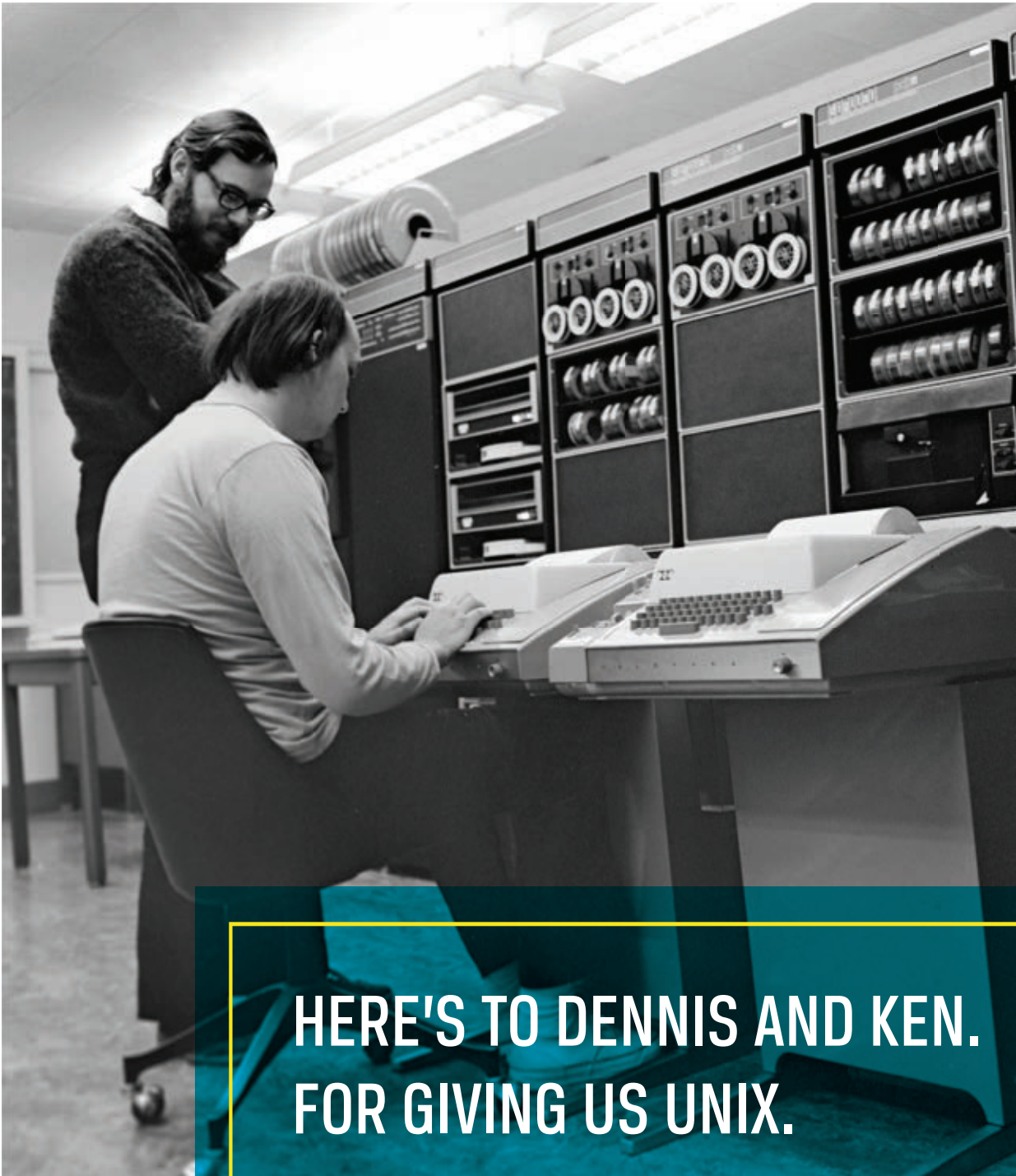
When we are on the cusp of generating an Internet of Things, humanoid and functional robots, smart cities, smart dwellings, and smart vehicles, to say nothing of instrumented and augmented bodies, it does not seem excessive to suggest the world of 2064 will be as far beyond imagining as 2014 was in 1964, except that Asimov had a remarkably clairvoyant view of what 50 years of engineering and discovery could achieve. A huge challenge will be to understand and characterize the level of complexity of such a world in which many billions of devices are interacting with one another often in unplanned ways.

For those of us who were around in 1964, we may recall our naïve aspirations for the decades ahead and realize how ambitious our expectations were. On the other hand, what is commonplace in 2014 would have been economically unthinkable 50 years ago. So perhaps exabyte, cognitive implants are a trifle ambitious in the short term, but a lot can happen in 50 years time. Just as we have adapted to the past 50 years, I expect we will rapidly embrace some of the functionality coming in the next five decades. It is already difficult to remember how we lived our lives without mobiles and the Internet. Now, where did I put that time machine?

Vinton G. Cerf, ACM PRESIDENT

Copyright held by Author.

^a <http://www.newsmax.com/SciTech/isaac-asimov-predictions/2014/01/06/id/545487>



HERE'S TO DENNIS AND KEN.
FOR GIVING US UNIX.

We're more than computational theorists, database managers, UX mavens, coders and developers. We're on a mission to solve tomorrow. ACM gives us the resources, the access and the tools to invent the future. Join ACM today and receive 25% off your first year of membership.

BE CREATIVE. STAY CONNECTED. KEEP INVENTING.

[ACM.org/KeepInventing](https://www.acm.org/KeepInventing)



Association for
Computing Machinery

DOI:10.1145/2556647.2556650

Contribute More Than Algorithmic Speculation

JACOB LOVELESS ET AL.'S article "Online Algorithms in High-Frequency Trading" (Oct. 2013) is an example of potentially valuable research misdirected. Ask any proponent of free-enterprise economics to explain its merits, and you will likely hear two themes: Profit motivates, and profit accrues by producing and selling valuable goods and services. The first buys the producer a bigger piece of the pie; the second increases the total size of the pie, thus raising, at least on average, the economic status of all. It works, most of the time, quite well.

Unfortunately, there are also many ways to profit while producing grossly inadequate, zero, or even negative economic value. Some of us are drawn to such schemes, so much so they work much more diligently at them than at a productive enterprise. To the extent this happens, free enterprise is undermined. Like printing counterfeit money, it works only if a minority does it, and even then, at the expense of everyone else.

Among the most serious such non-value-producing profit schemes is speculating in zero-sum derivative markets that produce no economic value at all, managing only to shuffle cash between winners and losers. Millisecond trading is just an escalation in vying for money this way. Even in financial markets like common stocks, where the original purpose is investment, and that do contribute to producing value, trading at sub-second time intervals is pure speculation or worse, as genuine investors could collectively be net losers to speculators. Putting effort into developing and using more successful speculation strategies is like going to a potluck dinner but bringing no food, just a bigger plate, while pushing more aggressively toward the front of the line.

Online and one-pass algorithm research can surely be redirected toward value-producing applications (such as robotics) where they can do

more than just seize profits at someone else's expense.

Rodney M. Bates, Strong City, KS

Put Thrills In Everyday Products, Too

As a user experience (UX) researcher, I took note of Steve Benford et al.'s article "Uncomfortable User Experience" (Sept. 2013) on designing discomfort into users' experience with technology. I appreciated Benford et al.'s interest in the framework of Freytag's pyramid and their examples of physical experience (such as amusement park rides and breathing exercises) but was left with questions about applying these ideas to the commercial HCI, particularly the UX, realm.

I venture to say the majority of UX designers reading *Communications* design hardware or software, not just for entertainment but for educational and productivity purposes. In any domain, UX designers are always looking for new interaction methods on mobile devices, ways to "gamify" tasks, or unique interactions that make their brands more desirable, popular, and memorable. For me, Benford et al. started down an interesting new path but stopped short of defining a clear link between these tactics and the kind of HCI work most developers do, which is probably more cognitive than physical. Could these tactics work for us?

For example, Benford et al. reminded us of interface innovator Ben Shneiderman's guideline that the locus of control should remain with the user, suggesting "distorting this relationship" would only generate discomfort. Moreover, Benford et al.'s examples were physical: thrill ride, walking tour, performance audience member. But this would seem to have been the perfect place to explore possibilities in everyday software development. If in your next mobile app project you wanted to build in a "thrill" for user sociality or enlightenment, how would it work?

Benford et al. certainly inspired unconventional thinking, but I was left wanting acknowledgment there is a

place for uncomfortable user experience in everyday products as well.

Elise Lind, Portland, OR

Survival vs. Reflection in Education

Karen A. Frenkel's news story "CS Enrollments Rise... at the Expense of the Humanities?" (Dec. 2013) reminded me why the trend toward computer science does not diminish the value of a well-rounded education or the humanities in general, even as it identified two aspects of the humanities making them less desirable than computing and IT in today's academic environment:

Bias. The humanities have become politicized to the point they often seem intended to put the agendas of tenured faculty or intellectual movement ahead of students' interests. Such bias plagues all traditional academic disciplines but is disproportionate in the humanities. Moreover, there is often no objective, measurable, or quantifiable way to assess opinions, short of a professor's publishing history, while schools of thought splinter into factions; see, for example, literary criticism; and

Employment. Getting a job with just a degree in the humanities, even in teaching, is a challenge. I know; as an undergrad I studied comparative French and German literature. Granted, humanities graduates may write well and make persuasive arguments, but so do IT workers and programmers. I fault academic institutions more than students for ignoring the employment implications of their programs, including the skills the economy demands and employers pay for; my college did not, for example, offer accounting...on ideological grounds. Humanities professors comfortable within their intellectual microcosms should reassess their role in today's academic climate and help their students learn the skills they need to create and survive, not just reflect.

Dimitri Darras, Sterling, VA

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

© 2014 ACM 0001-0782/14/02 \$15.00

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:10.1145/2557448

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

Clarifying Human-Computer Interaction

Philip Guo teaches an undergrad through the use of examples.



Philip Guo
Two Examples
of HCI Research

<http://cacm.acm.org/blogs/blog-cacm/163199-two-examples-of-hci-research/fulltext>

April 10, 2013

AN UNDERGRAD RECENTLY sent me the following message: “I was thinking today that I would like to learn more about what HCI research involves. Can you recommend any papers for me to read?”

I decided to follow Matt Might’s advice (at <http://matt.might.net/articles/how-to-blog-as-an-academic/>) and write a public blog post about this topic, rather than just replying privately to this student.

(Disclaimer: HCI is a *very* diverse field, so I obviously do not claim to speak for all HCI researchers. If you asked 10 randomly selected HCI researchers to write this post, you will get 10 different answers.)

What Is HCI Research?

To me, research in HCI (human-computer interaction) involves

- Understanding how humans interact with computers, and

- Creating new and effective ways for humans to interact with computers.

Here, the term “computer” can refer to a desktop machine, laptop, tablet, mobile phone, digital eyewear (<http://www.google.com/glass/start/>), or an assortment of other electronic devices; it can also refer to both software and hardware running on these devices.

Some HCI research involves doing science (such as understanding), while others are more focused on engineering (such as creating).

Two Examples of HCI Research

There is no way that I can do justice to the entire world of HCI in one blog post, so instead I will present two papers that exemplify some typical characteristics of modern HCI research.

The lead author on both papers is my colleague Joel Brandt (<http://www.joel-brandt.org/>), who performed this work while he was a Ph.D. student in the Stanford computer science department. At the time, Joel’s focus within HCI was on how programmers (humans!) interact with computer software used throughout the programming process (for example, IDEs (http://en.wikipedia.org/wiki/Integrated_development_environment), debuggers, Web browsers).

Paper 1

Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code (Brandt et al., CHI 2009, http://www.joelbrandt.org/publications/brandt_chi2009_programmer_web_use.pdf) was published at CHI 2009 (<http://www.chi2009.org/>), a notable academic conference for HCI research.

The research described by this paper is an example of “understanding how humans interact with computers.” Specifically, Joel and his colleagues sought to understand how programmers interact with digital resources found on the Web.

To do so, the research team performed two studies:

1. **Lab study:** They invited 20 programmers into a computer lab one at a time, gave each subject a two-hour-long programming task, and watched how the subject used Web resources while programming. Drawing from direct observations of these 20 subjects in a controlled lab setting, the team observed three main forms of interaction with Web resources—learning, clarification, and reminder—and described the unique aspects of each form in their paper.

2. **Query log analysis:** The team wanted to validate whether these observations generalize beyond their small and relatively homogeneous population of 20 lab subjects, who were all Stanford students. Working with industry colleagues at Adobe, they obtained a dataset containing over 100,000 queries made by over 24,000 programmers to a custom search engine for Adobe programming tools

(<http://www.adobe.com/devnet.html>). They parsed and analyzed the data to discover insights that supported observations from their prior lab study.

These two studies complement and reinforce one another. The first provides a great level of detail (direct human observation) but a small sample size ($N=20$). The second provides little detail (search queries) but a large sample size ($N=24,000$). By reading both studies in the paper, you can understand the relative strengths and weaknesses of each approach.

The findings presented by HCI studies such as the ones in this paper serve two roles: they contribute to the body of scientific knowledge about a form of human-computer interaction (for example, Web usage during programming), and they inspire researchers to create new kinds of tools to improve such interactions.

For example, the findings in this paper suggest ways that existing IDEs can be augmented to help programmers better leverage Web resources. These findings directly inspired Joel's next research project, which led to...

Paper 2

A year later, Joel published *Example-Centric Programming: Integrating Web Search into the Development Environment* (Brandt et al., CHI 2010, http://www.joelbrandt.org/publications/brandt_chi2010_example_centric_programming.pdf).

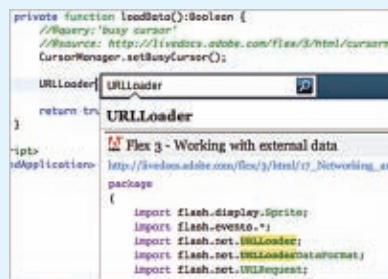
The research described by this paper is an example of “creating new and effective ways for humans to interact with computers.” Here, Joel and his colleagues sought to create a new and better way for programmers to use snippets of example code they find on the Web.

To do so, Joel spent a summer internship at Adobe building a plug-in for Adobe Flash Builder (<http://www.adobe.com/technology/projects/blueprint.html>), which embeds a domain-specific search engine within the IDE (see screenshot).

This system, called Blueprint, combines an IDE plugin and custom search engine to enable new kinds of user interactions, such as:

- ▶ Instant Web search without leaving the IDE's code editor,
- ▶ Browsing through search results that are automatically formatted in a “code-centric” format, which is more useful to programmers than plain Web pages,

Screenshot of a Blueprint code search.



▶ Fast copy and paste of retrieved example code snippets into the user's code base, and

▶ Links between the copied code and its source, to support notifications if the source gets updated.

The first half of his paper describes how Joel used insights from the studies in his prior paper to design the Blueprint system. The second half describes two studies the team ran to show that Blueprint was effective:

1. **User study:** They recruited 20 professional programmers at Adobe to perform a series of programming tasks in a controlled lab setting. They let half the participants use the Blueprint system (treatment group) and the other half use an ordinary Web browser (control group). They then compared the performance of participants in both groups on metrics such as time to complete each task and resulting code quality.

2. **Longitudinal study:** To understand how Blueprint is used in real-world settings, the team deployed the system to over 2,000 users over a three-month time span. They recorded 17,000 queries made by these users and analyzed the contents of those queries to discover insights that complemented their user study findings.

Finally, a customary way to end these sorts of papers is by discussing current limitations of the system and some ideas for future work.

Conclusion and Further Reading

These two papers formed the bulk of Joel's 2010 Ph.D. dissertation (http://www.joelbrandt.org/publications/brandt_2010_phd_dissertation.pdf). His research started in a university lab at Stanford, continued during summer internships at Adobe, and eventually turned into a feature within a

commercial software product (Blueprint) that thousands of people use on a daily basis. I like presenting this work because it is a good example of how HCI research can be done in both academia and industry, and can range from scientific studies to the development of practical tools.

Joel's work is just the tip of the iceberg, though. Besides studying the interaction between humans and computers, there is a lot of HCI research that explores how humans interact with one another via computers. For example, projects might involve:

▶ **Understanding** how programmers interact with one another on the Stack-Overflow (<http://stackoverflow.com/>) Q&A site (Mamykina et al., CHI 2011, <http://www.cs.berkeley.edu/~bjoern/papers/mamykina-stackoverflow-chi2011.pdf>), and

▶ **Creating** a mobile phone app called VizWiz (<http://vizwiz.org/>) that lets blind users quickly and effectively solicit help from strangers on the Internet (Bigham et al., UIST 2010, <http://www.cs.rochester.edu/hci/pubs/pdfs/vizwiz.pdf>).

Reading the four papers mentioned in this blog post will give you a sense of how HCI papers are structured. Enjoy!

Reader's comment:

HCI research: I ask 10 different researchers and get 15 different answers!

What about theoretical models for HCI; that may reduce the dimensions.

—Anonymous

Some nice points in this post, thanks. Responding to your anonymous commenter, HCI is indeed a diverse field that moves with the times, so as technologies change, so do some (but not all) of the research foci.

The SIGCHI Executive Committee has been looking into HCI as a field in a project focused on HCI Education. We have written an interim report which is accessible from our website (<http://www.sigchi.org/>). The results are also summarized on the Interactions website (see <http://interactions.acm.org/archive/view/march-april-2013/teaching-and-learning-human-computer-interaction>).

—Elizabeth Churchill

Philip Guo is a postdoctoral scholar in the Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory.

A horizontal banner with a solid orange background. The text "MOBILEHCI" is written in large, white, sans-serif capital letters. The year "2014" is written in smaller, red, sans-serif capital letters inside the second 'O' of "MOBILE". Below "MOBILEHCI", the text "TORONTO, CANADA" is written in red, sans-serif capital letters. A dark grey silhouette of the Toronto skyline is at the bottom, with the CN Tower being the most prominent feature, positioned behind the letter 'L' in "MOBILEHCI".

MOBILEHCI

2014

TORONTO, CANADA

>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<

16th International Conference On Human-Computer Interaction with Mobile Devices and Services

>>>>>>>>> Due Date: <<<<<<<<<<

**Full and Short Papers
Submissions:
February 21, 2014**

For more information please visit:
<http://mobilehci.acm.org/2014/>

On behalf of Canada's HCI community and mobile industries, OCAD University and the Graphics, Animation and New Media Network of Centres of Excellence (GRAND NCE) invite you to Toronto for MobileHCI 2014. We are preparing to host the MobileHCI 2014 conference with much excitement as this is the first time it will be held in Canada and only the second time that it is taking place in North America. We look forward to seeing you in 2014!



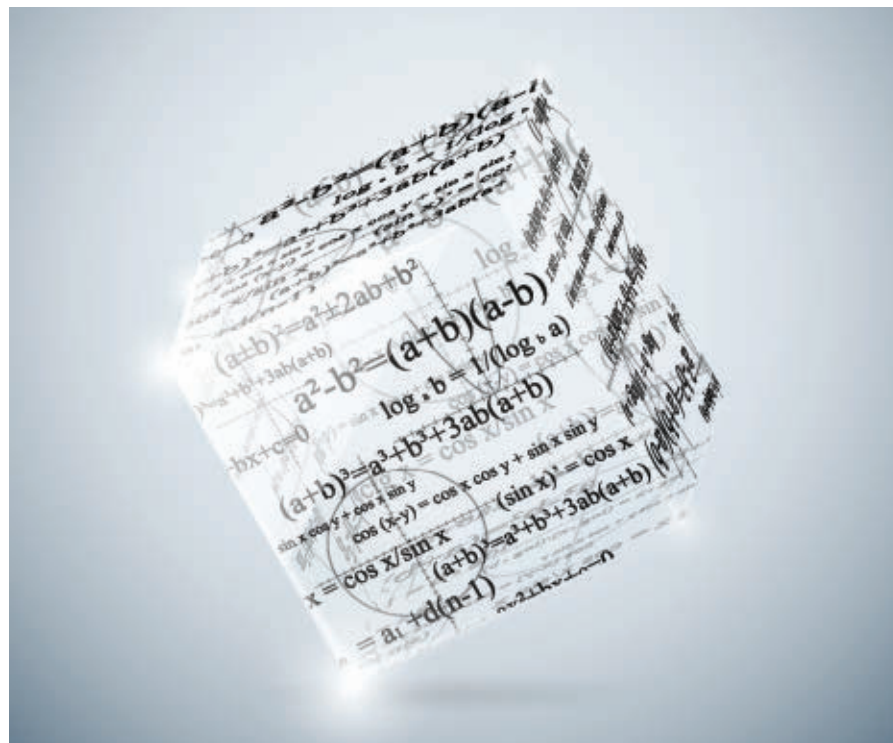
A New Type of Mathematics?

New discoveries expand the scope of computer-assisted proofs of theorems.

COMPUTERS HAVE TRANSFORMED a broad range of human activities, from sales to basic research. Now, for an enthusiastic contingent of mathematicians and computer scientists, they are poised to deliver on a long-standing promise to do the same for mathematics.

The renewed excitement grows from discoveries that expand the scope of computer-assisted proofs of theorems, but also provide a new and more intuitive way of grounding new results to the bedrock foundations of mathematics, even as those results grow more complex. Tools based on these developments could help establish a growing library of certified results backed by computer verification. Along the way, it could change the culture of mathematics by making it easier for individuals to dependably add to this growing edifice.

The most prominent face of the new movement is Vladimir Voevodsky, at the School of Mathematics of the storied Institute for Advanced Study (IAS) in Princeton, NJ. In 2002, Voevodsky shared the International Math Union's Fields Medal which, together with the Abel Prize, are often called the math-



ematics equivalent of the Nobel Prize. In recent years, he has turned his attention to exploring how computers can enhance mathematics research.

In the process, Voevodsky has inspired a high-powered group of mathematicians and computer scientists

to explore the implications of the new framework. Over the 2012–2013 academic year, several dozen of them assembled at IAS to work towards this goal.

“It was really thrilling,” said co-organizer Steve Awodey of Carnegie Mellon University in Pittsburgh, PA.

“You felt like you were part of the Manhattan Project of computer science. Everybody had the feeling they were part of something important, and I think they were. I think we really made some amazing progress there, the consequences of which are going to take time now to play out.”

In part to build on this momentum, some 25 of the researchers wrote a 600-page textbook (available free online at <http://homotopytypetheory.org/book/>) describing the new view, called Homotopy Type Theory (HoTT) for reasons discussed later. The book was prepared collaboratively over just a few months, as illustrated in the time-lapse video available at <http://vimeo.com/68761218>, sharing and editing documents using the GitHub version-control platform originally designed for code development.

This collaborative authorship is an intriguingly parallel to the group’s vision of a reliable and consistent encapsulation of a body of mathematical knowledge, analogous to a library of trusted subroutines in a computer program. This vision is an old one: more than a century ago, mathematicians strove to formalize all mathematics,

and trace it all back to a few axioms. That goal ran into trouble when it was proved that, for any formal system, it is impossible to prove all theorems about that system. As a result, “working mathematicians decided that foundations are irrelevant for their purposes,” says Awodey. Moreover, there was no particular payoff for tracing everyday work back to first principles. Although the new framework does not avoid these problems, Awodey said, it doesn’t much matter. “This system of foundations has a much more practical aspect; it is closer to the way mathematicians reason.”

“This system of foundations has a much more practical aspect; it is closer to the way mathematicians reason.”

The system could also enhance cooperation between mathematicians. If computer-verified proofs become mainstream, Voevodsky says, “it will eventually lead us to a possibility of big, collaborative projects.” If the computer can guarantee that a particular result is correct, others can build on it with confidence, even if it comes from an unknown or novice researcher, or an expert who is importing novel ideas into a different subfield.

Voevodsky stresses that human ingenuity will always be important. “It’s not like someone who doesn’t know anything in mathematics can just use this library and start producing great mathematics on top of it. One still has to have the internal [mental] representation of what’s going on there.”

Another advantage of a computer library of mathematics, says IAS participant Andrej Bauer of the University of Ljubljana, Slovenia, would be the ability to search for relevant results. Even if the new framework lives up to its promise, he cautions, “ultimately, it is not just the math; it is the question of how new mathematical ideas get adopted by the wider community of mathematicians.”

Milestones

Computer Science Honors

ACM, IEEE CHOOSE GOODMAN TO RECEIVE ECKERT-MAUCHLY AWARD

ACM and the IEEE Computer Society have named James R. Goodman the recipient of the Eckert-Mauchly Award in recognition of his contributions to the hardware/software interface of computer architecture.

The Eckert-Mauchly award is given annually in recognition of contributions to computer and digital systems architecture.

Goodman is currently a professor of computer science and a department chair at the University of Auckland, New Zealand.

His innovations led to the development of hybrid approaches to high-performance computer memory systems that can achieve nearly the performance of hardware but with the flexibility of software.

Principal co-inventor of hardware queue-based locks, which allow programs with busy-wait synchronization, also known as spinning, to scale to very large multiprocessors, Goodman also introduced critical section speculation, which helped launch the resurgence of transactional memory as a parallel programming and synchronization method.

Co-author of *A Programmer’s View of Computer Architecture*, a highly acclaimed book on computer architecture, with Karen Miller, and *Structured Computer Organization* with Andrew Tanenbaum, Goodman is a principal supervisor of 10 Ph.D. students, as well as a Fellow of both IEEE and ACM.

HANSON ELECTED FELLOW OF ROYAL SOCIETY OF EDINBURGH

Included among the more than

40 people elected Fellows of The Royal Society of Edinburgh (RSE) in 2013 was ACM Secretary/Treasurer Vicki L. Hanson, in recognition of her contributions to human computer interaction.

The RSE is Scotland’s national academy of science and letters.

Hanson is professor of Inclusive Technologies at the University of Dundee, and research staff member emeritus from IBM Research. Her research explores design issues related to inclusion, seeking to understand and address problems that create barriers to technology adoption and use by examining ways in which existing technology can be adapted to better support older adults and disabled users.

An active ACM member for more than 20 years, Hanson currently serves the organization’s Secretary/Treasurer, in addition to being

a member of the ACM-W Europe Executive Committee, and the founder and co-editor-in-chief of *ACM’s Transactions on Accessible Computing*. She is past chair of the ACM SIG Governing Board and of ACM SIGACCESS, and was named an ACM Fellow in 2004. In 2008, Hanson received the ACM SIGCHI Social Impact Award for the application of HCI research to pressing social needs.

Hanson also has been named a Fellow of the British Computer Society, and was the 2013 recipient of the Anita Borg Institute Woman of Vision Award for Social Impact. She received an IBM Corporate Award for Contributions to Accessibility, multiple IBM Outstanding Contribution Awards for her work in accessibility and education, the University of Oregon Arts and Sciences Alumni Fellows Award, and a Royal Society Wolfson Research Merit Award.

Indeed, for a century, mathematicians have considered set theory to be an adequate basis for formalizing all of mathematics. Starting with concepts like the null set (corresponding to zero) and the set containing only the null set (corresponding to one), one can, in principle, systematically construct all the objects of mathematics. In practice, however, the process is clunky and time-consuming—and therefore, rare. The proponents of HoTT hope it will provide easier and more intuitive tools that will allow rigorously formalized mathematics to become standard practice.

HoTT is based on a mathematical framework called type theory. Unlike sets, which are like bags that can contain various kinds of object, objects of a particular type have specific rules about how they can be manipulated. They are reminiscent of the data types that help enforce rigor in high-level programming languages, but the mathematical version of types can be more elaborate; for example, an n -dimensional vector whose precise character depends on a natural number n that must be computed.

A version of type theory is used in most versions of automated “proof assistants,” which have been growing in power since their introduction in the 1960s but are still not widely used in pure mathematics. This framework expands the notion of types so that, for example, the formulation of a theorem can itself be a type, and a proof of the theorem can be an object of that type; thus, if such an object even exists, the theorem is proved. These exotic types can even ensure that no logical cases are overlooked.

One of the best-known successes was the formal proof of the four-color map theorem in 2005 by Georges Gonthier of Microsoft Research in Cambridge, UK. (The earlier 1976 proof by Kenneth Appel and Wolfgang Haken of the University of Illinois at Urbana-Champaign had combined computer code for some parts with text arguments for others.) Gonthier used the proof assistant Coq, which in 2013 won the Programming Languages Software Award of the ACM Special Interest Group on Programming Languages (SIGPLAN).

This Coq flavor of type theory has good computational properties, in particular a feature called decidabil-

A version of type theory is used in most versions of automated “proof assistants,” which were introduced in the 1960s but are still not widely used in pure mathematics.

ity of equality. This means there is an algorithm to determine whether any two objects are equal, which is a key step in its use for proofs. Yet Thierry Coquand of the University of Gothenburg, Sweden, who was one of the developers of Coq, noted “there was always something missing” in the notion of equality in this type theory. “The concept of equal is better” in HoTT, he said. “Hopefully, it will lead to practical things.”

The new view of equality arises from the realization, arrived at independently by Voevodsky and by Awodey and his student Michael Warren, of a connection between completely different branches of mathematics. To give a more familiar example, the ancient understanding of circles, ellipses, hyperbolas, and parabolas as cross-sections is enriched and complemented by their description by algebraic equations. In a particular situation, one description or the other may be more useful, but their combination can lead to new insights and perhaps a glimpse of a larger reality. “This happens in mathematics over and over again,” says Bauer. “We’re discovering a new connection, and this new connection is now influencing both sides of the connection.”

Similarly, homotopy type theory represents a connection between type theory and homotopy theory, which is a branch of topology. In the homotopy view, the types (which can be theorems) are envisioned as spaces, and the objects of that type (which can be proofs) are points in the space. The equality

of two points can then be thought of as the existence of a path in the space connecting the two points. This connection allows a host of tools from homotopy theory to be applied the task of mathematical proof.

At a minimum, HoTT extends the domain of proof assistant to new areas of mathematics. “Coq was designed by computer scientists, so its initial area of application was basically programming language theory and combinatorics,” Gonthier noted. Many papers presented at SIGPLAN’s Principles of Programming Languages (POPL) symposium are backed by computer proofs, but this kind of problem “isn’t all of mathematics,” he says, and applying it to other areas “is quite important and interesting.”

Beyond applying logical reasoning, for example, to topological problems, Gonthier says, “the correspondence can be used in the reverse way, to try to deduce something about the logical system based on insights that have been developed for, basically, topology.”

“It’s definitely expanding our notion of what a proof can mean, because it’s explaining proofs in a geometric way,” says Bauer. “We know it’s going to bring us something new, but time will tell what.”

“Formalization of mathematics is somehow too compelling and effective to go away,” said Awodey. “It’s going to happen. It’s just a question of when and using what system,” whether HoTT or something else. □

Further Reading

The Univalent Foundations Program, Institute for Advanced Study
“Homotopy Type Theory: Univalent Foundations of Mathematics” (Princeton, 2013) available for download or purchase at <http://homotopytypetheory.org/book/>.

Georges Gonthier
“Formal Proof—The Four-Color Theorem”, *Notices of the American Mathematical Society* 55 (11): 1382–1393 (2008).

Thomas Hales
“Mathematics in the Age of the Turing Machine”, to be published in *Turing’s Legacy: Developments from Turing’s Ideas in Logic (Lecture Notes in Logic)*, ed. R. Downey, (Cambridge University Press, 2014)

Don Monroe is a science and technology writer based in Murray Hill, N.J.

© 2014 ACM 0001-0782/14/02 \$15.00

Should *Everybody* Learn to Code?

Not everyone needs coding skills, but learning how to think like a programmer can be useful in many disciplines.

TO GAUGE THE ability of professional graphic designers to do basic programming, Brian Dorn, then a graduate student at the Georgia Institute of Technology (Georgia Tech), asked a group of them to read and modify a piece of program code. The idea was to see whether they could turn themselves into informal programmers and figure out how to develop automated functions in Adobe Photoshop. Unfortunately, when the designers conducted Web searches to look for information on the code they needed, they sometimes used results that pointed them in the wrong direction, which was toward Java—when they actually needed to be using JavaScript for this particular project.

One of the underlying causes could have been tied to the participants' "lack of sufficient general, abstract knowledge of the computing and/or programming structures at play," wrote Dorn in *Communications* in May 2011.

His advisor, Mark Guzdial, who relayed the story, said the findings indicate to him "that there are a lot of people who need knowledge of computer science ... who are going to use it in their lives, but because they never learned anything about computer science, they are teaching it [to] themselves and coding inefficiently, and wasting a lot of time and getting frustrated."

If someone is going to become a knowledge worker, or take on any job "that requires an undergraduate degree," they should know how to read a piece of code that is useful to them and be able to make changes to it, says Guzdial, a professor and director of Contextualized Support for Learning in the School of Interactive Computing at Georgia Tech.

People ranging from former President Bill Clinton to Facebook creator



Second-grade students in Kevin Jarrett's Elementary 'STEMLAB' at Northfield Community School (New Jersey) participate in the 2013 Hour of Code.

Mark Zuckerberg to physicist, cosmologist, and author Stephen Hawking have expressed the belief that basic computer programming is an essential skill in today's world. "Code has become the 4th literacy. Everyone needs to know how our digital world works, not just engineers," says Mark Surman, executive director of the Mozilla Foundation, whose comments are among those of dozens of luminaries on code.org.

The demand for computer scientists and technical professionals in the U.S. is projected to grow 34% through 2018, according to the Bureau of Labor Statistics. Many people already engage in some level of programming; Guzdial cites a 2005 Carnegie Mellon University study indicating that in 2012 there would be 90 million workers in the U.S., more than 55 million of whom would use spreadsheets and databases, which can be deemed programming. The study also projected that more than 13

million would describe themselves as "programmers" in 2012, although only three million of them would be professional software developers.

The Carnegie Mellon study also noted that a lot of people were doing programming without realizing it, by creating macros for spreadsheets or doing database queries using SQL. "So the argument is, lots of people are going to do programming," says Guzdial, "and the data we have studying how end user programmers teach themselves and the types of mistakes they make suggest if they knew something about computer science, they might not have to struggle so much later."

Many people who avoided taking science and math courses in college are now struggling as they try to teach themselves how to program, he points out. "How many more would be doing some programming if we helped them? That is the interesting part."

What You Should Learn

Everyone should learn computational thinking, maintains Jeannette Wing, corporate vice president at Microsoft Research. Computational thinking helps people learn how to think abstractly and pull apart a problem into smaller pieces. One concrete way to learn aspects of those skills is programming, Wing says.

That is not to say everyone needs to learn a specific programming language like Python or C++, even though many people identify programming with turning out code, Wing says. “First of all, that is too low-level, and it is also very narrow an interpretation of what I believe is more important.”

Instead of teaching everyone to churn out code, the emphasis should be on learning problem-solving skills in computer science, much like the problem-solving skills one learns in math and engineering, says Wing, who is on leave as President’s Professor of Computer Science at Carnegie Mellon. Writing a program is an explicit way of expressing a solution that a human or machine can carry out, she says. “The more fundamental skill and more critical thinking skill is what comes before you write down this piece of code, and that is computational thinking.”

Guzdial agrees. “Should we learn enough so that you can write a script to do something that otherwise would have to be done by hand? I would like to see that, but I cannot make the argument that it is an absolute necessity.”

He adds that ignorance of computer science puts people at a disadvantage in today’s world. “Not knowing anything about programming makes it more difficult to pick it up.”

The Flip Side

The issue is far more black and white to software engineer Chase Felker, who wrote an article for *Slate* magazine entitled “Maybe Not Everybody Should Learn to Code.” Felker writes, “Frankly, just the idea that you can learn to code in a year gives me the creeps: I would be terrified if someone with only a couple of classes were writing programs for me, not because he (of course, and unfortunately, most programmers are men) has learned anything wrong—but because of what he doesn’t know.”

Computational thinking helps people learn how to think abstractly and pull a problem apart into smaller pieces.

While noting that several of his colleagues are successful self-taught programmers, and that learning to program does not necessarily have to be done at a university, Felker says people need to know more than memorizing the technology du jour and, as Wing said, they need the critical ability to think things through.

“[I]f you aren’t dreaming of becoming a programmer—and therefore planning to embark on a lengthy course of study, whether self-directed or formal—I can’t endorse learning to code,” Felker writes. “Yes, it is a creative endeavor. At its base, it’s problem-solving, and the rewards for exposing holes in your thinking and discovering elegant solutions are awesome.” He goes on to say he does not believe that most people who learn to code end up learning anything that stays with them.

Referencing a comment made by New York City Mayor Michael Bloomberg in 2012 that he would learn to code, programmer Jeff Atwood, writing in his blog “Code Horror,” poses the question, “...can you explain to me how Michael Bloomberg would be better at his day-to-day job of leading the largest city in the USA if he woke up one morning as a crack Java coder?” While agreeing that programming is important, Atwood says many other skills are important, too. “I would no more urge everyone to learn programming than I would urge everyone to learn plumbing,” he writes.

The so-called “everyone should learn to code” movement is wrong for several reasons, according to Atwood, including the assumption that more code in the world is an inherently desirable thing. That assumes code is the goal; it puts the method before the

ACM Member News

TECHNOLOGY, ALWAYS CHANGING, SHOULD BE EASY TO USE



The two guiding principles of Jeff Johnson’s 35-year career in human-computer interaction

(HCI) have been the constantly changing nature of technology, and that technology should be easy to use.

Johnson, president and principal consultant for product usability consultancy UI Wizards Inc., earned B.A. and Ph.D. degrees in psychology and computer science from Yale and Stanford universities, respectively. He has worked as a user-interface designer and implementer, engineer manager, usability tester, and researcher at Xerox, Hewlett-Packard Labs, and Sun Microsystems, and recently returned from a stint as a Visiting Erskine Fellow at the University of Canterbury, Christchurch, New Zealand. These experiences have solidified Johnson’s commitment to continually refresh his work to keep pace with the latest digital advances. “There never will be a time when everyone is a digital native, because the definition of that term changes as digital technology progresses,” he says.

Johnson, an ACM Distinguished Speaker, recently updated his 2009 HCI book *Designing with the Mind in Mind* with a new section on peripheral vision and chapters on decision-making and hand-eye coordination.

He also has a new corporate endeavor: Wiser Usability, a consultancy that helps companies design senior-friendly websites. “Older adults tend to have limited mobility and transportation, and they could benefit most from online shopping and online access to services,” Johnson says. “No one who isn’t a hardcore computer-geek, least of all seniors, wants to use technology for its own sake. Digital tools that don’t help seniors accomplish their goals with a minimum of learning and bother are not worth the time and expense,” he adds.

—Laura DiDio

problem, and assumes adding coders to the workforce is a net positive.

"The general populace (and its political leadership) could probably benefit most of all from a basic understanding of how computers, and the Internet, work," he says. "Being able to get around on the Internet is becoming a basic life skill, and we should be worried about fixing that first and most of all, before we start jumping all the way into code."

Guzdial speculates there may be pushback from programmers, because they think not everyone can be taught what they do. "I am not suggesting everyone produce thousands and thousands of lines of code. I would love if everyone could graduate from a university writing 10 lines of code that are useful to them."

The point of teaching programming in high school would be to give students some level of literacy relative to programming, including the ability to think about things in terms of code, and to understand what code can do, Guzdial adds.

The State of Computer Science in Public Education

That may not, however, occur in schools. Many in the computer science field say the U.S. is severely lagging in making even basic computer science a priority in K–12 schools. "While other countries have designed and implemented national computer science education programs in order to better prepare their students for the increasingly competitive global economy, the decentralized (state, district-wide, and even school-based) educational decision-making process in the U.S. has severely hampered efforts to standardize our computer science curriculum and create coherence in student learning," according to the 2010 report, "Addressing Core Equities in K–12 Computer Science Education."

Guzdial believes the biggest problem in teaching computer science in the U.S. is the lack of teachers who know the discipline. He estimates there are about 30,000 high schools in the country, but only 2,000 Advanced Placement computer science teachers.

Students as young as five can learn to program, Guzdial maintains, but he

Students as young as five can learn some basic concepts of programming, similar to the number and counting skills children typically are taught at that age.

questions why they should. "I am not sure what we can teach them from a cognitive perspective." Although there have been studies done on children learning to program in the Scratch programming language, "in general what we find is kids that young do not do the things you naturally would expect coding to involve," including loops and conditionals.

He says he is concerned about cognitive development. "What we know about cognitive development is you typically develop the ability to do abstract reasoning around the age of 12," and programming is a very abstract activity. Guzdial is unsure whether young children who program develop abstract reasoning earlier, or if they are only able to learn a little bit of programming skills.

Overall, though, he says computer science should be taught in schools—but starting at age five or six, when only 12% of high schools in the U.S. offer computer science courses and far fewer middle and elementary schools, creating a great divide. Guzdial says, "They are unlikely to see it again for a dozen years, so why offer it at five or six?"

Wing also says that, while age five may be too early to teach how to code, students that young can learn some basic concepts similar to the number and counting skills children typically are taught at that age. As they get older, students should be taught other concepts, like what an algorithm is, ways to represent data, and different analysis techniques in order to understand and reason, she says.

Looking Ahead

Just as students are taught reading, writing, and the fundamentals of math and the sciences, computer science may one day become a standard part of a K–12 school curriculum. If that happens, there will be significant benefits, observers say. As the kinds of problems we will face in the future will continue to increase in complexity, the systems being built to deal with that complexity will require increasingly sophisticated computational thinking skills, such as abstraction, decomposition, and composition, says Wing.

"If I had a magic wand, we would have some programming in every science, mathematics, and arts class, maybe even in English classes, too," says Guzdial. "I definitely do not want to see computer science on the side ... I would have computer science in every high school available to students as one of their required science or mathematics classes." 

Further Reading

C. Simard, C. Stephenson, D. Kosaraju
"Addressing Core Equities in K–12 Computer Science Education: Identifying Barriers And Sharing Strategies," 2009.
<http://anitaborg.org/files/ABI-csta-full-report.pdf>

C. Felker
"Maybe Not Everybody Should Learn to Code," 2013. http://www.slate.com/articles/technology/future_tense/2013/08/everybody_does_not_need_to_learn_to_code.html

Martyr2
"Why Everyone Should NOT Learn to Code," 2013. <http://www.coderslexicon.com/why-everyone-should-not-learn-to-code/>

D. Haggard
"Why Everyone Should Learn to Program," 2011. <http://reviewsindepth.com/2011/04/why-everyone-should-learn-to-program/>

P. Norvig
"Teach Yourself Programming in Ten Years," 2001. <http://norvig.com/21-days.html>

J. Lave
"Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life," 1988, Cambridge University Press.

J.R. Hayes
"The Complete Problem Solver," 1989. Lawrence Erlbaum Associates, Inc.

Esther Shein is a freelance technology and business writer based in the Boston area.

Computational Photography Comes into Focus

Advances in computational photography are making image capture the starting point. The technology is transforming the field.

OVER THE LAST decade, digital cameras have radically refocused the way people capture and manipulate pictures. Today, the snap of a photo is merely a starting point for composing and manipulating an image. A photographer can make basic changes to a picture from within the camera, but also may use photoediting software on a computer to significantly alter the look, feel and composition. “We can use computation to make the process better, both aesthetically and in terms of greater flexibility,” explains Frédo Durand, a professor in the Computer Science and Artificial Intelligence Laboratory at MIT in Cambridge, MA.

Researchers and engineers are now taking the concept further. They are designing different types of cameras, developing increasingly sophisticated algorithms, and using new types of sensors and systems to boldly go where no camera has gone before. The ability to record richer information about a scene and use powerful image enhancement techniques are redefining the field. “Computational photography and computational imaging are extremely vibrant areas,” states Shree K. Nayar, professor of computer science at Columbia University in New York City.

These cameras, along with more advanced software, will radically change the way people view and use images. For example, they will make it possible to detect a tiny object or imperceptible motion from the field of view. They might change the perspective or angle after a photo is snapped, or provide a 360-degree panoramic view. They might also augment reality and refocus various objects in scenes, after a photo has been shot.



The Lytro camera captures the entire light field.

Meanwhile, smartphone cameras will further redefine photography by incorporating sensors and greater onboard computational power. Combined with specialized apps or cloud-based services, they will stretch the current concept of photography in new and intriguing ways.

A Better Image

It is no secret that digital cameras have reinvented photography. The transition from film to pixels has created an opportunity to manipulate and share photos in ways that were not imaginable in the past. However, today's cameras rely heavily on the same features and image capture techniques as film cameras; they are largely designed the same way film cameras were, but with new features. “They present a lot of limitations. It is very difficult to change

the way the camera behaves or the way it captures images,” Durand explains.

However, the use of computational photography, imaging, and optics promises to significantly change the way people approach photography, capture images, and edit them. For example, William Freeman, a professor of computer science at MIT, says computational cameras could capture multiple images at a time to compensate for glare, oversaturation, and other exposure problems. They could also eliminate the need for a flash. “Too often, flash ruins the tonal scale of images,” he says, “but by combining multiple shots, both with flash and without, it is possible to create a single sharp, low-noise image that has a beautiful tone scale.”

Similarly, the ability to change focus after capturing a shot would make it

possible to fix on a person in the foreground while also focusing on an object in the distance, like the Eiffel Tower or Statue of Liberty; everything else in the photo would appear blurred. The commercially available Lytro camera—which records the entire light field in the frame (essentially, depth of field data about the entire scene)—already allows a user to refocus pictures and adjust lighting after image capture. Likewise, a sensor that would capture different levels of light on different pixels could create entirely new types of photographs, including images with markedly different brightness and color ranges.

The technology of computational photography could also lead to changes in camera design. As Columbia's Nayar points out, computational features alone deliver significant improvements, but they also create the possibility for new types of camera bodies, lenses, and optics. Adding a computational lens to a smartphone, for instance, could mimic the high-end features of an expensive optical lens at a much lower price point, or may create entirely new features. A photographer might snap on a lens or multiple lenses that would provide 3-D capabilities, or marry video and still photography to address camera shake, particularly in difficult low-light or high-speed environments.

The benefits of computational cameras and software are likely to extend far beyond consumers. The technology could impact an array of industries, including medicine, manufacturing, transportation and security, points out Marc Levoy, a professor of computer science and electrical engineering at Stanford University in Palo Alto, CA, who recently took leave to work with the Google Glass development team. Levoy says cameras with more advanced computational capabilities could redefine the way we think about the world around us, and provide insights that extend beyond basic images or video.

For example, he and other researchers have explored the idea of developing a computational camera that could see through crowds, objects, and people. The technology could also generate a focal stack within a single snapshot. This could create new opportunities

Computational photography could lead to changes in camera design, such as new types of camera bodies, lenses, and optics.

in biology and microscopy, Levoy says. "A technician could capture images of cell cultures without focusing a microscope; focusing would take place after the picture is taken." A computational camera could also automatically count the number of cells in an image and provide information faster and more accurately than any human, he adds.

Perhaps the highest-profile example of a computational photography system to date is Google Glass. Its camera captures images and provides additional information and insight in an array of situations and scenarios—a step toward more-advanced augmented reality tools. Among other things, the Google Glass team is focused on developing map data, language translations, travel and transit information,

and apps that track health, exercise data and body information. The device also can capture a burst of images and deliver improved high-dynamic-range imaging and low-light imaging.

Beyond Pixels

Engineering these systems and developing the algorithms to support these devices is no simple task, particularly as researchers look to extend computational capabilities beyond the world of consumer cameras into fields such as astronomy, medical photography, and automobile photography. There also is the possibility of capturing images beyond the visible spectrum of light, incorporating environmental sensors, or finding ways to apply algorithms to detect small but important changes in the environment. As Levoy puts it, "There is a potential for this technology to be extremely disruptive."

Durand also says the gains are not limited to conventional cameras. New types of cameras and software could generate robust 3-D images that reveal things not visible through optics alone. Already, he and Freeman have developed algorithms that can sense the flow of blood in a person's face, or detect one's heartbeat based on subtle head motions. This relates to a technique called *motion magnification*, which could potentially be used to detect weaknesses in bridges and build-



A cutaway view of the Canon EOS 5D Mark II camera body.

IMAGE COURTESY OF CANON USA

ings; it amplifies pulse signals and color variations. “These signals cannot be detected by the human eye, but they are revealed through advanced computational imaging and slow-motion analysis,” Freeman explains.

Vladimir Katkovnik, a professor of signal processing at Tampere University of Technology in Finland, says a significant hurdle to accomplishing all this is the development of algorithms that sort through all the data and apply it in usable ways. Despite the prospect of larger sensors that can capture more data, there is a trend toward more pixels in images. “Larger numbers of megapixels means images with more pixels of a smaller size. As smaller numbers of photons appear on a pixel during exposure time, there is a larger amount of noise generated. Noise removal is a growing challenge in any imaging or sensing device; the end quality depends on how well noise is removed.”

Another challenge, Durand says, is developing robust algorithms that work effectively on relatively small devices such as cameras, smartphones, and tablets. “The issue is not necessarily whether you can develop an algorithm that works; it is whether it is possible to map the computation to the hardware in an efficient manner. Writing optimized code that can take advantage of modern hardware, including mobile processors, is extremely difficult.” He is currently developing a compiler to make it easier to achieve high performance, without devoting a large development team to the task.

Nayar believes researchers will tap into big data techniques and, in some cases, examine and analyze existing photos to build algorithms that drive even more sophisticated image processing. Right now, “if you try to remove a person or object from a photo, there is no easy way to fill the hole, even with fairly sophisticated photoediting software,” he says. “By using millions of pictures and applying machine learning algorithms, it is possible to fill the holes in visually plausible ways.” At some point, he adds, these capabilities will likely appear on cameras, smartphones, and tablets, and provide nearly instantaneous manipulation and editing tools that make today’s image-editing options pale by comparison.

Researchers are likely to hit the tipping point within the next decade, as increasingly powerful processors and a greater knowledge of physics push the technology forward. “The algorithms being used today are still mostly in the infant stages,” Nayar says. “So far, most of the research has revolved around extending the capabilities of traditional imaging and finding ways to improve the performance of digital cameras.” As knowledge about non-traditional imaging and optics converge, he notes, “everything from chip design to lens and camera design will undergo major changes.”

In the end, Durand says it is important to place computational photography, imaging, and optics in the right context. The technology will not replace today’s cameras and photographs; it will enhance them and continue advancing a process that dates back thousands of years, to the development of pinhole cameras. Computational photography puts data to use in new and better ways, whether it is applied to DNA sequencing or to improved traffic cameras or security tools.

Says Durand, “Photography is just one aspect of a much bigger picture. With it, we are able to see the world in a fundamentally different way.” **C**

Further Reading

Ragan-Kelley, J., Adams, A., Paris, S., Levoy, M., Amarasinghe, S., Durand, F.

Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines, SIGGRAPH 2012, <http://people.csail.mit.edu/jrk/halide12/halide12.pdf>.

Bychkovsky, V., Paris, S., Chan, E., Durand, F. When Does Computational Imaging Improve Performance?, IEEE Transactions on Image Processing, 2012. http://www1.cs.columbia.edu/CAVE/publications/pdfs/Cossairt_TIP12.pdf

Cossairt, O., Gupta, M., Nayar, S.K. Ironies of automation. *New Technology and Human Error*, J. Rasmussen, K. Duncan, J. Leplat (Eds.). Wiley, Chichester, U.K., 1987, 271–283.

Cho, T.S., Avidan, S., Freeman, W.T. A Probabilistic Image Jigsaw Puzzle Solver, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2010. <http://people.csail.mit.edu/billf/papers/JigsawSolverCVPR2010.pdf>

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

© 2014 ACM 0001-0782/14/02 \$15.00

Opportunity

Heidelberg Laureate Forum

Individuals may apply through the end of February for one of 200 openings to attend the second Heidelberg Laureate Forum (HLF), to be held Sept. 21–26 at Heidelberg University in Germany.

HLF allows researchers from all over the world to interact with laureates of the most prestigious awards in computer science and mathematics. Last year, 40 laureates, including recipients of the ACM A.M. Turing Award, the International Mathematical Union’s Fields Medal and Nevanlinna Prize, the Norwegian Academy of Science and Letter’s Abel Prize, addressed young researchers on topics that ranged from “how to do research” to deep technical areas of science and math.

ACM Europe chair Fabrizio Gagliardi, one of a number of ACM members (among others) who helped organize the initial Forum, said about last year’s event, “I was impressed by the attitude of the laureates who all spent a considerable amount of time networking with the young researchers; not only during the HLF sessions, but also during meals and in after-dinner discussions. This is probably the highest value of the event: providing a relative small set of promising future scientists with the unique opportunity to engage with some of the most brilliant minds in mathematics and computer science.”

Interested researchers may apply on the HLF website, at <https://application.heidelberg-laureate-forum.org>.

HLF is organized by the Heidelberg Laureate Forum Foundation in cooperation with Klaus Tschira Stiftung and the Heidelberg Institute for Theoretical Studies, as well as ACM, the International Mathematical Union, and The Norwegian Academy of Science and Letters.

—Lawrence Fisher

ACM Fellows Inducted

ACM HAS RECOGNIZED 50 of its members for their contributions to computing that are driving innovations across multiple domains and disciplines. The 2013 ACM Fellows, representing many of the world's leading universities, corporations, and research labs, have achieved advances in computing research and development that are accelerating the digital revolution and impacting every dimension of how we live, work, and play ... worldwide.

"We recognize these scientists and engineers, creators and builders, theorists and practitioners who are making a difference in our lives," said ACM President Vinton G. Cerf. "They're enabling us to listen, learn, calculate, and communicate in ways that underscore the benefits of the digital age. Their advances have led to opportunities for improved healthcare, enhanced security, expanded interactions, and enriched lifestyles. Some recipients have also led efforts to extend computing across continents and countries including Brazil, China, and Germany."

The ACM Fellows Program was established by Council in 1993 to recognize and honor outstanding ACM members for their achievements in computer science and information technology and for their significant contributions to the mission of the ACM. For a complete list of ACM Fellows, visit <http://fellows.acm.org/>

2013 ACM Fellows

Mark S. Ackerman
University of Michigan

Charu C. Aggarwal
IBM Research

James H. Anderson
University of North Carolina
at Chapel Hill

Mihir Bellare
University of California, San Diego

Christine L. Borgman
University of California,
Los Angeles

Stefano Ceri
Politecnico di Milano

Krishnendu Chakrabarty
Duke University

Ramalingam Chellappa
University of Maryland

Ingemar J. Cox
University of Copenhagen,
University College London

Carlos J. P. De Lucena
Pontifical Catholic University
of Rio de Janeiro

Rina Dechter
University of California, Irvine

Chip Elliott
Raytheon BBN Technologies

David Forsyth
University of Illinois
at Urbana-Champaign

Wen Gao
Peking University

David Garlan
Carnegie Mellon University

James Gosling
Liquid Robotics

Peter Haas
IBM Research - Almaden

Marti Hearst
University of California, Berkeley

Matthias Jarke
RWTH Aachen University
(Germany Aachen University
of Technology)

Sampath K. Kannan
University of Pennsylvania

David J. Kasik
Boeing

Dina Katabi
Massachusetts Institute
of Technology

Henry A. Kautz
University of Rochester

Jon Kleinberg
Cornell University

Panganamala Kumar
Texas A&M University

Douglas S. Lea
State University of New York, Oswego

Yoelle Maarek
Yahoo!

Christopher D. Manning
Stanford University

Madhav V. Marathe
Virginia Bioinformatics Institute
and Virginia Polytechnic Institute

John M. Mellor-Crummey
Rice University

Greg Morrisett
Harvard University

Andrew C. Myers
Cornell University

Dana Nau
University of Maryland

Satish Rao
University of California, Berkeley

S.E. Robertson
University College London

Timothy Roscoe
ETH Zurich

Timoleon K. Sellis
RMIT University (Australia)

Dennis E. Shasha
Courant Institute,
New York University

Nir N. Shavit
Massachusetts Institute
of Technology

Kyuseok Shim
Seoul National University

Padhraic Smyth
University of California, Irvine

Milind Tambe
University of Southern California

Val Tannen
University of Pennsylvania

David P. Williamson
Cornell University

Limsoon Wong
National University of Singapore

Moti Yung
Google Inc.

Ellen Zegura
Georgia Institute of Technology

Zhengyou Zhang
Microsoft Research

Yuan Yuan Zhou
University of California, San Diego

David Zuckerman
University of Texas at Austin

Inviting Young Scientists

Meet Some of the Greatest Minds of Mathematics and Computer Science

Young researchers in the fields of mathematics and/or computer science are invited to participate in an extraordinary opportunity to meet some of the preeminent scientists in the field. ACM has joined forces with the Heidelberg Laureate Forum (HLF) to bring students together with the very pioneering researchers who may have sparked their passion for science and math. These role models include recipients of the Abel Prize, the ACM A.M. Turing Award, and the Fields Medal.

The next Heidelberg Laureate Forum will take place September 21-26, 2014 in Heidelberg, Germany.

The week-long event will focus on scientific inspiration and exchange through a series of presentations, workshops, panel discussions, and social events involving both the laureates and the young scientists.

Who can participate?

The HLF invites new and recent Ph.D's, Ph.D. candidates, other graduate students involved in research and undergraduate students with solid experience in and a commitment to computing research to apply.

How to apply:

Young researchers can apply online:

<https://application.heidelberg-laureate-forum.org/>

The materials required for a complete application are listed on the site.

What is the schedule?

The deadline for applications is **February 28, 2014**.

We reserve the right to close the application website early should we receive more applications and nominations than our reviewers can handle.

Successful applicants will be notified by **April 15, 2014**.

Privacy and Security

Would Cybersecurity Professionalization Help Address the Cybersecurity Crisis?

Evaluating the trade-offs involved in cybersecurity professionalization.

THE THOUSANDS OF serious cyber attacks occurring daily highlight the critical need for a workforce with the requisite skillset and of sufficient size to meet growing and increasingly complex demands. Yet despite significant investments in the development of the cybersecurity workforce from governments across the globe, the U.S. and many other nations lack a sufficient supply of well-trained cybersecurity professionals. It is often argued that this workforce shortage, and the consequent openness to attack, is a pressing security threat facing the U.S.¹

Professionalization—activities such as certification, licensure, and skill-based competency exams—has been advanced as a strategy for creating a workforce capable of addressing the growing cybersecurity threat.

To explore this argument, the U.S. Department of Homeland Security sponsored a National Research Council committee, which we led. What follows are insights largely drawing on the study and although the impetus

Despite descriptions of the cybersecurity workforce as a “profession”—meaning a single occupational category—it is not.

for asking the question at this moment came from the U.S. government, the issues and analysis would have general applicability. Our key question was: What is the role that professionalization might play in enhancing the capacity and capability of the U.S. national cybersecurity workforce? This question led to a complex mosaic of answers to the cybersecurity workforce issue.

The Cybersecurity Workforce

Despite descriptions of the cybersecurity workforce as a “profession”—meaning a single occupational category, it is not. Rather, cybersecurity is a broad field comprised of many occupations spanning the range from highly technical to the management- or policy-oriented. Some of these occupations may be ready for professionalization, while others are



not. Others are yet to be defined. Still others may never be defined either because the fluidity of the roles and responsibilities change too rapidly to allow for categorization or because they are hybrid occupations that blend cybersecurity responsibilities with other, often unrelated work roles. Given the great diversity of roles, responsibilities, and contexts, the fact that professionalization measures may be warranted in a particular subfield and context should not be confused with a broad need for professionalization.

Before professionalization activities are undertaken for an occupation, the profession itself must have well-defined characteristics: stable knowledge and skill requirements, stable job roles, occupational boundaries, and career ladders.

► Stable knowledge and skill requirements: The occupation should

have a stable (but not necessarily static) common body of knowledge on which members of the profession can be judged to a generally agreed upon standard. This does not imply, however, that the occupation is static; even within a rapidly evolving profession, core knowledge elements that remain stable can be identified.

► Stable roles and responsibilities and occupational boundaries that distinguish the profession from others.

► Well-defined career ladders that are linked to professionalization mechanisms.

► Agreed-upon ethical standards to which members of the profession will be held and a mechanism for removing noncompliant individuals from the professional ranks.

The fact that the current cybersecurity workforce is a field of multiple occupations highlights a significant

problem with current approaches to professionalization. Realistically, such professionalization can only be undertaken for specific occupations within the field, but not for the field as a whole.

Professionalization

Professionalization is the process by which an occupation (or an individual who works within that occupation) is transformed through education, training, and other activities into a professional. Each occupation must exhibit some set of well-defined characteristics before professionalization activities commence. Not all of these characteristics or standards must be met, but the level of occupational readiness for professionalization is higher when more of them are. Readiness for professionalization, however, does not imply the occupation should

be professionalized, nor does it identify the appropriate professionalization mechanism. It simply means the occupation could be professionalized if circumstances warrant the activity. At this point, the question becomes what are the deficiencies within the occupation that could be alleviated through professionalization.

The process of professionalization is initiated based on some deficiency in the occupational workforce—a lack of public trust, questionable skill or performance, weak behavioral or ethical standards, low status, noncompliance with regulatory or legal requirements, ill-defined career pathways, or unregulated labor supply (when a steady flow of workers is desired or necessary). But as has been stated, the cybersecurity workforce challenge is one of capacity and capability. This statement, though compelling, is not sufficient to initiate professionalization activities.

Rather, we must unbundle this statement and ask difficult questions about the precise nature of the need. If the workforce need is for more accountability in the maintenance of hands-on skillsets within a particular occupation, then the professionalization mechanism should be focused on continuing education requirements and skill-based testing. If, on the other hand, the nature of the workforce challenge is related to troubling examples of ethical lapses, then professionalization activities should focus on some type of compliance mechanisms from a formal authority. The alignment of professionalization strategies with specific workforce challenges is necessary to ensure the deficiency is, in fact, addressed. It is also critical to ensuring the possible negative consequences of professionalization do not outweigh the good.

Trade-Offs of Professionalization

Even when the professionalization activity is aligned with the occupational deficiency, it will have associated trade-offs. These costs and benefits should be considered before embarking on a professionalization activity.

Do the benefits of a given professionalization mechanism outweigh the potential supply restrictions resulting

from the additional barriers to entry? Professionalization can serve as a magnet that attracts people to the occupation, as a funnel that restricts the supply of people entering the occupation, or as a sieve that filters people out of the occupation based on increased requirements.

► **The Magnet:** Professionalization may increase the supply over time as it helps increase awareness and desirability of that profession, and thus increases the number of individuals who consider cybersecurity as a career. By helping define roles and career paths, it can also help workers identify suitable jobs and help employers identify suitable workers. Specialization and stratification may also help address supply issues, much as the introduction of nurse practitioners and physical assistants expanded the workforce providing primary medical care.

► **The Funnel:** No one would argue against restricting the supply of unqualified individuals in a workforce. Certainly, professionalization mechanisms that address the capability of the workforce should be in place if capability is a concern. However, overly narrow professionalization or mismatched mechanisms may unnecessarily filter out qualified workers whose skills are needed. For example, the requirement for entry-level, technical employees to hold a bachelor's degree when an associate's degree and passing a skill-based exam may be more appropriate unnecessarily re-

stricts the flow of qualified workers.

► **The Sieve:** The sieve function is of particular concern in cybersecurity where many members of the workforce function in hybrid positions and are subject to professionalization requirements in those other roles. Consider, for example, the healthcare professional who has added cybersecurity responsibilities to her portfolio and must meet a double set of requirements. If the professionalization requirement is necessary to determine or verify skill requirements then it may be appropriate. If, on the other hand, the requirement has been implemented without regard to remedying a specific deficiency, then it may unnecessarily burden and ultimately encourage the departure of the individual from the workforce.

Does the potential to provide additional information about a candidate outweigh the risks of false certainty about who is actually best suited for a job? Certificates and certifications may provide useful tools for vetting job candidates, but overreliance on them may screen out some of the most talented and suitable individuals. This is particularly true in cybersecurity today, where some of the most effective workers develop their skillsets through informal methods (for example, self-taught hackers). Organizations that do not already have a sophisticated cybersecurity workforce may place a greater value on professionalization measures because they make it easier for them to identify qualified workers. However, at a time when few think the cybersecurity situation is improving, and where “side-ways” thinking may be at a premium, creativity and innovation may be lost with overly rigid screening. Moreover, given the fluid and changing nature of cybersecurity work, the knowledge, skills, and abilities actually needed in a particular job can change, and workers’ roles and responsibilities can also shift rapidly.

Do the benefits of establishing the standards needed for professionalization outweigh the risks of obsolescence (when the knowledge or skills associated with the standard are out of date by the time a standard is agreed on) and ossification (when the establishment of a standard inhibits further

Before professionalization activities are undertaken for an occupation, the profession itself must have well-defined characteristics.

development by workers of their skills and knowledge)? It takes time to reach consensus on the standards needed to establish a curriculum or certification, and it can be difficult to reach convergence, given the rate of change in underlying technologies and the rapid pace at which the context and threat evolves. Following receipt of a degree or certification, workers may stop developing their skills and knowledge. Strategies for addressing these challenges, including focusing assessments as much as possible on fundamental concepts, segmenting a field (where possible) into sufficiently narrow specialty roles, adopting more nimble processes for updating content, and requiring continuing education and periodic recertification to refresh requirements.

These trade-offs illustrate the complex set of costs and benefits associated with professionalization. Some of the uncertainties may diminish over time, and long-term benefits may ultimately outweigh short-term costs. It may, thus, be an effective strategy to encourage, rather than require, the use of certain professionalization mechanisms so as to avoid overly restricting supply in the short term while still establishing a long-term path to enhancing quality.

Conclusion

Continued attention to the capacity and capability of the cybersecurity workforce is needed. Over time, parts of the cybersecurity field will likely reach the point where professionalization will be warranted. But blanket professionalization strategies will hinder efforts to build a national cybersecurity workforce of sufficient size, scope, and ability to meet the demands of the rapidly evolving field. The criteria set forth in the National Research Council *Professionalization of the Nation's Cybersecurity Workforce?* report² can be used by decision-makers to judge when that time has come.

Activities by the U.S. federal government and other entities to professionalize cybersecurity should be undertaken *only when* the occupations and specific occupational characteristics have been defined, when there are observed deficiencies in the occupational workforce that professionaliza-

Continued attention to the capacity and capability of the cybersecurity workforce is needed.

tion could help remedy, and when the benefits of those activities outweigh the costs. When stakeholders believe those conditions have been met, we suggest they convene subject matter experts to outline a professionalization strategy—including timeline, process, and other implementation details.

This process will take time. But the path to professionalization of a field is slow and difficult, and not all portions of a field can or should be professionalized at the same time. Until that time, our work to develop a national cybersecurity workforce of sufficient capacity and capability should move away from overly broad generalizations based on anecdotal evidence and context-specific challenges, toward a set of targeted activities that meet identified and specific occupational workforce deficiencies. **C**

References

1. Homeland Security Advisory Council. *Cyber Skills Task Force Report*. Department of Homeland Security, Washington, D.C., 2012.
2. National Research Council. *Professionalizing the Nation's Cybersecurity Workforce?: Criteria for Decision-Making*. The National Academies Press, Washington, D.C., 2013.

Diana L. Burley (dburley@gwu.edu) is an associate professor of Human and Organizational Learning in the Graduate School of Education and Human Development at George Washington University.

Jon Eisenberg (JEisenbe@nas.edu) is the director of the Computer Science and Telecommunications Board at the National Research Council in Washington, D.C.

Seymour (Sy) E. Goodman (seymour.goodman@cc.gatech.edu) is a professor of International Affairs and Computing, jointly at the Sam Nunn School of International Affairs and the College of Computing at the Georgia Institute of Technology.

The views expressed in this Viewpoint are those of the authors and do not necessarily reflect those of the National Research Council, the Committee on Professionalizing the Nation's Cybersecurity Workforce, which wrote the report, or the U.S. Department of Homeland Security, which sponsored the study.

Copyright held by Author/Owner(s).

Calendar of Events

March 19–21

Multimedia Systems Conference 2014, Singapore, Sponsored: SIGMM, Contact: Roger Zimmermann, Email: rogerz@comp.nus.edu.sg

March 24–28

Design, Automation and Test in Europe, Dresden, Germany, Sponsored: SIGDA, Contact: Gerhard Fettweis, Email: Gerhard.fettweis@tu-dresden.de

March 24–28

Symposium on Applied Computing, Gyeongju, Republic of Korea, Sponsored: SIGAPP, Contact: Sung Shin, Email: sung.shin@sdstate.edu

March 26–28

Eye Tracking Research and Applications, Safety Harbor, FL, Sponsored: SIGCHI, SIGGRAPH, Contact: Pernilla Qvarfordt, Email: pernilla.qvarfordt@gmail.com

March 29–April 2

12th Annual/IEEE/ACM International Symposium on Code Generation and Optimization, Orlando, FL, Sponsored: SIGMICRO, SIGPLAN, Contact: David Kaeli, Email: kaeli@ece.neu.edu

March 30–April 2

International Symposium on Physical Design, Petaluma, CA, Sponsored: SIGDA, Contact: Cliff Chin Ngai Sze, Email: csze@us.ibm.com

May 6–9

ACM The First Annual International Conference on Nanoscale Computing and Communication, Atlanta, GA, Contact: Ian F. Akyildiz, Email: ian@ece.gatech.edu

May 7–9

Gender and IT Appropriation, Science and Praxis in Dialogue – Forum for Interdisciplinary Exchange, Siegen, Germany, Contact: Wulf Volker, Email: volker.wulf@uni-siegen.de

Education

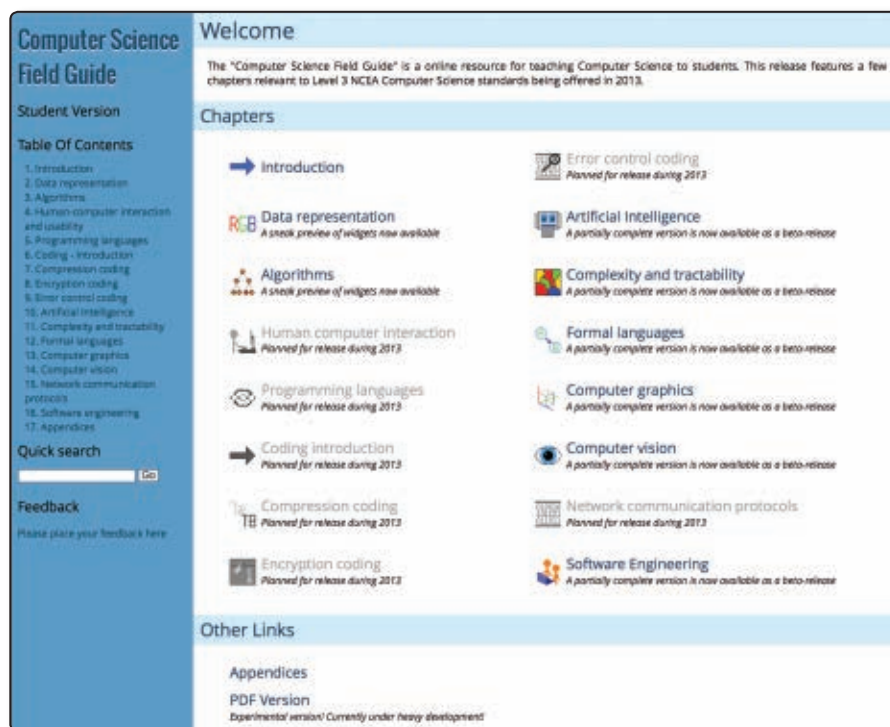
Establishing a Nationwide CS Curriculum in New Zealand High Schools

Providing students, teachers, and parents with a better understanding of computer science and programming.

IN 2011, COMPUTER SCIENCE was introduced as a subject in New Zealand high schools with a similar standing to subjects like physics, as part of a new set of education standards under the umbrella term “digital technologies.”² Since then, at least 150 teachers have increased their skill sets in order to teach these unfamiliar topics, and thousands of students have passed courses in programming and computer science topics.

This rapid introduction of radically new material has not been easy; as well as having to train teachers and develop new teaching material, during the transition students had to prepare for assessments that no other students had done before, and teachers who embraced the changes often had to work with school leaders who had little understanding of what computer science involves.

The details of the implementation are available elsewhere,¹ but the key points are that “Programming and Computer Science” was introduced as an assessable subject for the final three years of high school, and the material was phased in from 2011 to 2013 respectively, so the first cohort of students to have taken the new topics left school in December 2013. With the new content, a course on computer science can include topics like



Computer Science Field Guide table of contents (see <http://csfieldguide.org.nz>).

algorithms, HCI, formal languages, complexity and tractability, intelligent systems, software engineering, and graphics and visual computing, in addition to programming. Problem solving and creativity already permeate the New Zealand curriculum, and the new content gives students the opportunity and tools to be creative in new ways.

When outsiders see topics such as

“formal languages” as a school topic, they may wonder if things have been taken too far, and when teachers do an online search for that phrase they are likely to encounter an overwhelming array of teaching material. However, the purpose of the new curriculum is to give students a taste of the field of computer science, not to teach it in great detail. For example, formal

languages can be introduced by working with some simple Finite State Automata and experimenting with some regular expressions, concepts that can be introduced in a few hours of class time. From this students can appreciate the role of formal languages (for example, to find identifiers in a program) without having to grapple with details. To support teachers, a number of resources have been collected on the national teachers' association website (see <http://nzacditt.org.nz>), and are now being worked into a free interactive online textbook called the *Computer Science Field Guide* (see <http://csfieldguide.org.nz>).

The broad range of topics is important to help students see the breadth of options computer science offers, and also takes some focus away from merely programming by showing the range of knowledge needed to produce effective software, and the value that people with good human skills can bring to the discipline. For example, Human-Computer Interaction (HCI) is deliberately included early. The way it is presented is focused on evaluating existing interfaces to find design errors, taking into account usability principles and basic psychology relating to interaction. This can level the playing field—students who have been frustrated with computers find a way to articulate what is wrong with them, whereas sometimes students who themselves can develop programs with elaborate but confusing interfaces struggle to see the flaws because they assume the user understands the program as well as they do.

For teaching programming, the choice of language is not specified, although for the final year the language needs to support object-oriented programming and graphical user interface development. Many schools are using Scratch as an introductory language at lower levels, and then switching to a text-based language for the more advanced levels. Python is emerging as the most popular text-based language, but JavaScript, Java, and Visual Basic are also widely used.

Although the new content is targeted at the final three years of high school (when students have exams that become a permanent record of their school achievements), the material is filtering down to lower levels.

The magnitude of the changes is easy to underestimate, and the teachers have borne the brunt of the transition.

This is proving to be valuable, as it is easier to come to grips with a subject if it is taught over a number of years rather than suddenly becomes available in the same year it is being assessed for a student's permanent record. Without the gentle introduction the risk is that students become focused on learning just enough to pass assessment with minimal effort, and they can miss out on enjoying and exploring the subject. The new curricula about to be introduced in the U.K. and Australia explicitly cover the full range of levels at school, whereas in New Zealand the curriculum allows it to appear at lower levels, but it is up to the enthusiasm and awareness of a particular teacher or their school leadership to make that happen.

Challenges with the Implementation

Before the 2011 changes occurred, computing in schools was focused on teaching students how to *use* computers, which led to courses that were not regarded as academically challenging, and thus were avoided by the students who were most likely to do well in a career in computer science. The changes have transformed it into a challenging subject, and this has created some problems in the transition as students, parents, and even student advisors are unable to understand the difference, resulting in the wrong students joining these courses (in New Zealand, students have a lot of choice over which courses they take in high school). In a 2012 survey, one year after the new courses started, 46% of teachers reported their courses attracted a significant number of students who lacked the academic ability to perform well,

which reflects a difficulty communicating the nature of the new courses.⁴ An important challenge is to help school management and student advisors to understand what computer science is, its value for students, and the kind of students who would do well to study it.

Marketing the subject is always a challenge. It is reasonable to let students know about the high demand for computer science graduates, the high salaries that are available, and the great work environments, but we also must ensure students do not overlook the fact that there is a lot of work to be done to get to the point of being hired, and that computer science can be a demanding subject. Learning to program is not trivial, and we need to straddle the line between not dressing it up as an easy option, but also encouraging students to try out the subject to find out if it is indeed a strength they did not realize they had. Also, money may not be a motivator for all students; for some students the best message may be that they can make a difference in the world, designing software that helps people in areas such as medicine, communication, and safety.

The Key Role of Teachers

The magnitude of the changes is easy to underestimate, and teachers have borne the brunt of the transition. In New Zealand, the new material was introduced with very little lead time, and for various reasons there was no opportunity to train a new generation of teachers, and very little time and resourcing to increase the skills of existing teachers. The majority of existing computing teachers are over 50 years old, yet most have embraced the change, primarily because of wanting to do the right thing for the students and the country, rather than due to any directives from management.⁴

The rapid transition has been somewhat unsettling for many teachers. From the New Zealand experience, we have found that teachers are willing to learn, but they need to be valued and supported. For those without a background in computer science (and this is the majority of teachers in New Zealand, as many had started as typing or commerce teachers), they could feel like an imposter, and it is important to value what they bring—



Association for
Computing Machinery

ACM Conference Proceedings Now Available via Print-on-Demand!

*Did you know that you can
now order many popular
ACM conference proceedings
via print-on-demand?*

Institutions, libraries and individuals can choose from more than 100 titles on a continually updated list through Amazon, Barnes & Noble, Baker & Taylor, Ingram and NACSCORP: CHI, KDD, Multimedia, SIGIR, SIGCOMM, SIGCSE, SIGMOD/PODS, and many more.

**For available titles and
ordering info, visit:
librarians.acm.org/pod**

A key factor with the new material has been to avoid covering too much new content.

their teaching experience, wisdom, passion, and ability to relate to students—and provide them with learning opportunities in a form and at a time that they can access them. Some teachers will pick up things quickly if they have already had some experience in programming, while others will need some time.

The single most popular source of help has been a national mailing list run by the teachers themselves, where many thousands of email messages have been exchanged sharing ideas, teaching material, and peer support. This is a powerful tool because peers best understand the issues that each other face. In addition, several Google CS4HS events³ have been run that provided intense training sessions focused entirely on preparing for the changes. More recent initiatives have been a formal postgraduate course teachers can take in their own time, and having university CS students work alongside teachers, helping them to understand the content while the teacher provides the classroom management skills.

Changes like this require a strong message to school management, who need to support it with resourcing, time for teachers to get up to speed, putting structures in place to enable students to make informed decisions about the new courses (such as having lower-level classes with some input from a computer science teacher), and providing the right advice to students and parents. One teacher commented that the management assumes that “all students know how to surf the Net and operate a smart device and therefore will find computer science easy.” Changing this attitude will make the introduction of computer science con-

siderably easier for teachers, and without it the changes can backfire, with the wrong students taking the subject, leading to a vicious cycle of low pass rates, students avoiding the subject, smaller classes in computing, and threats to computing teachers' jobs.

A key factor with the new material has been to avoid covering too much new content. Setting expectations too high could backfire by causing schools and students to avoid the subject; the changes will be a success if high school students simply find out what CS and programming are about. We do not need to teach them everything, and do not need to produce expert programmers, since there are plenty of opportunities to do that beyond high school. If students just find out they have a passion for the subject, the changes will have been a success. For some this may involve deciding that although they like computers, they do not like computer science, while others may find it is much more interesting than they imagined. The key is that they make informed choices about their career.

The overarching issue that continually arises is the lack of understanding of what computer science is about. If students, school administrators, teachers, and parents gain a good idea of what it really means, this will have the biggest impact on the skills pipeline. C

References

1. Bell, T., Andreae, P., and Lambert, L. Computer science in New Zealand high schools. In *ACE '10: Proceedings of the 12th Conference on Australasian Computing Education (Australian Computer Science Communications)*, volume 32. T. Clear and J. Hamer, Eds. Australian Computer Society, Inc., Brisbane, Australia, 2012, 15–22.
2. Bell, T., Andreae, P., and Robins, A. Computer science in NZ high schools: The first year of the new standards. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (Raleigh, N.C., 2012)* L.A. Smith King, D.R. Musicant, T. Camp, and P. Tymann, Eds. ACM, New York, 343–348.
3. Blum, L. and Cortina, T.J. CS4HS: An outreach program for high school CS teachers. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2007* (Covington, KY, 2007) I. Russell, S.M. Haller, J.D. Dougherty, and S.H. Rodger, Eds. ACM, 2007, 19–23; <http://dblp.uni-trier.de/db/conf/sigcse/sigcse2007.html#BlumC07>.
4. Thompson, D., Bell, T., Andreae, P., and Robins, A. The role of teachers in implementing curriculum changes. In *Proceedings of IGCSE 2013* (Denver, CO, 2013), 245–250.

Tim Bell (tim.bell@canterbury.ac.nz) is a professor in the Department of Computer Science and Software Engineering at the University of Canterbury in Christchurch, New Zealand.

Copyright held by Author/Owner(s).

Inside Risks

An Integrated Approach to Safety and Security Based on Systems Theory

Applying a more powerful new safety methodology to security risks.

DESPITE AN ENORMOUS amount of effort and resources applied to security in recent years, significant progress seems to be lacking. Similarly, changes in engineering are making traditional safety analysis techniques increasingly less effective. Most of these techniques were created over 50 years ago when systems were primarily composed of electromechanical components and were orders of magnitude less complex than today's software-intensive systems. New, more powerful safety analysis techniques, based on systems theory, are being developed and successfully used on a large variety of systems today, including aircraft, spacecraft, nuclear power plants, automobiles, medical devices, and so forth.² Systems theory can, in the same way, provide a powerful foundation for security. An additional benefit is the potential for creating an integrated approach to both security and safety.

The Relationship Between Safety and Security

Practitioners have traditionally treated safety and security as different system properties. Both communities generally work in isolation using their respective vocabulary and frameworks. Safety experts see their role as preventing losses due to *unintentional* actions by *benevolent* actors. Security experts see their role as preventing



Control room of a nuclear power plant.

losses due to *intentional* actions by *malevolent* actors. The key difference is the intent of the actor that produced the loss event. It may never be possible to determine this intent—but if the majority of our energy and analysis is refocused on building better loss prevention strategies (regardless of actor

intent), then it may not matter. We are not suggesting that intent need not be considered, only that the problem can be reframed as a general loss prevention problem that focuses on the aspects of the problem (such as the system design) that we have control over rather than immediately jumping to

the parts about which we have little information, such as identifying all the potential external threats.

Note the common goal of mission assurance here, that is, the ability to complete a mission while enforcing constraints on how the mission can be achieved. In a nuclear power plant, for example, the goal is to produce power while preventing the release of radioactivity. The causes for not producing the power or for releasing radioactivity may be due to accidental or malicious reasons, but the high-level goal of preventing these events is the same.

By taking a common top-down, system engineering approach to security and safety, several benefits accrue. One is that the overall role of the entire socio-technical system as a whole in achieving security and safety can be considered, not just low-level hardware or operator behavior. Others include more efficient use of resources and the potential for resolving conflicts between safety and security early in the development process.

Applying systems theory and systems engineering to security requires initially focusing security on high-

Tactics are prudent means to accomplish a specific action (such as guarding networks and other information assets).

level strategy rather than immediately jumping to the tactics problem. Certainly adversary action is a critical consideration in addressing security and preventing intentional losses. Yet, focusing on adversaries or threats too early in the process, absent the benefit of context, limits the overall strategic-level utility of the security assessment. Stated another way, the goal of security is not to guard the physical network and prevent intrusions, which is threat focused. The goal is to ensure the critical functions and ultimately the services that the network and systems provide are maintained in the face of disruptions. By changing to a strategic viewpoint rather than starting with tactics, security analysts and defenders can proactively shape the situation by identifying and controlling system vulnerabilities rather than defending from a position of disadvantage by being forced to react to continually changing threats and other environmental disruptions.

Strategy vs. Tactics in Security

The security field tends to draw heavily on language, metaphors, and models from military operations. As a result, much of cybersecurity is typically framed as a battle between intelligent, adaptive adversaries and defenders. Security focuses on how defenders can close holes in their networks that might otherwise allow adversaries to gain access and create disruptions. Defenders apply best practices (tactics) in order to protect the network and other information assets.

There is an important distinction between tactics and strategy. Strategy can be considered as the art of gaining and maintaining continuing advantage.

On the other hand, tactics are prudent means to accomplish a specific action (such as guarding networks and other information assets). Tactics is focused on physical threats, while strategy is focused on abstract outcomes.

In tactics models, losses are conceptualized as specific events *caused* by threats. For example, a security incident consisting of a data breach with an accompanying loss of customer Personally Identifiable Information (PII) is viewed as a single occurrence, where an adversary successfully precipitates a chain of events leading to a loss. The chain of events typically translates into attackers successfully negotiating several layers of defenses such as firewalls and encryption. In almost all such cases, security analysts will identify some proximate cause that should have served as the last barrier or line of defense. If only the barrier would have been in place, then the attack would have failed. Although threats exploiting vulnerabilities produce the loss event, tactics models treat the threat as the *cause* of the loss.

Preventing losses, then, is heavily dependent on the degree to which security analysts can correctly identify potential attackers—their motives, capabilities, and targeting. Once equipped with this knowledge, security experts can analyze their systems to determine the most likely route (or causal chain) attackers may take to achieve their goal. Resources can then be allocated to erect a “defense in depth” to prevent losses.

Threat prioritization is also challenging given the sheer volume of threats. If the defense is optimized against the wrong threat, then the barriers may be ineffective. Perhaps an unstated assumption is that defense against the more sophisticated threats can handle so-called lesser-included cases, but this is not necessarily the case. Simple requirements errors or operational procedures may allow even unsophisticated attacks from previously ignored or lower-level adversaries to succeed.

In contrast to a tactics-based, bottom-up approach, a top-down, *strategic* approach starts with identifying the system losses that are unacceptable and against which the system must be protected. The result is a

Tracing Trends in Steganography

How to Build a Bad Research Center

Broadening Conversations about CS Scholars and Scholarship

Using Conferences to Recruit Women into CS

Big Data in the Government Sector

Making Parallel Programs Reliable with Stable Multitasking

TaintDroid

Plus, all the latest news on a mathematical model that reads thoughts, ways to power devices by TV signals, and why video games are good for you.

small and more manageable set of potential losses stated at a high-level of abstraction. These losses likely extend beyond the physical and logical system entities into the higher-level services provided by these entities.

Rather than starting with the tactics questions of *how* best to guard the network against threats, a strategic approach begins with questions about *what* essential services and functions must be secured against disruptions and *what* represents an unacceptable loss. The “whats” will be used later to reason more thoroughly about only the “hows” that can lead to specific undesirable outcomes. The analysis moves from general to specific, from abstract to concrete. (Robinson and Levitt⁵ similarly considered abstraction layers with respect to being able to prove emergent system properties hierarchically.)

One of the most powerful ways human minds deal with complexity is by using hierarchical abstraction and refinement. By starting at a high level of abstraction with a small list and then refining that list with a more detailed list at each step (working top down), one can be more confident about completeness because each of the longer lists of causes (refined hazards or causes) can be traced to one or more of the small starting list (and vice versa).

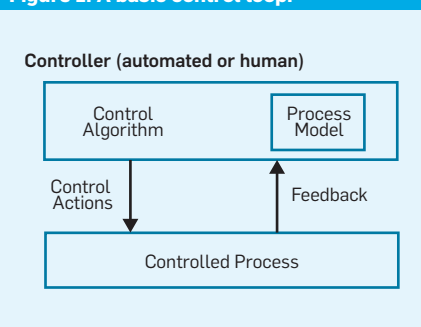
With traceability, it is also easier for human reviewers to find any incompleteness. We say “more confident” because such a list can never be proven to be complete—there is no formal (mathematical) model of the entire system and how it will operate. Human participation in the analysis and human review of the results will always be required and, therefore, incompleteness will always be possible. But structuring the process in a way that optimizes human processing and review will reduce any potential incompleteness.

Focusing first on strategy rather than tactics can be achieved by adopting a new systems-theoretic causality model recently developed to provide a more powerful approach to engineering for safety.

A New Systems-Theoretic Approach to Security and Safety

The limitations of traditional engineering methods and the need to field

Figure 1. A basic control loop.



increasingly complex systems during and immediately following World War II led to the development of modern systems theory in the 1940s and 1950s.¹ Systems theory provides the philosophical and intellectual foundation for systems engineering and for a new, more inclusive model of accident causality called STAMP (System-Theoretic Accident Model and Processes).²

Traditional causality models used in safety attribute accidents to an initial component failure or human error that cascades through a set of other components. One way to envision this model is as a set of dominoes. At one end is the initial domino, which is representative of a single human error or component failure. This initial error is labeled as the root cause. The failure propagates through the system, leading to the failure of other components until the last domino falls and the loss occurs. In this model, the first domino causes the last domino to fall (the actual loss event). Moreover, if any of the intervening dominoes are removed, the chain is broken.

This model is effective for systems with limited complexity, for example, linear interactions and simple cause-and-effect linkages like dominos (or holes in Swiss cheese, another common analogy).

Today’s increasingly complex, software-intensive systems, however, are exhibiting new causes of losses, such as accidents caused by unsafe interactions among components (none of which may have failed), system requirements and design errors, and indirect interactions and systemic factors leading to unidentified common-cause failures of barriers and protection devices. Linear causality models and the tools built upon them, like fault trees, simply lack the power to include these new causes of losses.

STAMP is a new systems-theoretic model of causality related to emergent system properties. It was originally created to act as a foundation for more powerful approaches to safety. Security, however, is also an emergent system property, and STAMP and its associated analysis tools are equally applicable to security. STAMP envisions losses as resulting from interactions among humans, physical system components, and the environment that lead to the violation of safety constraints. The focus shifts from “preventing failures” to “enforcing safety constraints on system behavior.” While enforcing safety constraints may require handling component failures, other inadvertent and advertent causes must also be controlled.

Constraints on system behavior are enforced by controls in a hierarchical control structure, where each level of the structure enforces the required constraints on the behavior of the components at the next lower level. Control loops operate between each level of this control structure, with control actions shown on the downward arrows and feedback on the upward arrows. Figure 1 shows the general form of such control loops. In both safety and security, the goal is to prevent (constrain) control actions that can lead to losses under worst-case environmental conditions.³

In systems and control theory, every controller must contain a model of the process it is controlling. This model is used to determine what control actions are necessary. Many accidents related to software or human operators are not the result of software or human “failure” (whatever that might mean), but instead stem from inconsistencies between the controller’s models of the controlled process (usually called a mental model for human controllers) and the actual process state. For example, friendly fire accidents are usually the result of thinking a friendly aircraft is an enemy and executing unsafe control actions. Whether the inconsistency results from an inadvertent reason (accidental loss of feedback, for example) or tricking the controller into thinking that the friendly aircraft is an enemy (purposeful creation of incorrect feedback), the result remains the same—an unsafe or unwanted control action.

Stuxnet provides another example. The automated system (controller) thought the centrifuges (controlled process) were spinning at a slower speed than they actually were, and issued an *Increase Speed* command when the centrifuges were already spinning at maximum speed, which led to equipment damage. (A loss that officials probably wanted to prevent.)

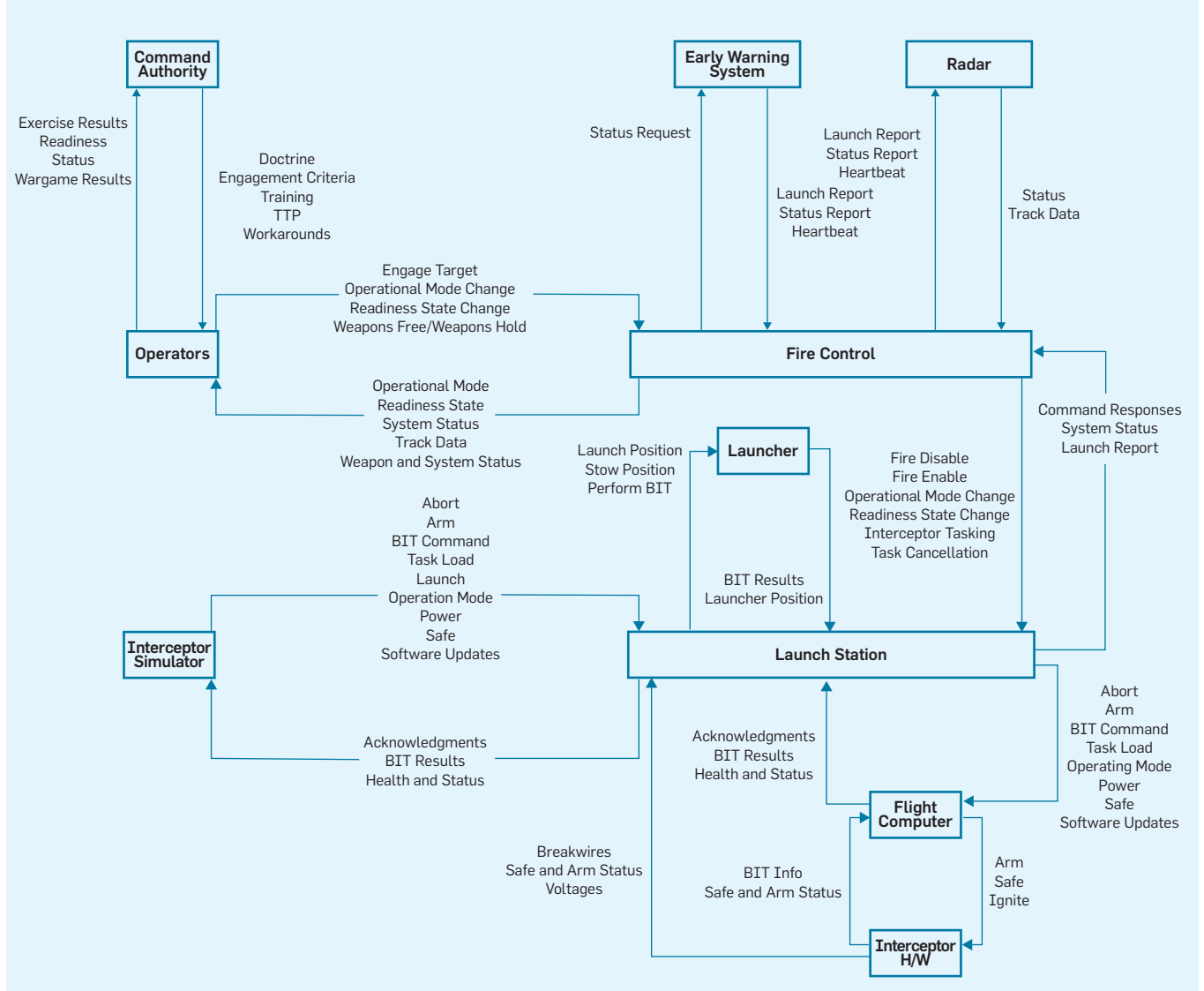
New and more powerful techniques for safety analysis and design have been created on this theoretical foundation. STPA (System-Theoretic Process Analysis), for example, is a new hazard analysis technique based on the STAMP model of causality. The analysis is performed on the system functional control structure. Figure 2 depicts an illustrative functional control structure for a ballistic-missile de-

fense system.^{2,4} In this example, there are several safety and security critical control commands, such as *fire enable* and *launch interceptor*.

One key point worth emphasizing is the fact that the function control model contains physical aspects, social aspects, logical and information aspects, operations and management aspects. Performing the hazard (safety) or vulnerability (security) analysis on such a model allows a broad perspective on potential causes for a loss. Most hazard and vulnerability analysis techniques use physical system models rather than functional system models, and thus concentrate on physical component failures rather than dysfunctional (unsafe or insecure) system behavior and broader social and organizational factors.

Once the control structure is created, the first step in the STPA analysis is to identify potentially unsafe control actions, which in general include (1) providing a control action that leads to a hazard (for example, a missile is launched at a friendly aircraft), (2) not providing a control action that is needed to prevent a hazard (for example, a missile is not launched to down an enemy aircraft), (3) providing a control action too early or too late or out of sequence (for example, a missile is launched but too early or too late to be effective in preventing a loss), or (4) continuing a control action too long or stopping it too soon. Losses can also result from a safe (required) control action that is not executed properly (for example, the launch missile instruction is not executed correctly). After

Figure 2. Functional control structure for a ballistic missile defense system.



the unsafe control actions have been identified, the second step involves examining the system control loops (using a structured and guided process) to identify scenarios that can lead to the identified unsafe control actions.

STPA-Sec is an extension to STPA to include security analysis. The initial steps in the analysis are identical to those for safety: identifying the losses to be considered, identifying system hazards or security vulnerabilities, drawing the system functional control structure, and identifying unsafe, or in this case, insecure, control actions. The only difference is the addition of intentional actions in the generation of the causal scenarios, the last step in the process.

STPA is currently being used on safety problems in a wide variety of industries. Careful evaluations and comparisons with traditional hazard analysis techniques have found that STPA finds the loss scenarios found by the traditional approaches (such as Fault Tree Analysis and Failure Modes and Effects Analysis) as well as many more that do not involve component failures. Surprisingly, while STPA is more powerful, it also appears to require fewer resources, including time.

STPA-Sec is only now being applied to cybersecurity problems, but is showing promise in these case studies. A formal evaluation and comparison with real red teams using traditional security analysis techniques such as attack trees will be completed by spring 2014.

Another benefit of using a tool based on a system-theoretic model is that it can be applied earlier in the design process and in situations where specific component data is unavailable. Analysis can begin as soon as the basic high-level goals (mission) of the system is identified and design decisions evaluated for their impact on safety and security before expensive rework is necessary. As the detailed design decisions are made and the design refined, the STPA/STPA-Sec analysis is refined in parallel.

Conclusion

By using a causality model based on systems theory, an integrated and more powerful approach to safety and security is possible. Hazards lead to safety incidents in the same way that vulnerabilities lead to security incidents. We

The key question facing security analysts should be how to control vulnerabilities, not how to avoid threats.

argued in this column that the key question facing security analysts should be how to control vulnerabilities, not how to avoid threats. Rather than initially trying to identify all the threats and then move up to the vulnerabilities they might exploit to produce a loss, a top-down systems engineering approach starts with system vulnerabilities, which are likely far fewer than threats and, if controlled, can prevent losses due to numerous types of threats and disruptions. This top-down approach also elevates the security problem from guarding the network to the higher-level problem of assuring the overall function of the enterprise.

Use of a systems-theoretic approach to security, however, requires a reframing of the usual security problem. Just as STAMP reframes the safety problem as a control rather than a failure problem, applying STAMP to security involves reframing the security problem into one of strategy rather than tactics. In practice, this reframing involves shifting the majority of security analysis away from guarding against attacks (tactics) and more toward design of the broader socio-technical system (strategy). Put another way, rather than focusing the majority of the security efforts on threats from adversary action, which we have limited control over, security efforts should be focused on the larger, more inclusive goal of controlling system vulnerabilities.

Controlling vulnerabilities allows security analysts to prevent not only disruptions from known threats, but also disruptions introduced by unknown threats, such as insiders. In other words, the source of the disruption does not matter. What matters is identifying and controlling the vulner-

abilities. This approach limits the intelligence burden required to perform the initial system security analysis. The analysis will eventually address threats, but does so much later in the process after generating a deeper systemic understanding of the context under which the threats may operate and the disruptions that actually lead to critical loss events.

Because contemporary security and safety both attempt to prevent losses in complex software-controlled systems, we believe applying the same system-theoretic causality model may benefit security the same way it is benefitting safety. Research is currently under way to test this notion. The key underlying idea is that from a strategy perspective, the physical (or proximate) cause of a disruption does not really matter. What matters is the efficacy of the strategy in dealing with (controlling) the effects of that disruption on overall system function or assuring the mission. This is a significant paradigm shift for security experts (as it was for safety experts). While likely to force a reexamination of many of the accepted truths of security, we believe such a refocus will help address three of the major problems with contemporary approaches to security—quantity, threat variety, and threat prioritization—can all be addressed more effectively through this new approach than through existing approaches. The new approach does not discard traditional security thinking, but does suggest it is tactically focused and must be augmented by an effective strategy in order to succeed. 

References

1. Checkland, P. *Systems Thinking, Systems Practice*. John Wiley & Sons, New York, 1981.
2. Leveson, N.G. *Engineering a Safer World*. MIT Press, 2012.
3. Leveson, N. and Thomas, J. *An STPA Primer*; <http://sunnyday.mit.edu/STPA-Primer-v0.pdf>
4. Pereira, S.J., Lee, G. and Howard, J. A system-theoretic hazard analysis methodology for a non-advocate safety assessment of the ballistic missile defense system. In *Proceedings of the 2006 AIAA Missile Sciences Conference* (Monterey, CA, Nov. 2006).
5. Robinson, L. and Levitt, K.N. Proof techniques for hierarchically structured programs. *Commun. ACM* 20, 4 (Apr. 1977), 271–283.

William Young (wyoung@mit.edu) is a Ph.D. candidate in the Engineering Systems division at Massachusetts Institute of Technology, Cambridge, MA.

Nancy G. Leveson (leveson@mit.edu) is Professor of Aeronautics and Astronautics and also Professor of Engineering Systems at Massachusetts Institute of Technology, Cambridge, MA.

Copyright held by Author/Owner(s).



DOI:10.1145/2556939

George V. Neville-Neil

Article development led by **acmqueue**
queue.acm.org

Kode Vicious Bugs and Bragging Rights

It is not always size that matters.

Dear KV,

I have been dealing with a large program written in Java that seems to spend most of its time asking me to restart it because it has run out of memory. I am not sure if this is an issue in the JVM (Java Virtual Machine) I am using or in the program itself, but during these frequent restarts, I keep wondering why this program is so incredibly bloated. I would have thought Java's garbage collector would prevent programs from running out of memory, especially when my desktop has quite a lot of it. It seems that eight gigabytes just is not enough to handle a modern IDE anymore.

Lack of RAM

Dear Lack,

Eight gigabytes?! Is that all you have? Are you writing me from the desert wasteland where PCs go to die? No one in his or her right mind runs a machine with less than 48GB in our modern era, at least no one who wants to run certain, very special, pieces of Java code.

While I would love to spend several hundred words bashing Java—for, like all languages, it has many sins—the problem you are experiencing is probably not related to a bug in the garbage collector. It has to do with bugs in the code you are running, and with a certain, fundamental bug in the human mind. I will address both of these in turn.

The bug in the code is easy enough to describe. Any computer language that takes the management of memory out of the hands of the programmer and puts it into an automatic garbage-collection system has one fatal flaw: the programmer can easily prevent the garbage collector from doing its work. Any object that continues to have a reference cannot be garbage collected, and therefore freed back into the system's memory.

Sloppy programmers who do not free their references cause memory leaks. In systems with many objects (and almost everything in a Java program is an object) a few small leaks can lead to out-of-memory errors quite quickly. These memory leaks are difficult to find. Sometimes they reside in the code you, yourself, are working on, but often they reside in libraries that your code depends on. Without access to the library code, the bugs are impossible to fix, and even with access to the source, who wants to spend their time fixing memory leaks in other people's code. I certainly don't. Moore's Law often protects fools and little children from these problems, because while frequency scaling has stopped, memory density continues to increase. Why bother trying to find that small leak in your code when your boss is screaming to ship the next version of whatever it is you are working on? "The system stayed up for a whole day, ship it!"

The second bug is far more pernicious. One thing you did not ask was,



ILLUSTRATION BY IWONA USAKIEWICZ/ANDRIJ BORYS ASSOCIATES

“Why do we have a garbage collector in our system?” The reason we have a garbage collector is because sometime in the past, someone—well, really, a group of someones—wanted to remedy another problem: programmers who could not manage their own memory. C++, another object-oriented language, also has lots of objects floating around when its programs execute. In C++, as we all know, objects must be created or destroyed using new and delete. If they are not destroyed, then we have a memory leak. Not only must the programmer manage objects, but in C++, the programmer can also get direct access to the memory that underlies the object, which leads naughty programmers to touch things they ought not to. The C++ runtime does not really say, “Bad touch, call an adult,” but that is what a segmentation fault really means. Depending on your point of view, garbage collection was promulgated either to free programmers from the tedium of managing memory by hand or to prevent them from doing naughty things.

The problem is that we traded one set of problems for another. Before garbage collection, we would forget to delete an object, or double delete it by mistake; and after garbage collection, we had to manage our references to objects, which, in all honesty, is the exact same problem as forgetting to delete an object. We traded pointers for references and are none the wiser for it.

Longtime readers of KV know that silver bullets never work, and that one has to be very careful about protecting programmers from themselves. A side effect of creating a garbage-collected language was the overhead of having the virtual machine manage memory was too high for many workloads. The performance penalty has led to people building huge Java libraries that do not use garbage collection and in which the objects must be managed manually, just as they did with languages such as C++. When one of your key features has such high overhead that your own users create huge frameworks that avoid that feature, something has gone terribly wrong.

The situation as it stands is this: with a C++ (or C) program, you are more likely to see segmentation faults and memory-smashing bugs than you

Programmers never stop comparing their code with the code of their peers.

are to see out-of-memory errors on a modern system with a lot of RAM. If you are running something written in Java, then you had better pony up the cash for all the memory sticks you can manage because you are going to need them.

KV

Dear KV,

I cannot help but notice that a lot of large systems call themselves “operating systems” when they really do not bear much resemblance to one. Has the definition of operating system changed to the point where any large piece of software can call itself one?

OS or Not OS

Dear OS,

Certainly my definition of operating system has not changed to the point where any large piece of software can call itself one, but I have also spotted the trend. An old joke is that every program grows in size until it can be used to read email, which, if you can believe Wikipedia, is attributed to Jamie Zawinski, based on an earlier joke by Greg Kuperberg, “Every program in development at MIT expands until it can read mail.” Now, it seems, mail is not enough. Every large program expands until it gets “OS” appended to its name.

An operating system is a program that is used to give efficient access to an underlying piece of hardware, hopefully in a portable manner, though that is not a strict requirement. The purpose of the software is to provide a consistent set of APIs to programmers such that they do not

need to rewrite low-level code every time they want to run their programs on a new computer model. That may not be what the *Oxford English Dictionary* defines as an OS, but as it recently added “selfie” to its dictionary and named it word of the year for 2013, I am starting to think a bit less of the quality of their output, anyway.

I think the propensity for programmers to label their larger creations as operating systems comes from the need to secure bragging rights. Programmers never stop comparing their code with the code of their peers. The same can be seen even within actual operating-system projects. Everyone seems to want to (re)write the scheduler. Why? Because to many programmers, it is the most important piece of code in the system, and if they do a great job, and the scheduler runs really well, they will give their peers a good dose of coder envy. Never mind that the scheduler really ought to be incredibly small, and very, very simple, but that is not the point. The point is the bragging rights one gets from having rewritten it, often for the umpteenth time.

None of this is meant to belittle those programmers or teams of programmers who have slaved long and hard to produce elegant pieces of complex code that make our lives better. If you look closely, though, you will find that those pieces of code are appropriately named, and they do not need to tack on an OS to make them look bigger.

KV

Q Related articles
on queue.acm.org

Reveling in Constraints

Bruce Johnson

<http://queue.acm.org/detail.cfm?id=1572457>

Gettin' Your Kode On

George Neville-Neil

<http://queue.acm.org/detail.cfm?id=1117397>

Self-Healing in Modern Operating Systems

Michael W. Shapiro

<http://queue.acm.org/detail.cfm?id=1039537>

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

Copyright held by Owner/Author(s).

Economic and Business Dimensions

Digital Platforms: When Is Participation Valuable?

Assessing the benefits and challenges of knowledge spillovers.

SMART OWNERS OF information technology (IT) platforms develop ecosystems and encourage third-party producers to develop complements—products that run on the platform. Independent software vendors develop complements for the platforms of major vendors (for example, SAP, Apple). In addition, such platform companies sometimes create Internet-based knowledge-sharing communities in which users exchange ideas. The benefits and costs of such participation are tied to what economists call “knowledge spillovers.”

This column reports on four studies of complements to SAP’s flagship enterprise resource planning software.^{1–4} Users can participate in the SAP Developer Network (SDN), a knowledge-sharing community that encourages voluntary knowledge exchange regarding the implementation, use, and customization of SAP software. A study of 275 firms participating in SDN between 2004 and 2008 showed firms using the platform’s online question-and-answer forum had significantly higher productivity. Knowledge spillovers were at work. Valuable knowledge gained from investments in SAP by user firms was transferred to others through the online forum, helping the diffusion of best practices related to the platform. Preliminary estimates suggest a 1%

increase in such inward knowledge spillovers will increase production output by many thousands of dollars.

While knowledge spillovers benefit software users, they present challenges for producers. SAP bolsters development of third-party software products and complements by encouraging start-ups to ensure their products are compatible with the platform and to advertise this fact through formal certification. Small start-ups receiving SAP certification had higher sales and a greater likeli-

hood of issuing an initial public offering. Complements and platforms in enterprise software are tightly coupled, and applications developed and used without formal endorsement from the platform owner might not work well. Platform certification signals compatibility and higher quality. The study, using data from 1996–2004, looked at small start-up firms before and after they joined SAP’s certification program.

While start-ups benefited significantly from certification, some start-



ILLUSTRATION BY SEBASTIAN KAULITZKI

ups benefited more than others. Firms with the greatest gains also had strong intellectual property protection from patents, copyrights, and trademarks, the latter being a common indicator of brand strength. Again, knowledge spillovers were at work. Certification requires start-ups to go through a process of documentation and testing. Signaling compatibility with the SAP platform has benefits for the start-up, but poses risks of unintended knowledge spillovers leading to possible competition from SAP itself. SAP's early website portal stated, "Part of being an open ecosystem is open and fair competition among partners, and between SAP and partners. SAP cannot guarantee exclusivity of partner solutions, nor can we guarantee that we won't offer competing solutions."

Large platform companies like SAP realize their possible entry into complementary markets can dissuade complementors from joining the platform. However, the intellectual property asset holdings by start-ups help mitigate this problem. Protected start-ups are more likely to join the platform and tend to do so earlier, and large companies can face higher costs in competing with start-ups that hold intellectual property assets.^a

Intellectual property rights thus play a dual role in shaping platform growth. On the one hand, strong intellectual property rights can help protect small companies, reducing the threat of imitation and entry by a platform owner, and increasing the value of the platform and benefit the platform owner. On the other hand, intellectual property rights in the form of "patent thickets" (dense, overlapping webs of intellectual property rights held by incumbents) can slow platform growth by raising complementors' costs of potential patent infringement. Patent thickets make it easier to inadvertently infringe on intellectual property rights of other firms in the marketplace, a recent example being the "smartphone war" between Apple, Samsung, and other firms. Patent thickets might be a more serious problem for small, entrepreneurial firms that have few intellectual assets of their own and that

While start-ups benefited significantly from certification, some start-ups benefited more than others.

are unable to navigate patent thickets by negotiating patent cross-licensing deals with other firms.

Managers of technology platforms can bet that potential complementors might refuse to join a platform because of the risks of platform owner entry. Those platforms can alleviate complementor risks by giving up control of the platform or "opening" it up to encourage participation. Such approaches are less essential when strong IP rights protect complementors from the consequences of unwanted spillovers to potential competitors. **C**

References

1. Benbya, H. and Van Alstyne, M. How to find answers within your company. *Sloan Management Review* 52, 2 (Feb. 2011), 65–75; <http://sloanreview.mit.edu/article/how-to-find-answers-within-your-company/>
2. Ceccagnoli, M., Forman, C., Huang, P., and Wu, D.J. Cocreation of value in a platform ecosystem: The case of enterprise software. *MIS Quarterly* 36, 1 (Jan. 2012), 263–290.
3. Huang, P., Ceccagnoli, M., Forman, C., and Wu, D.J. Appropriability mechanisms and the platform partnership decision: Evidence from enterprise software. *Management Science* 59, 1 (Jan. 2013), 102–121.
4. Huang, P., Ceccagnoli, M., Forman, C., and Wu, D.J. IT knowledge spillovers and productivity: Evidence from enterprise software. Working Paper. University of Maryland and Georgia Institute of Technology, 2013; http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2243886

Marco Ceccagnoli (marco.ceccagnoli@scheller.gatech.edu) is an associate professor of strategic management in the Scheller College of Business at the Georgia Institute of Technology in Atlanta.

Chris Forman (Chris.Forman@scheller.gatech.edu) is the Brady Family Term Professor in the Scheller College of Business at the Georgia Institute of Technology in Atlanta.

Peng Huang (huang@umd.edu) is an assistant professor in the Decisions, Operations and Information Technologies Department at the Robert H. Smith School of Business at the University of Maryland.

D.J. Wu (dj.wu@scheller.gatech.edu) is an associate professor in Information Technology Management in the Scheller College of Business at the Georgia Institute of Technology in Atlanta.

Copyright held by Author/Owner(s).



ACM

Transactions on

Accessible Computing



◆ ◆ ◆ ◆ ◆

This quarterly publication is a quarterly journal that publishes refereed articles addressing issues of computing as it impacts the lives of people with disabilities. The journal will be of particular interest to SIGACCESS members and delegates to its affiliated conference (i.e., ASSETS), as well as other international accessibility conferences.

◆ ◆ ◆ ◆ ◆

www.acm.org/taccess
www.acm.org/subscribe



**Association for
Computing Machinery**

^a SAP. Powered by SAP Netweaver Partner Program FAQs; <http://bit.ly/J7UYgc>.

Viewpoint

Ready Technology

Fast-tracking emerging business technologies.

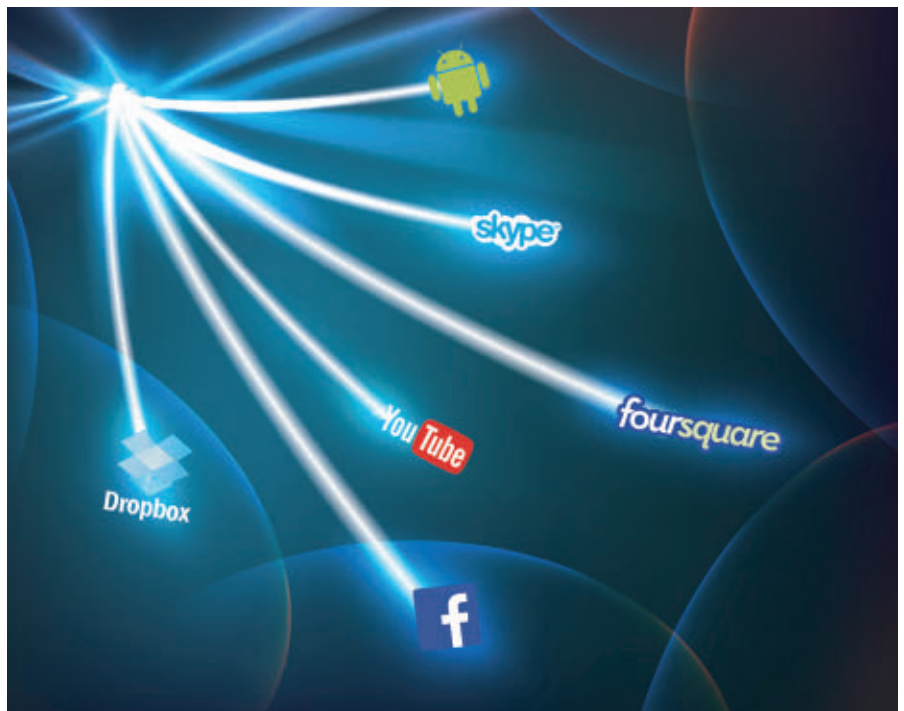
IN THE 20TH century, companies waited until their industries and competitors fully vetted technologies before investing in even the most tried-and-true ones. Technophobes believed that investing too early was indulgent and reckless. Executives wore their late technology adoption strategies as badges of corporate honor.

Today, many emerging technologies are ready for immediate deployment.^a iPads are ready. Dropbox is ready. Skype is ready. ListenLogic is ready. Foursquare is ready. Ready technology is accessible and cost-effective. It also often arrives at companies without the participation of the corporate IT team, especially in federated or decentralized companies where business units and employees are encouraged to solve their own problems. The accompanying figure summarizes defined and ready technology adoption—and the implications of ready technology adoption. It also provides some examples of ready technology.

Technology Adoption

Defined Adoption. The 20th-century technology adoption models were predicated on the diagnosticity of business requirements and technology maturity. The assumption was that technology and business requirements evolve at a pace that justifies phased adoption. Early deployments were assumed to be risky, costly, and therefore unnecessary.

^a Ready technology is easily accessible technology, requires minimal support, and is mature enough to make immediate—and major—problem-solving contributions.



Defined and validated business requirements were prized. An enormous industry was created around requirements analysis, requirements modeling, and requirements validation. Books, articles, conferences, and workshops were everywhere. The prevailing wisdom was that business requirements modeling and validation were prerequisites to technology adoption, and that structured pilot demonstrations with compelling TCO and ROI results were necessary to justify deployment. Technology also had to integrate and interoperate with existing technology infrastructures and architectures. If it failed to cost-effectively integrate, adoption was often halted. If it did integrate, then a structured transition

period was defined to test and deploy the new technology before the technology went into production. Finally, new technology—just like old technology—required continuous support and expensive refreshes.

Ready Adoption. Technology adoption is different today.^b Requirements

^b Not all technology adoption falls in the “ready” category. There are still defined adoption projects that assume the value of prolonged requirements analyses, testing, and phased deployment. Major enterprise applications like ERP, CRM, and network and systems management, for example, fall into the defined category. But not all technology is created equal: some—*ready*—technology can go to work without prolonged requirements analysis, testing, and phased deployment.

are often undefined and driven by employees—consumers who adopt technologies to solve a variety of problems with technologies that are acquired—and sometimes even supported—way outside the corporate firewall. Consumer-driven requirements analysis, exploration, and discovery is the mainstay of ready technology adoption. Note also that what was previously described as controlled pilots are largely ad hoc opportunistic experiments that often quickly turn into technology deployments—with or without the approval of corporate IT departments.

Examples. The figure here lists a variety of ready technologies already at work solving a variety of problems across multiple vertical industries. The new governance process is significant because it often bypasses corporate IT and the policies and procedures aligned to specific vertical industries, like manufacturing, pharmaceuticals, and financial services. In fact, every industry is ready—though must keep compliance and security in mind as it officially—or unofficially—adopts new technology.

The list of ready technologies includes the following—but note this is not an exhaustive list and is representative of the growing number of ready technologies companies are quickly adopting:

- ▶ BYO: Devices, applications, data, and so forth.
- ▶ Tablets (such as iPads).

Ready technology adoption unleashes the power of emerging technology as early as possible.

- ▶ Smartphones (such as iPhones).
- ▶ Content Sharing (such as with Dropbox).
- ▶ Mobile and Other Applications (from App Stores).
- ▶ Social Networking (with, for example, ListenLogic).
- ▶ Video-Teleconferencing (with Facetime, Skype).
- ▶ Video Sharing and Marketing (with YouTube).
- ▶ Location Awareness (with Foursquare).

At Shire Pharmaceuticals, for example, ready technology found its way into the trenches through the C-suite: Shire professionals adopted iPhones, iPads, Skype, ListenLogic, and Dropbox before corporate IT could assess their reliability, security, or TCO/ROI. Hundreds of iPads were deployed at Shire before corporate IT declared them “non-standard” and

unsecure. Corporate IT chased them around, but everyone got to keep their iPads (and other devices) when IT ultimately declared them “safe”—well after their deployment.^c Similar events occurred at Balfour Beatty, Luxottica, and RehabCare.^d The same process is playing out in banks, consultancies, and retailers. In fact, there is no way to stop the process—as many companies discovered when they tried to ban social networks.^e

The deployment of iPads, social media, file sharing, and video-teleconferencing (among other technologies) at Shire and other companies demon-

^c There are many stories about how iPads found their way into the enterprise—without the approval of corporate IT departments. See T. Kaneshige, “iPads in the Enterprise: IT Must Stay Ahead of the Curve,” *CIO Magazine* (May 1, 2012), (www.cio.com/article/705379/); and S. Ludwig, “The iPad is an Incredible Tool for Work—If Your IT Department Will Allow It,” *VB Mobile* (Jan. 4, 2012). The Shire case is documented in S.J. Andriole, “The Transformation of Technology at Shire Pharmaceuticals,” the Acentio Group, December 2012.

^d See M. Rosoff, “Huge Construction Firm Uses iPads and Apple TV to Save Millions,” *CITE-world* (Mar. 28, 2013), and S. Ludwig (<http://bit.ly/1hP0NhP>).

^e See J. Bughin and M. Chui, “How Social Technologies are Extending the Organization,” *McKinsey Quarterly* (Apr. 2011); (<http://bit.ly/JiBPJr>). Also see: “54% of Companies Ban Facebook, Twitter at Work,” *ComputerWorld* (Oct. 6, 2009); and K. Bhasin, “Companies Around The World Are Banning Social Media Sites At Work More Than Ever,” *Business Insider* (Sept. 6, 2011); <http://bit.ly/JaKnbV>.

Ready technology.

Defined Adoption

- Defined Business-Driven Requirements Analysis and Validation
- Full Technology Pilot Demonstrations Prior to Deployment
- Required Integration of New Technology into Existing Technology Architectures
- Transition Period to Test and Integrate New Technology
- Continuous Support and Refresh Requirements

Ready Adoption

- Defined and Undefined Consumer-Driven Requirements Analysis and Exploration
- Uncontrolled, Ad Hoc Technology Pilots
- Limited or No Integration of New Technology into Existing Technology Architectures
- Immediate Adoption and Delivery Through Cloud Providers
- Limited Support and Refresh Requirements

Implications

- Accelerated Technology Adoption
- Increase in “Fail Fast/Fail Cheap” Pilots
- Rapid Technology-Driven Business Process Change
- Improved Technology TCO/ROI, Especially Through the Avoidance of Large Integration, Support and Refresh Costs
- Major Changes in Corporate Governance of Information Technology

Examples

- BYO: Devices, Applications, Data, and so forth.
- Tablets (such as iPads)
- Smartphones (such as iPhones and Androids)
- Content/File Sharing (with, for example, Dropbox)
- Mobile Applications (from App Stores)
- Social Networking (with Facebook, Twitter, Flickr, and so forth)
- Video-Teleconferencing (with Facetime, Skype)
- Video Sharing and Marketing (with YouTube)
- Location Awareness (with, for example, Foursquare)

ACM Journal on Computing and Cultural Heritage



JOCCH publishes papers of significant and lasting value in all areas relating to the use of ICT in support of Cultural Heritage, seeking to combine the best of computing science with real attention to any aspect of the cultural heritage sector.

www.acm.org/jocch
www.acm.org/subscribe



Association for
Computing Machinery

strates just how fast technology can be adopted. (In fact, 93% of Fortune 100 companies adopted iPads immediately after they were introduced.^f) Fast creative deployments legitimized iPads as solutions to an array of well-known and yet-to-be-discovered problems. No one took the position that the first iPad was probably half-baked and that it made sense to wait for the iPad 2, 3, or 4.

The rapid adoption of social media listening technology is another example. Once it was possible to listen to all flavors of social conversations, companies quickly found listening partners (like ListenLogic and Radian6 [now part of salesforce.com]), and started mining social data about what their customers liked and disliked about their products and services.^g

Dropbox is cloud file sharing. Countless professionals use Dropbox to store and share files of all kinds. Skype and Facetime are ready technologies used extensively for collaboration and communication—even at companies with expensive proprietary video conferencing (VTC) systems. App stores are not the stores of last resort, but often the first stores visited by professionals with problems to solve. Mobile application development is also exploding. Foursquare is ready for location-based services, and YouTube for video sharing for training and marketing.

Implications. The first implication of ready technology adoption is speed. Ready technology adoption unleashes the power of emerging technology as early as possible. Put another way, ready technology adoption—as chaotic as it sometimes is—enables us to “fail fast/fail cheap”—and redefines the whole “piloting” process. It also enables rapid business process modeling (BPM) by introducing new capabilities applied to old processes—like how we collaborate through cloud file sharing (Dropbox) or how we see each other while traveling (Skype, Face-

time). But perhaps the largest implication of ready technology adoption is how it rearranges technology governance. In the 20th century, technology governance was centralized or federated. Ready technology adoption is decentralized. This means employees-consumers in business units govern technology adoption and exploit what the technologies can provide without the “guidance” of corporate or business unit CIOs. This has profound implications for the acquisition and support of enterprise technology. Ready technology also challenges our technology cost models, how we define and measure Service Level Agreements (SLAs) and ultimately how we calculate technology TCO and ROI. Ready technology—as appealing and productive as it can be—also challenges our already formidable problems around technology integration, interoperability, scalability, and support.

Conclusion

While there are still plenty of technologies that require traditional adoption processes—like big data analytics, ERP and CRM applications—there is a growing number of technologies ready to go to work immediately. Many of these technologies are cloud-based, open source, and live happily outside of corporate firewalls. Many of them are easily and inexpensively accessible to corporate professionals and will therefore continue to find their way into companies of all shapes and sizes—regardless of what CIOs think about their readiness. At the end of the day, ready technologies is upsetting just about every governance ap-
plecart at work today—as they rapidly discover, define, and solve more and more corporate problems. Rather than scramble to get all the apples back in the cart, CIOs and CTOs should rethink the way useful technology enters the enterprise and embrace the role ready technologies can play in the problem-solving process—especially since they have no choice. 

^f Apple reported 93% of Fortune 500 companies have deployed or are testing iPads; <http://bit.ly/JiC6Mv>.

^g See S.J. Andriole, V.J. Schiavone, L.F. Stevens, M. Harrington, and M. Langsfeld, *Social Business Intelligence: Reducing Risk, Managing Brands & Defining Markets with Social Media* (Ascendigm Press, 2013) for a deeper look at the business role social business intelligence can play.

Stephen J. Andriole (steve@andriole.com) is a professor in the Department of Accounting and Information Systems at the Villanova School of Business, Villanova University, PA.

LEARNING @ SCALE

MARCH 4-5 2014
ATLANTA, GEORGIA, USA

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



LEARNING @ SCALE

ACM will host the first **ACM Conference on Learning at Scale** to be held March 4-5, 2014 at the Hyatt Regency Atlanta, in Atlanta, Georgia, USA.

Inspired by the emergence of Massive Open Online Courses (MOOCs) and the shift in thinking about education, ACM created this conference as a new venue to explore how learning and teaching can change and improve when done “at scale.”

ABOUT THE CONFERENCE

ACM Learning at Scale 2014 is the first in a new conference series intended to promote scientific exchange of interdisciplinary research at the intersection of the learning sciences and computer science.

Topics of accepted papers include peer and self-assessment in MOOCs, tools for instructors, the effect of video production values on student engagement, automating grading, educational games, analyzing discussion forum behavior, and more. In addition to paper presentations, the

conference will feature posters and demonstrations of research prototypes and production tools, tutorials in natural language processing and educational analytics, panels covering topics in both the learning sciences and the practice of scalable education, and invited presentations on scalable education in the developing world.

The inaugural ACM Learning at Scale will be co-located with **SIGCSE 2014**, the annual Technical Symposium of the ACM Special Interest Group on Computer Science Education (<http://sigcse2014.sigcse.org/>). Together, these conferences will make for a great week spotlighting education!



COMMITTEE

GENERAL CHAIR

Mehran Sahami (Stanford University)

PROGRAM CHAIRS

Armando Fox (UC Berkeley)

Micheline T.H. Chi (Arizona State University)

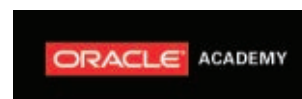
Marti Hearst (UC Berkeley)

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

L@S Corporate Support Provided By:



Microsoft Research



Article development led by **acmqueue**
queue.acm.org

**A discussion with Kiran Prasad,
Kelly Norton, and Terry Coatta.**

Node at LinkedIn: The Pursuit of Thinner, Lighter, Faster

NODE.JS, THE SERVER-SIDE JavaScript-based software platform used to build scalable network applications, has been all the rage among many developers for the past couple of years, although its popularity has also managed to enrage some others, who have unleashed a barrage of negative blog posts to point out its perceived shortcomings. Still, while new and untested, Node continues to win more converts.

In 2011, LinkedIn joined the movement when it opted to rebuild its core mobile services in Node. The professional networking site, which had been relying on Ruby on Rails, was looking for performance and scalability gains. With its pervasive use of non-blocking primitives and a single-threaded event loop, Node seemed promising.

Following the creation of Node.js in 2009 by Ryan Dahl (now at Joyent, which sponsors and maintains Node), it did not take long for developers to seize upon it. Because Node uses JavaScript, a language largely associated with the client side of Web apps, it clears the way for developers working on the client side to also work on corresponding functions over on the server side.

Kiran Prasad, who joined LinkedIn as senior director of mobile engineering in 2011, led the company's transition to Node. On the server side, LinkedIn's entire mobile front end is now built entirely in Node. Prasad admits Node isn't the best tool for every job, but upon analyzing LinkedIn's system, Prasad and his team determined that what was needed to improve efficiency was an event-driven system. Node also proved attractive because it's thin and light while also allowing for the direct manipulation of data objects.

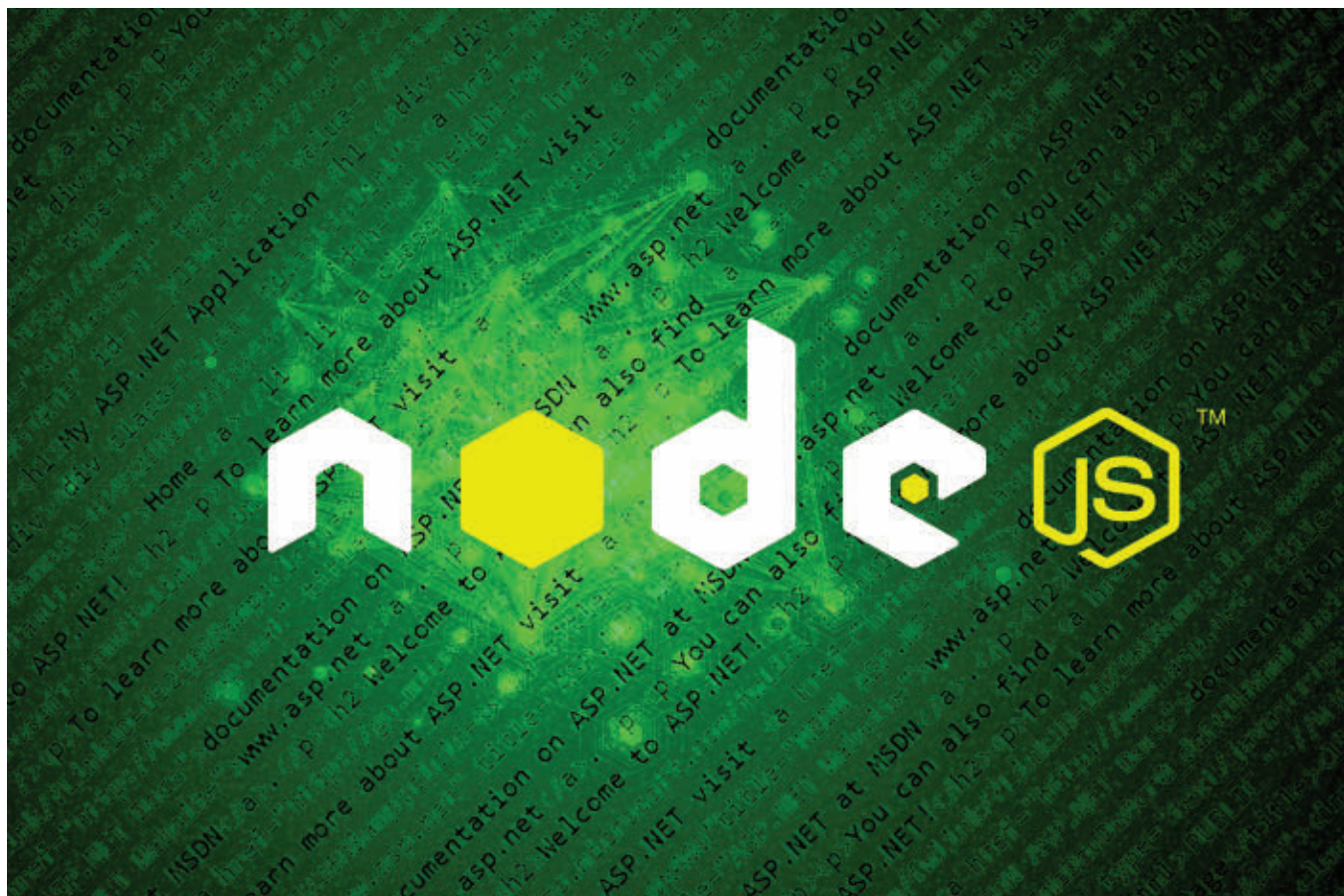
Prasad was well prepared for his role in mobile services at LinkedIn, having already accumulated years of experience in mobile apps working on the WebOS platform at Palm and Handspring in addition to stints as an independent developer of mobile Web software (as CEO at Sliced Simple and CTO at Aliaron).

He talks here about LinkedIn's adoption of Node.js with **Kelly Norton** and **Terry Coatta**. Norton was one of the first software engineers to work on the Google Web Toolkit (GWT) before cofounding Homebase.io, which develops next-generation marketing tools.

Coatta is CTO at Marine Learning Systems, which has developed a learning management system targeted at the marine industry. He previously worked for AssociCom, Vitrium Systems, GPS Industries, and Silicon Chalk.

KELLY NORTON: Tell us what went into LinkedIn's decision to use Node.js.

KIRAN PRASAD: We were running a Ruby on Rails process-based system, and it became pretty clear that



just wasn't going to scale the way we needed it to. I guess you can always scale something if you're willing to throw enough money at it, but obviously that didn't seem like the right way to go. Also, working with the mobile model—where there are lots of microconnections—we could see that a process-based approach was going to run into difficulty with the Ruby on Rails stack.

We also noticed there was a pretty big performance hit with Rails since we were doing a lot of string manipulation and the Ruby interpreter version we were using was struggling to garbage-collect all the small-string objects. Nor was it particularly optimized for JSON (JavaScript Object Notation) translation, which was what our back end was giving us, as well as what our front end was looking to consume.

Clearly, Ruby on Rails was built more as a Web stack in that its real

value lies in the templating it offers for that structure, along with some of the framework concepts it provides for the app and the controllers. But the controllers and the views actually move down to the client whenever you're doing client-side rendering, which is what happens with mobile systems.

With the larger, higher-scale stacks you find at places like LinkedIn, you also start breaking apart the model. That is, you're not really inside ActiveRecord on Rails at that point since you essentially end up moving down a level into other servers or services. This means the middle layer starts to get pretty thin and really focused on string manipulation.

So when we really started to look at that, we said, "This sure doesn't feel like such a great fit anymore. It's not designed to do what we're trying to accomplish now." So, what could we replace it with? It had to be some-

thing that was event-driven, was good at string manipulation, and was light and quick and easy to use. We started looking at some event-driven frameworks in Ruby such as EventMachine, as well as Twisted in Python. But there seemed to be no mainstream event-driven Java frameworks when we first started checking into this. While Play has become a little more prevalent now, back when we were doing our analysis, in early 2011, it didn't pop up on our radar for some reason.

NORTON: Besides being event-driven, what was so compelling about Node?

PRASAD: We looked at Node and ran some load against it, at which point it would send a request out to something like six other services, grab that data, merge it, and then pop it back out in a fairly simple way. We just ran with that all the way up to about 50,000 QPS (queries per second). Along the way we discovered that Node was roughly 20



KIRAN PRASAD

The reason why Node is so fast and so good is that it's light and thin. It barely has anything in it at all, so every little thing you add to it, each additional Node module you want to use with it, comes at a cost.



times faster than what we had been using and its memory footprint was smaller as well.

Obviously, Node.js also offers other benefits beyond the technical aspects. JavaScript is a language lots of people understand and are comfortable coding in. Besides, it didn't hurt that Node was getting a lot of hype at the time—and still is. In some ways, that makes it easier for me to recruit.

TERRY COATTA: You mentioned that you had moved away from using Active Record in Ruby as your model representation, but obviously that model had to end up going somewhere. Why didn't you choose to use the infrastructure you already had for the model to handle the middleware as well?

PRASAD: The thing about models is that they are really designed around objects. They have properties; they have methods; they are very structured and statically typed; and you're trying to create an environment where they're very solid so you know exactly what each object is. This is just the sort of approach we were all taught in our first exposure to object-oriented design.

So you would think a more object-oriented language would be ideal for building systems like that. But then that moves out to the view controllers, and in mobile systems, the view con-

trollers get pushed out all the way to the client. Then the question becomes: What's left in the middle? Basically what you find there is just a bunch of functions that effectively manipulate hashes of data in order to format them. So now you're just down to formatting and a little bit of aggregation.

I'm not sure I really had the clarity at the time to articulate why we chose Node.js. Now after using it for a few years, it has become pretty clear that in this layer that's essentially the glue between your front end (which is literally in the client now) and your back end (which happens to be your data model), a functional sort of language is actually the best fit. At this point, we write all our back-end stuff in Java, and all the Java stacks we're using are more process based.

COATTA: Having had this experience with Node.js in the middle layer, then, would you contemplate using it for the back end?

PRASAD: Right now, we're working on creating a data store that's mobile-specific, and as part of that we did an analysis of whether we should build it in Node.js and JavaScript. It turned out the team wanted something that was a little more precise—or typed, I suppose. I'm actually trying to steer clear of classic programming stuff such

as taking a more statically typed approach, but the team definitely wanted the ability to define an object with certain methods and properties, and for that to be guaranteed so nobody could mess it up. They don't want just anyone out there taking over that prototype or doing anything to it. Still, from a performance standpoint, I would love to try a more event-driven framework. Currently, we're using Rest.li internally to do some of this back-end, data-store stuff because we really do believe the event-driven stuff has transformed our architecture.

COATTA: In terms of the performance speedup you observed with Node.js, did you also construct lightweight prototypes for the other languages you were considering for your middleware layer?

PRASAD: We did some prototyping with Ruby and Python using the evented frameworks EventMachine and Twisted. The bottom line was that Node proved to be 2–5 times faster than both of those in terms of raw throughput. What was even more exciting and really sold us on Node was that it took only two or three hours to write the Node prototype while it took us more on the order of a day or two to write the EventMachine and Twisted ones, just because we had to download so much more stuff.

For example, you need to make sure you're not using the standard HTTP library in Python, but instead the Async HTTP library. That's the sort of thing that pretty much applied across the board. No matter what we wanted to do, we couldn't use the standard Python library. Instead, we had to use the special Twisted version. The same held true with Ruby. Like a lot of others in the community, we found out just how much easier it is to get started with Node, where everything you need is essentially there by default. Further, the fact that we could get stuff up so much faster with Node was really important. That's just another form of performance, right? Developer productivity definitely counts for something.

Memory footprint was also a factor. We looked at how well VMs (virtual machines) worked in each of these languages, and the V8 JavaScript Engine just blew everything else away. We were doing 50,000 QPS with all

this manipulation, and we were running that in about 20MB to 25MB of memory. In EventMachine and Twisted, just loading all of the classes necessary to do the async stuff rather than the standard stuff was more like 60MB to 80MB. Just at a raw level, Node was going to run in only about half the memory footprint.

NORTON: What sorts of things did you end up missing that you would have had in a more “modeled” environment?

PRASAD: Because we embraced the functional nature of JavaScript, we didn't think we would first have to translate everything coming in from the back end into a set of objects and a set of methods on those objects. This also meant we didn't have to sort out the hierarchy of those objects, what the subclasses were, what the base class was, how all those things were structured, and what the relationship was among all those different objects. But that's what you would need to do in a model- and object-based system. So for us it was more like saying, “OK, you're going to hit these three endpoints, and then you're going to merge the data and pop out this other object.”

In reality, though, it's not an object but a hash, right? You're consuming adjacent things and then popping out some other adjacent things. It's almost like you've got a filter that you run a stream through. Data comes through and then pops out the other side—which gets you completely around all that thinking about what the objects are, how they work, and how they interact. So that let us get where we wanted to go a lot faster.

Now that we're through that, we're going back and working on some functions. Right now, they all talk to whatever they want. If Function A wanted to talk to Profile and Companies and Jobs and then merge them, then it would just go ahead and talk to Profiles and Companies and Jobs. And then if Function B wanted to talk to Profiles, Companies and something else, it would just go ahead and do that. But the two functions—A and B—wouldn't be using the same functional interface to talk to Profiles or Companies. The problem is that if there is a bug in the way we're talking to Companies, I would have to go in and fix that in

two places. If I wanted to add logging so I could see all the instances where somebody was talking to Companies, things are not centralized such that everything would funnel through nicely. While we're not creating an object layer, we are starting to recognize that, at least for RESTful APIs, we need to create a set of functions that sit in front of each resource type we're looking to communicate with. It's kind of like proxying that interface. That's one of the things we've learned along the way and are now starting to fix by creating this new layer of abstraction.

COATTA: You mentioned refactoring the code to introduce additional layers. If you consider the amount of time you're putting in now on the refactoring side, do you think you might still end up taking roughly as much time to create your code base as you would have had you taken some other approach?

PRASAD: I would guess not because it isn't just about how long it takes to do the coding itself. It's also about every other aspect of the process. For example, when you write your app and then type “Node” to run the app, it literally takes only about 20–100 milliseconds for it to come up. With Ruby, just getting the Rails console to come up sometimes takes more like 15–30 seconds. I'm also not ready to say Node comes up short strictly from a coding standpoint in any event. All the way around, Node is just built thinner, lighter, faster. So every last little step, every single nuance of everything I do every day ends up being faster.

With the more structural languages, you've got to account for things like compile times and build times. Then you end up building essentially hot-swap environments where the environment is running but the IDE is also able to connect with and manipulate the runtime. You pretty much have to do it that way, because it takes so long for the thing to boot up, make your changes, and then boot up again. Node eliminates a whole range of these problems just because it's so fast.

Yes, the refactoring has added to the coding time while it has also taken away from the simplicity we enjoyed initially when we were just slashing and burning through things. But I think the overall efficiency we've been able to achieve because Node is so

thin and light more than makes up for any minor amount of refactoring we've had to do.

COATTA: If you were about to dive into another project based on Node, would you try to build a little more of your structure initially?

PRASAD: I don't think that's a Node-specific question. It's more a matter of personal preferences. My own view is that front-end UI code tends to last only about one-and-a-half to two years anyway. Very little of that code lasts as long as five to 10 years. The reason for this, I believe, doesn't even have all that much to do with the quality of the code, but instead is driven more by the evolution of technology and the fact that software developers are encouraged to work in four-year increments. You constantly have new people looking over your code, and I know that whenever I look at anything, even if it's just a table, I'm sure I could build a better one. So I believe the natural tendency is that, every year-and-a-half to two years, a new set of people is going to look over some particular code base and decide they can do a better job.

Given that, it's probably faster just to rewrite a whole chunk of that code base so long as it's modularized—or maybe even rewrite the whole thing—and take your one- or two-month hit for that, than it would be to slowly evolve the code base. So, if I had this project to do over again, I'd probably do it the same way. In any event, I'm more inclined to build a project first, get it out, and then extract a platform later if I can—as opposed to building a perfect platform and all the components upfront and then trying to hook them all up in the right order. I don't think you can possibly know what the right order is until you've actually got something running in production and can see where you're hitting pain points. Then you can tell where you actually need to extract libraries and start doing something about it.

Once the decision had been made to convert to Node, Prasad's team had to figure out the best approach to implementation and maintenance. Because LinkedIn's mobile services team was largely accustomed to Ruby on Rails, they mimicked parts of their previ-

ous Rails structure in setting up the Node.js structure, allowing them to jumpstart the project. The team was small, so Prasad was able to monitor the transition to JavaScript and detect problems quickly.

Programming in Node.js is event-driven and so required different approaches. Although a minor amount of refactoring was involved, the team didn't have to create new abstractions or additional functions. Most of what they did was syntactical in nature. They also shed layers of abstraction, which significantly reduced the size of the code base.

NORTON: You say your preference is to jump right into a project and do things the fast way. I wholeheartedly agree. In fact, I'd say my own philosophy is that since you don't really know going into a project what it is you're going to need to optimize for, then what you really ought to be setting out to optimize is your ability to change things.

So I'm really curious as to how you structured your code and what practices you used to let you move fast initially and then continue moving fast as the constraints for your project became clearer.

PRASAD: A lot of our people at the time were Ruby on Rails developers, so they were familiar with that directory-tree structure and the terminology used in the Rails world. We mimicked that terminology, and that gave us a huge jumpstart. It also helped that Node is pretty barebones, much like Rails, but that also was a little scary because you're not sure how to structure stuff, and there were no guides to tell you how to do it.

A second helpful decision we made was to put all the controllers and views on the client side, while the models were placed on the back end. From a directory-structure standpoint, that meant we wouldn't have any files showing up in our model structure or in our views directory. Although we used the term *controllers*, we were really working more with formatters in the sense that you would get something in, make a couple of requests, format the thing, and then pop it out.

This meant we still had this old Rails structure, only most of the di-

rectories were actually empty—and, by the way, an empty directory was actually a good thing. We would leave the directory there just so people wouldn't try to create things in that directory. It was like a code-review tool that said, "We must be really doing something wrong if we start creating things in that directory." And that's just from our server design pattern of how we wanted to use Node.

Another helpful thing we did initially was to set up a basic testing environment and framework. Rails, as well as Python's Django framework, were very big on TDD (test-driven development) and even BDD (behavior-driven development), where you could essentially write your tests and sequences before filling in all the code. It's a model that works really well when there's a testing framework. We essentially took the testing framework that was already there and slapped it on top of our directory structure using some scripts we'd written to get the two to interact. We started off using Vows, but after three months ended up writing all our own tests in Mocha.

You could say that setting up a testing environment and framework was sort of another code pattern, but it was probably more difficult than it would have been with any other language—especially in terms of dealing with the evented aspect of Node. Even though we ended up refactoring the code, we didn't have to do anything major such as coming up with new abstractions or any additional functions. Most of what we did was syntactical in nature. And then there were a bunch of little functional things—for example, Node's concept that in every callback the first argument should be Error, which makes total sense.

But then you couldn't figure out what the second and third and fourth arguments were. Or do you have only one argument? Or is it five arguments? How do you deal with that, and what is it going to be like after you do the callback? Let's say you now want to change the callback signature. How do you tell everyone who was calling the function and expecting a callback that the signatures have changed? I don't know if that's Node-specific or is common to all JavaScript, but it did take us awhile to figure it out.



KELLY NORTON

My own philosophy is that since you don't really know going into a project what it is you're going to need to optimize for, then what you really ought to be setting out to optimize is your ability to change things.



NORTON: How did your team communicate interface boundaries? People who are accustomed to working with types wouldn't want to give away interfaces, since they provide a concrete form of documentation that basically allows one team member to say to another, "Here is my intent. I expect you'll be calling me in this particular way." Even more importantly, if that way of calling turns out not to exist, that probably means it's a use case you haven't considered and so is one that probably won't work.

PRASAD: I think there are interfaces between libraries inside the code and between the client and the server. For the interfaces between the client and server, we used REST (representational state transfer), and we had a very defined model where we had what we called "view-based models" that were returned by the Node server. We would just document those and say, "Hey, here are the REST interfaces, and here's what we support." That's essentially along the lines of a versioned interface structure. It's classic REST, actually.

Within the code base, we heavily used the module systems. Each REST endpoint has a file that represents all the responses for that endpoint, as well as the public interface for the

module mapped to the routes. You can have as many functions as you want, but whatever you have will export out of that module. In that way you actually end up specifying the set of functions you're exposing. That's essentially what we use as our interface.

Then, structurally, we did something really simple: Every function inside the module, whether public or private, was defined in an old-school C style. You might put in a comment that said "private" and then list some of your private functions. Or you could put in a comment that said "public" and then list all your public functions. This is just like what you used to do in C, right? You had a bunch of functions and you'd put the private ones in one group and the public ones in another group. Then your header file would essentially expose your public interface. All of that happened in a single file. We don't have a header file, but `module.exports` effectively serves as our header.

NORTON: Another important communications consideration has to do with helping a team that's not accustomed to writing JavaScript navigate around some of the minefields they're likely to encounter.

PRASAD: When we started on this, we were a group of only four people,



TERRY COATTA

If you consider the amount of time you're putting in now on the refactoring side, do you think you might still end up taking roughly as much time to create your code base as you would have had you taken some other approach?



so I was able to watch every check-in. Whenever I saw anything odd, I would ask about it—not necessarily because I thought it was wrong but because I wanted to understand why we had used that particular pattern. It was easy to get to the bottom of why things had been done in a certain way and figure out what we were going to do next. Now the group is much bigger and we're percolating the nuances of the reasoning behind the choices we made, and that's definitely a lot harder. Now we hold a three- to five-day boot camp whenever a new person starts with the group, which gives us a chance to explain, "Here's how we do this. We know that might seem a little strange, but here's why we do it that way." I think that's probably the best way of exposing those code patterns.

COATTA: Which of those code patterns do you think are the most important?

PRASAD: We ended up using Step as our flow-control library. It's like a super-simple Stem, with two main constructs. We augmented those a little and ended up adding a third. Basically, Step has this concept of a waterfall callback, meaning that you pass into Step an array of functions and then it will call each function in order such that as the first function returns, the second one will be called, and so forth.

Step guarantees the order of this sequence, so even if the function is asynchronous—meaning it's going to do something evented—that function will be passed a callback, and it has to call once it's finished doing its thing. Independent of whether each function is synchronous or asynchronous, the waterfall will execute in sequence.

Step also includes a group method and a parallel method. We use the group method very heavily. That means you can give it a group of functions and it will execute all of them in parallel, and then return when all of them are done executing.

One nuance has turned out to be a pretty big deal for us. If we have a group that contains three functions and one of them is broken, Step won't capture the responses for the other two. Instead, it will just call the callback and say, "Sorry, I'm errored."

The negative thing about this environment is that if I call six things and two of them are required while four are optional, I'm not going to mind waiting for all of them with certain timeouts. But if one of those optional things ends up erroring out, then it's going to seem like the whole block errored. That's just not ideal for us. We therefore created a function called GroupKeep that runs through

and executes everything, and then if there should be an error, it will hold the error in an array. That way, when the calls go out for callbacks, there will be this array of errors. Based on the position of an error, you can get a pretty good idea of whether it relates to something that's required or something that's optional. That makes it possible to write code that can continue a process wherever necessary.

The light and thin nature of Node.js appealed to Prasad and his team more than anything else. The extent of code reduction this allowed proved to be huge—from 60,000 lines down to just a couple thousand. They also now have essentially no frameworks, thus eliminating a lot of extraneous code. Moreover, Node's event-driven approach requires fewer resources and moves more functions to the client side. Finally, it takes a functional approach that sheds layers of abstraction. In sum, this all serves to enable support for huge numbers of users on a wide array of devices in real time.

COATTA: When you were talking about how quickly you managed to get your initial Node prototypes up and running, it made me wonder whether you had also achieved some code reduction.

PRASAD: Absolutely. Our Node code base has grown a little from the original version, but it's still on the order of 1,000 to 2,000 lines of code. The Ruby code base we were using previously, in contrast, was in the neighborhood of 60,000 lines of code. The biggest reason for that reduction is that our current code base is essentially framework free, which means there's just not a whole lot of cruft in there.

The second big reason has to do with the more functional approach we're taking now, as opposed to an object-oriented one, which proved to be an important shift for us. In Ruby, the natural tendency is to create an object that essentially wraps every communication and type. Even though Ruby is actually a functional language, it has a much stronger notion of class and object than does JavaScript. So in our earlier code base we had lots of layers of abstraction and objects that had

been created under the guise of greater componentization, refactorability, and reusability. In retrospect, however, we really didn't need most of that.

Another significant reason for the code reduction is the momentum behind the MVC (model-view-controller) model, at least for mobile vs. Web-based systems. Before, we had mostly server-side rendering. Now with the move of templates and views over to the client side—along with rendering, of course—a lot of that code has just gone away. Along with that has come a new trust and belief that the back ends, where the models live, are where the validation and all the other more advanced things are going to happen. That means not having to double-check things, which eliminates another huge chunk of code.

NORTON: You indicated earlier that one of the insights that led to your rewrite in Node.js was that you realized you didn't really need a deep understanding of the objects you were manipulating, meaning you didn't need to mutate those objects a lot. Basically, you could just do a lot of merging of hashes. Do you think you could have gotten to that same end with some other language, even Ruby, just by working with the hash-map primitive?

PRASAD: Probably so, but if you look at Ruby, you see that Rails just has so much extra stuff in it, whereas Node at its base has an HTTP server aspect and a client aspect built into the binary. This means you don't need an HTTP Node module and an HTTP listening module.

So, yes, I suppose if we had eliminated all the object hierarchy and just used the hash structures, we might have been able to use Ruby. But then you would still have to listen to HTTP and turn that into a controller, which just gets you back into adding all these little microlayers. While each microlayer gives you a bunch of code you don't have to write, it also adds a bunch of requirements for stuff you do then have to write so everything will work nicely with your framework.

COATTA: If you were to talk to someone else who was about to undertake a similar project, what would you point to and say, "Hey, pay attention to this or you're going to be in trouble"?

PRASAD: Flow control. Exception

handling. And while this isn't really specific to Node, I'd say, "Keep it light. Keep it thin." I think there's a natural tendency for people to say, "Well, I need something that does HTTP, so I'll just find a module that does that," and then another 4,000 lines of code drops into their environment when all they really need is an HTTP request. Instead, they end up with this super-duper thing that gives them that and a whole bunch of other stuff besides.

Basically, the reason why Node is so fast and so good is that it's light and thin. It barely has anything in it at all, so every little thing you add to it, each additional Node module you want to use with it, comes at a cost.

NORTON: For those companies that have already launched projects in Node, what would you say are the three things they might want to add to their ecosystems to make them even stronger?

PRASAD: First would be a good IDE. IntelliJ IDEA is pretty good, but outside of that, I haven't really seen a great IDE and toolset for Node.

Second would be to allow for evolving performance analysis and monitoring. Better operational monitoring for Node would be great, but for now it's essentially a black box unless you put your own monitoring hooks into your code. I'd love to see a lot of the stuff a JMX layer in the Java VM provides. You can get out some really useful information that way.

And the third thing would be something like New Relic for Node—something that can inspect everything your Node system is doing and actually understand your application so it can provide you with detailed breakdowns of where your bottlenecks and slowdowns are. That would be awesome, actually. C

Related articles on queue.acm.org

Reveling in Constraints

Bruce Johnson

<http://queue.acm.org/detail.cfm?id=1572457>

Multitier Programming in Hop

Manuel Serrano and Gérard Berry

<http://queue.acm.org/detail.cfm?id=2330089>

High Performance Web Sites

Steve Souders

<http://queue.acm.org/detail.cfm?id=1466450>

© 2014 ACM 0001-0782/14/02 \$15.00

“Not invented here” syndrome is not unique to the IT world.

BY POUL-HENNING KAMP

Center Wheel for Success

WHEN I FIRST read the claim that HealthCare.gov, the website initiated by the Affordable Care Act, had cost \$500 million to create,⁴ I did not believe the number. There is no way to make a website cost that much. But the actual number seems not to be an order-of-magnitude lower, and as I understand the reports, the website does not have much to show for the high cost in term of performance, features, or quality in general.

This is hardly a unique experience in the IT world. In fact, it seems more the rule than the exception.

Here in Denmark we are in no way immune: POLSAG, a new case-management system for the Danish police force, was scrapped after running up a tab of \$100 million and having nothing usable to show for it. We are quick to dismiss these types of failures as politicians asking for the wrong systems and incompetent and/or greedy companies being happy to oblige. While that may be part of the explanation, it is hardly sufficient.

The traditional response from the IT world is that the Next Big Thing will fix this, where the Next Big Thing has been a seemingly infinite sequence of concepts such as high-level languages, structured programming, relational databases, SQL, fourth-generation languages, object-oriented programming, agile methodologies, and so on ad nauseam. I think it is fair to say none of these technologies has made any significant difference in the success/failure ratio of IT projects. Clearly they allow us to make much bigger projects, but the actual success/failure rate seems to be pretty much the same.

At the same time, there are all these amazing success stories, where a couple of college kids change the way we think about information retrieval with their Google information-scoring algorithm, or a bunch of friends change the way we communicate with their Twitter information-distribution system.

Why, despite politicians' lofty speeches, does that never happen in government IT applications? There is clearly something we are missing here, something we are doing wrong, without even thinking about it. That particular mistake is far more common than it should be in a (so-called) “knowledge economy.”

Lessons from Wheelbarrows

Growing up in the countryside, I spent a good portion of my youth operating a wheelbarrow. The European wheelbarrow is a rationalization of the handbarrow, which was basically two planks, two feet apart, with boards nailed or tied between them. One person grabs the two planks at the front, one in each hand, another grabs them at the back, and then they trudge away with their load.

Sometime back in the low thousands, a productivity consultant must have pointed out that if you replaced the person in front with a wheel, then you could get twice as many wheelbarrows moving with the same number of workers. (This industrial application of technology undoubtedly earned the consultant a hefty fee.)

And that is it! That is the very same contraption I lugged around as a kid and the same one I used just a few hours ago for gardening. As anybody knows, using a wheelbarrow is easier than carrying things, but it is still quite heavy work. You lift roughly half the load yourself, you provide the energy for motion, and you must steer it in the right direction, which is difficult on account of the first two expenditures of energy.

While a vast improvement over the handbarrow, the wheelbarrow is stupidly inefficient, at least compared with the Chinese version.² Somebody in China was smarter than the Medieval European downsizer and moved the wheel to the middle of the wheelbarrow, so that the entire weight of the load is carried by the wheel. The Chinese wheelbarrow will readily transport two or three times the load of a European wheelbarrow, with the operator hardly breaking a sweat, just pushing and steering, with barely any lifting.

From a management perspective, the Chinese wheelbarrow is identical to the European one: one wheel, two handles, one operator. Looking at it that way, however, we blind ourselves to how differently they work, and we miss the full productivity improvement of the wheel.

In Europe we have known about the Chinese wheelbarrow since at least 1797,² yet, to this day, we still sweat while lifting half the load carried on our nonoptimized wheelbarrows.

The “not invented here” syndrome is not unique to the IT world.

I am beginning to think the reason our big IT projects sink is that we make the same kind of mistake: mindlessly replacing human labor with technology instead of solving the actual problem.

Many human jobs can be replaced directly with computers. Email replaced the old telegraph system, delivering the exact same conceptual service: delivering a text message quickly while using hardly any manpower. But delivering text messages was the least email could do—once we got to know

it better.

First there were programs answering email messages, sending source code, or looking up things in databases. Next came programs sending email to other programs, to keep databases synchronized, and then email containing pictures, sound, and vice presidents.¹

However, the email system we know today, as envisioned by Ray Tomlinson, was not the only such system somebody created. The state-sanctioned post and telegraph monopolies attempted to standardize email—or “telematic services” as they called it—in CCITT (International Telegraph and Telephone Consultative Committee) recommendations X.400-X.599,³ as part of the grand vision of “The Intelligent Network.”

They started approximately 15 years before Tomlinson. They spent uncountable millions of all sorts of currencies. They had legislators mandating their way be the one and only legal way forward. And they failed utterly, miserably, and definitively.

Why is it that in IT one person can often do what thousands cannot?

It is tempting to speculate that HealthCare.gov would have worked much better had they given the task to a 10-person company rather than a conglomerate with 69,000 employees all over the globe. I am sure that is a necessary part of the solution, but again, it is hardly a sufficient condition for success.

For one thing, while there are “only” 380,000 words in the Affordable Care Act (also known as Obamacare), the regulations floating from the law amount to 12 million words (and counting). No 10-person company would even be able to read all that verbiage before the delivery deadline had whooshed past.

Interestingly, *The New York Times* reports that HealthCare.gov contains an estimated 500 million lines of code.⁴

That is no more likely to be true than the \$500 million price tag.

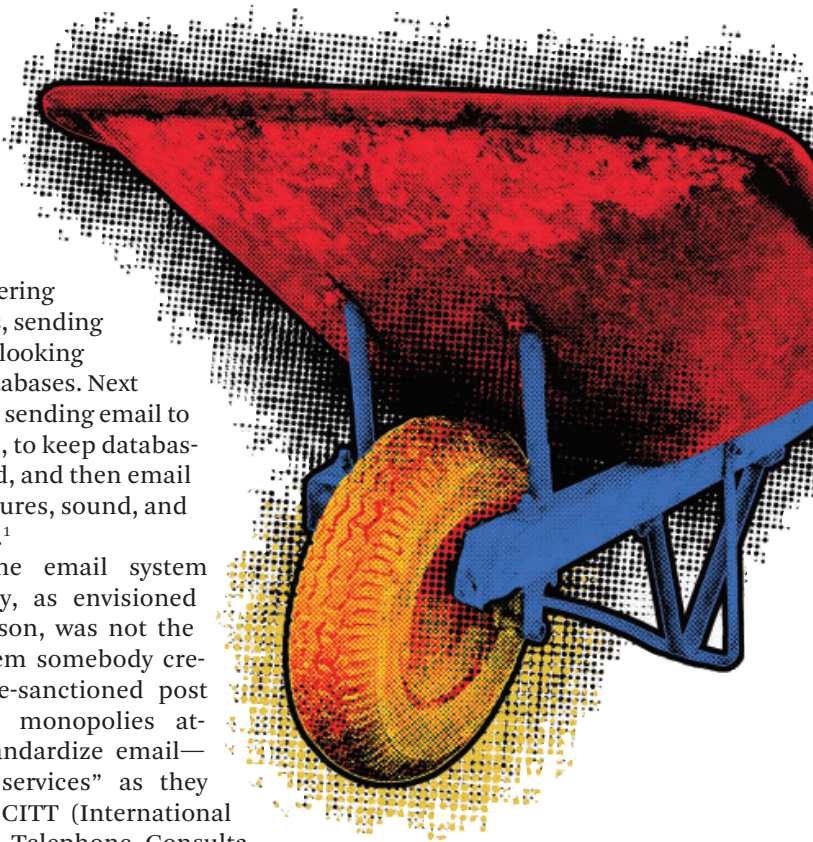
I looked at one of the actual laws that make up Obamacare, the Patient Protection and Affordable Care Act (PPACA),⁵ and since I was not going to read all 906 pages, I started in the middle, on page 403. After a few pages I ran into this definition of *patient decision aid*:

“(1) PATIENT DECISION AID—The term ‘patient decision aid’ means an educational tool that helps patients, caregivers, or authorized representatives understand and communicate their beliefs and preferences related to their treatment options, and to decide with their healthcare provider what treatments are best for them based on their treatment options, scientific evidence, circumstances, beliefs, and preferences.”

Reading on, I found the requirements:

“(2) REQUIREMENTS FOR PATIENT DECISION AIDS—Patient decision aids developed and produced pursuant to a grant or contract under paragraph (1):

“(A) shall be designed to engage patients, caregivers, and authorized representatives in informed decision making with healthcare providers;



“(B) shall present up-to-date clinical evidence about the risks and benefits of treatment options in a form and manner that is age-appropriate and can be adapted for patients, caregivers, and authorized representatives from a variety of cultural and educational backgrounds to reflect the varying needs of consumers and diverse levels of health literacy;

“(C) shall, where appropriate, explain why there is a lack of evidence to support one treatment option over another; and

“(D) shall address healthcare decisions across the age span, including those affecting vulnerable populations including children.”

Unless Congress thinks of teachers as “educational tools,” I think we can take it as written here that they expect this to be some kind of computer program. But read it again and pay attention to the language. When was the last time you saw a computer program that “engaged,” “explained,” or “addressed decisions?” Or, for that matter, when have you seen a program that “adapted for [...] a variety of cultural and educational backgrounds to reflect the varying needs of consumers and diverse levels of health literacy”?

These paragraphs legislate that Obamacare will fund research in heavy-duty state-of-the-art artificial intelligence—I somehow doubt that is what Congress intended it to say. I posit that Congress worried about having enough doctors and nurses for this new healthcare, so they wanted to use computers to cut down the talking and explaining. In other words, they want to save manpower—by replacing the front man on the handbarrow with a wheel.

I have used a handbarrow once, in an emergency. My fellow campers and I constructed it from two young pine trees, wrapping the sail from our tent around them. Compared to a wheelbarrow, it was both easier and faster, because the front man did not get stuck in any holes or hit any rocks, and he helped with all of navigation, lifting, locomotion, and steering. When we met the first responders, they gently lifted our friend with his injured leg from our makeshift version to their professional handbarrow

Blindly deciding that IT be substituted for humans is unenlightened. IT is not a magic potion that makes unpleasant or inconvenient things disappear.

and carried him the rest of the way to their ambulance on a high-tech aluminum stretcher.

I am absolutely sure that Congress would never replace the front man on an ambulance stretcher with a wheel to save manpower—yet, in a way, they did just that. I do not claim to know the correct way to optimize a healthcare consultation with computers—there may be one, but more importantly, there may not.

Blindly deciding that IT be substituted for humans is unenlightened. IT is not a magic potion that makes unpleasant or inconvenient things disappear. The right thing to do is to ask, as a Chinese engineer did 2,000 years ago, “If we’re going to put a wheel on this thing, where is the best place to put it?”

And to realize that two questions were asked. C

Related articles on queue.acm.org

More Encryption Is Not the Solution

Poul-Henning Kamp

<http://queue.acm.org/detail.cfm?id=2508864>

Better Health Care Through Technology

Mache Creeger

<http://queue.acm.org/detail.cfm?id=1180186>

A Requirements Primer

George W. Beeler and Dana Gardner

<http://queue.acm.org/detail.cfm?id=1160447>

References

1. Borenstein, M. and Linimon, M. The extension of MIME content-types to a new medium. RFC 1437, 1993; <http://www.rfc-editor.org/rfc/rfc1437.txt>.
2. De Decker, K. How to downsize a transport network: The Chinese wheelbarrow. *Low-tech Magazine*; <http://www.lowtechmagazine.com/2011/12/the-chinese-wheelbarrow.html>.
3. International Telecommunication Union. ITU-T recommendations; <http://www.itu.int/ITU-T/recommendations/index.aspx?ser=X>.
4. LaFraniere, S., Austen, I. and Pear, R. Contractors see weeks of work on health site. *The New York Times* (Oct. 20, 2013); <http://www.nytimes.com/2013/10/21/us/insurance-site-seen-needing-weeks-to-fix.html>.
5. Patient Protection and Affordable Care Act. 2010; <http://www.gpo.gov/fdsys/pkg/PLAW-111publ148/content-detail.html>.

Poul-Henning Kamp (phk@FreeBSD.org) is one of the primary developers of the FreeBSD operating system. He is widely unknown for his MD5-based password scrambler, which protects the passwords on Cisco routers, Juniper routers, and Linux and BSD systems. Today, he is an independent contractor.

A cohesive, independent solution for bringing provenance to scientific research.

BY ZACHARY HENSLEY, JIBONANANDA SANYAL, AND JOSHUA NEW

Provenance in Sensor Data Management

IN TODAY'S INFORMATION-DRIVEN workplaces, data is constantly being moved around and undergoing transformation. The typical business-as-usual approach is to use email attachments, shared network locations, databases, and more recently, the cloud. More often than not, there are multiple versions of the data sitting

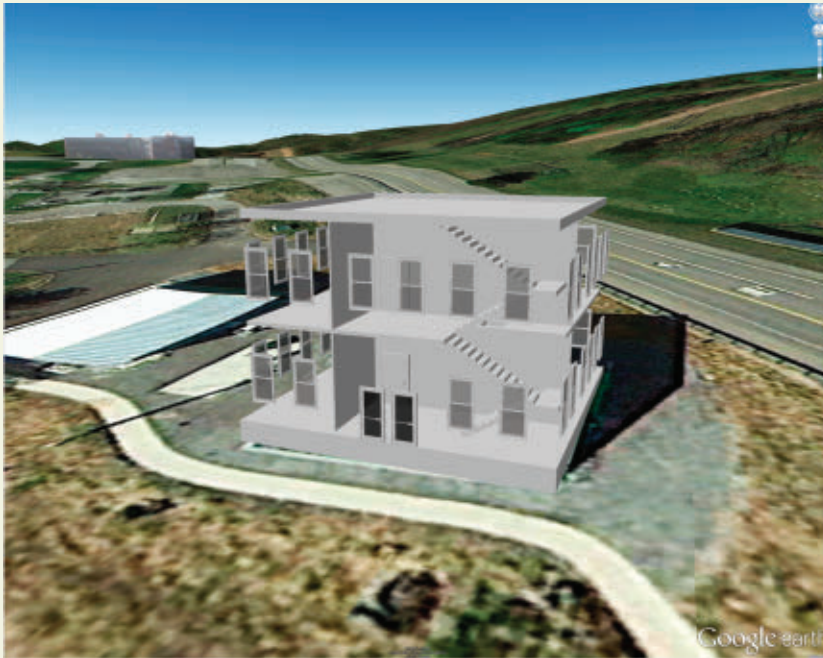
in different locations, and users of this data are confounded by the lack of metadata describing its provenance—or in other words, its lineage. The ProvDMS project at the Oak Ridge National Laboratory (ORNL) described in this article aims to solve this issue in the context of sensor data.

ORNL's Building Technologies Research and Integration Center has reconfigurable commercial buildings deployed on flexible research platforms (FRPs). Figure 1 is a Google Earth model of a medium-size commercial office building that is part of the ORNL's FRP apparatus. These buildings (metal warehouse and office) are instrumented with a large number of sensors that measure variables such as HVAC efficiency, relative humidity, and temperature gradients across doors, windows, and walls. The sensors acquire sub-minute resolution data from hundreds

of channels. Scientists conduct experiments, run simulations, and analyze the data. The sensor data is also used in elaborate quality assurance exercises to study the effect of systemic faults. The two types of commercial buildings comprising the FRPs stream data at a 30-second resolution for a total of 1,071 channels for both buildings.

The sensor data collected from the FRPs is saved to a shared network location accessible by researchers. It became apparent that proper scientific controls required not just managing the data acquisition and delivery, but also managing the metadata associated with temporal subsets of the sensor data. The ProvDMS, or Provenance Data Management System, for the FRPs allows researchers to retrieve data of interest, as well as trace its lineage. The life cycle of most objects consists of creation, curation, transfor-

Figure 1. Model of a building that is part of the ORNL's FRP apparatus.



mation, archival, and potentially deletion. *Provenance* is the tracking of such information.⁸

ProvDMS provides researchers with a one-stop shop for all data transformations, allowing them to effectively trace their data to its source so that experiments and derivations of experiments can be reused and reproduced without the overhead of repeating every experiment.

There are a number of existing software systems for provenance data collection with strong workflow integration. Chimera⁶ is a process-oriented provenance system that manages derivation and analysis of data objects in collaboratory environments. It stores provenance information that can be used to regenerate, compare, and audit data derivations within the system. The Karma provenance system⁷ allows users to collect and query provenance of scientific data processes with the ability either to run stand-alone or as part of a greater cyber-infrastructure setup. The Karma system is intimately connected with its data as a result of its close workflow integration. Vis-Trails^{4,5} provides support for scientific data exploration and visualization with a strong focus on work flow as provenance objects to represent complex computations. Workflow in Vis-

Trails can be visualized as pipelines of procedure sequences that lead to a computational output. The EU Provenance Project¹ uses an open provenance architecture for grid systems with a service-oriented approach, namely for aerospace engineering and organ-transplant management. In the EU Provenance Project, the provenance system was used to track medical information in units of patient/doctor interactions. The project attempted to find equilibrium between the amount of data collected and minimizing the intrusiveness of the collection effort in order to preserve the quality of medical care.

While many of these systems are complete software solutions, Core Provenance Library (CPL)³ was designed to be application independent and easy to integrate into new or existing systems. Because of its independent nature, CPL was used in ProvDMS to serve as the provenance backend. This allowed the user interface of ProvDMS to be separate from CPL's object constraints, thereby producing a positive user experience.

Effective Provenance Design

Particular implementations of provenance can vary greatly depending upon a few important attributes. The focus

of ProvDMS is on researcher requirements, granularity of the provenance, workflow requirements, and object design. Its design principles emphasize the importance of user needs, taking a cohesive but independent stance on the integration of provenance with user tools.

Granularity. Most systems incorporate both fine and coarse granularity to avoid restricting the type and amount of data available to users.² ProvDMS implements a fine-granularity system but provides a mixed-granularity interface for users so that tracing lineage using visualization is contextual. Users are shown generalizations as coarse provenance objects that can be contextually expanded to provide finer granularity information. This allows users still to see finer, exact provenance objects that specifically map to logical objects in the system but are not overburdened with unnecessary information when viewing the provenance.

Tool integration with workflow. ProvDMS's design was largely determined by the "when" and "how" of integrating with existing tools. Most tools have limited to no provenance-tracking abilities. ORNL researchers routinely use a wide array of specialized tools from various vendors that do not have provenance support. As a result, ProvDMS could not have many restrictions. While challenging, this steered the focus toward data infrastructure requirements to enable tracing the provenance while facilitating development of software interfaces to support future system integration. To enable a sense of workflow, ProvDMS uses the notion of a user *experiment*, where the sensor data resides once exported from the system. Users may choose any tool and have the option of importing different states of their experiments back into ProvDMS.

Provenance of provenance. The ability of a provenance system to track how it creates and tracks provenance objects was not an initial design requirement for ProvDMS but emerged from the abilities of CPL. By tracking provenance of provenance (PoP), ProvDMS provides specific information about when the provenance system created new objects or versions of objects, which user was responsible for the creation of objects, the process ID that performed tracking functions, and sys-

tem information such as the executing environment. Administrators of this system can now track system usage over time and may detect patterns in how the system and provenance data storage is being used.

Uniqueness. Provenance systems inherently involve hierarchical connectivity among objects. The use of CPL as the provenance back end allows users to access provenance object ancestry easily. Additionally, CPL's versioning system ensures each object is uniquely identifiable, which solves the design issue of the user's ability to define an experiment multiple times.

Object design. Object design shaped the entirety of ProvDMS and arguably comprised the most difficult set of decisions to make. The first challenge was to determine how users are expected to interact with the data that would determine the required provenance objects. This was difficult to gauge for a system that was still on paper. We were unsure of the level of granularity of provenance to store and expose since there was the possibility that much of the provenance data could go unused. We leaned on the side of finer granularity while providing support across a spectrum of granularity to account for the yet-to-discover unknowns in ProvDMS.

Provenance objects in CPL are uniquely defined using three main attributes: *Name*, *Type*, and *Originator*. In ProvDMS's use of CPL, *Name* describes the object, and *Type* determines its granularity. The *Originator* is to be used in a similar vein to Java's package-naming convention, via hierarchical domain namespaces. ProvDMS uses the name of the system as the top-level domain, user as the next level, and interface as the final level. This ensures the existence of understandable and unique originators differentiating the *experiments* (and corresponding provenance objects) based on authenticated users.

System Architecture, Design, Cohesion, and Independence

The FRPs use Campbell Scientific's data loggers for collecting data from 1,071 channels in the facility. Campbell Scientific's Loggernet Database (LNDB) runs on a dedicated server and populates a MySQL database with the raw sensor output. ProvDMS runs on

another dedicated server and retrieves the data from the MySQL database to fulfill user needs, thereby providing complete separation of the raw data store from the provenance trace. LNDB creates the required schema on the data server, and ProvDMS is architected to sense the schema and its logical relationship to the FRP in order to present a cogent, simplified interface to the users. Checks are in place to ensure data backup, security, and isolation since much of the data is proprietary.

Figure 2 shows the logical representation of the physical layout of FRP data. This influences the provenance object design of ProvDMS. As illustrated in the figure, the sensor data is separated into *stations*, each containing a set of *data loggers*. These data loggers consist of a set of data *channels*. Physically, these channels relate to *sensors* placed in different locations throughout the test facility.

The ultimate goal of the provenance system is to trace the participation of temporal subsets of sensor data in user experiments. ProvDMS treats these as objects. Researchers export a temporal subset of the chosen sensor chan-

nels as an experiment, which can then go through various transformations in the user's workspace. Once researchers feel ready, they may submit the "state" of their experiment to the system, along with any additional derived data, supporting files, results, or other metadata. ProvDMS allows users to map the uploaded files as a derivative of the original experiment.

The logical representation of FRP data was designed to correspond to the provenance objects. Each *type* of provenance object relates to its logical representation. These objects are similar in their representation, with a few differences. Importantly, there are additional links from data loggers to their associated files. In the case of user-defined experiments, these are the files holding channels of sensor data. For derived experiments, these are any associated files used in the derivation. Figure 3 illustrates the differences in their representation. Two types of links are used: version dependencies and data-flow dependencies. Differences between these links are important for the cycle-avoidance algorithm in the CPL.

In addition, there are two types of links between objects: version depen-

Figure 2. Logical representation of the physical layout of FRP data.

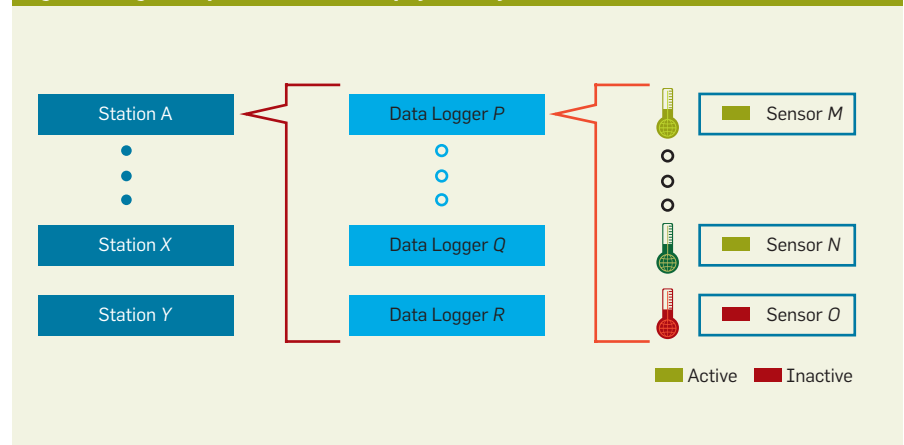
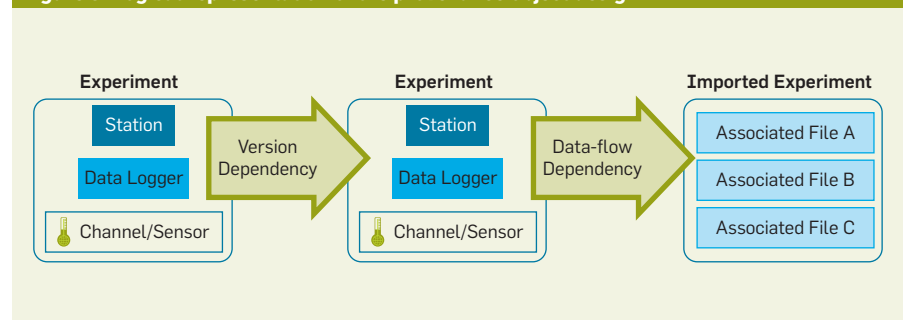


Figure 3. Logical representation of the provenance object design.



dencies are used for objects created as new versions of previous objects; data-flow dependencies are used as ancestry links between differing objects, representing a translation of data between them. The differences between these two types of links are very important for CPL's cycle-avoidance algorithm (discussed in more detail later in the section covering provenance visualization).

Architecturally, ProvDMS has a layered design. Figure 4 is a diagram showing its layers and components. The compatibility layer includes two wrappers: a PHP wrapper and a C++

wrapper the PHP wrapper interacts with. The C++ wrapper abstracts the provenance back-end interaction. The different components interact cohesively:

- The ProvDMS layer represents the user interface for provenance interaction. Figure 5 shows the experiment creation interface, which allows users to select subsets of data for experiment objects and to interact with managed sensor data, visualize provenance information, and either define or derive experiments.

- The compatibility layer abstracts the API calls of CPL so ProvDMS can

interact with the underlying system. Using the PHP wrapper, the system can pass queries from the coupled software to the database back end, as well as share those results.

- The provenance database is the storage layer of ProvDMS. The interactions between the database and the compatibility layer allow for provenance information to be gathered when users define or derive experiment objects while interacting with ProvDMS.

- The sensor database stores FRP sensor data, as well as stored procedures for fast querying when needed. Most data retrieved from this layer is joined with particular FRP stations or data loggers before transmission. It is independent of the provenance system in order to facilitate decoupled scalability and interfacing with other software components being developed for the FRPs.

Using CPL allows ProvDMS to act independently of provenance-calling API hooks when information has to be saved to the provenance database. An abstraction layer handles the translation of user actions to CPL API calls for inserting or querying provenance information. This is encapsulated into a compatibility layer containing the PHP and C++ wrappers.

The PHP code in Figure 6 demonstrates the wrapper's interaction with C++ in order to store or retrieve provenance data.

CPL, which is written in C, already includes some C++ functionality. The C++ wrapper abstracts the interaction with CPL via a heavily object-oriented interface. The code snippet in Figure 7 illustrates the creation of provenance objects. As illustrated, PHP communicates with the C++ wrapper using exec calls. The decision to forgo a PHP extension was based on a few driving factors:

Trade-off. The trade-off between decoupled generality and performance overhead of exec calls, especially for a small number of them, leaned toward a PHP exec framework rather than a full PHP extension.

Simplicity. Using an exec call to an external C++ executable made it possible to maintain a simple parameter-based call similar to that of bash.

Source. Including the C++ wrapper as an external executable while provid-

Figure 4. The layers and components of ProvDMS.

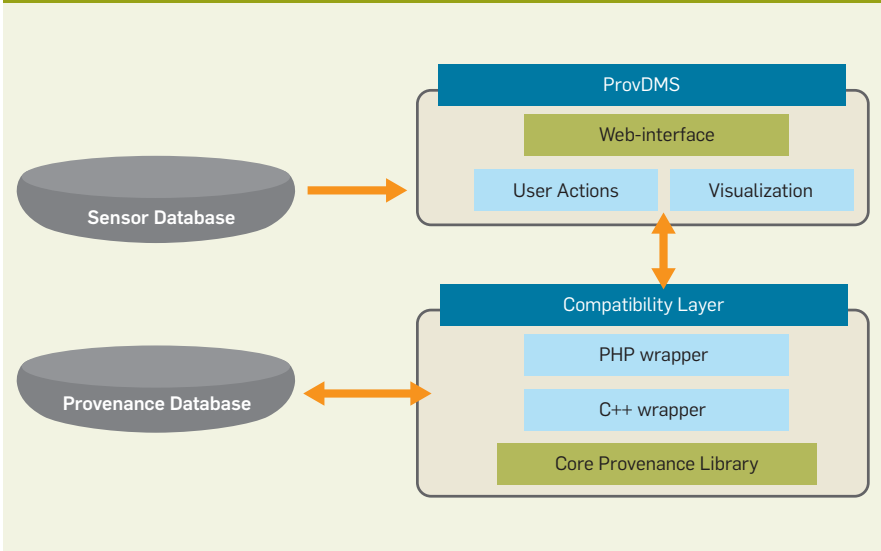
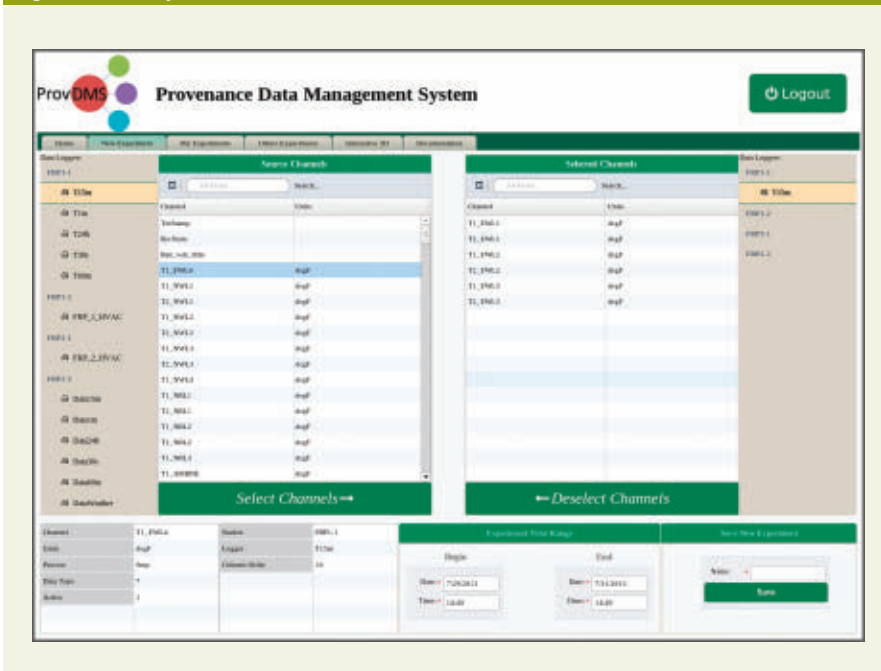


Figure 5. The experiment creation interface.



ing source code allows administrators to modify the wrapper based on organizational needs.

Time to Implement. ProvDMS was designed and implemented in eight to nine weeks, and we made the best of the rapid development. A complete PHP extension implementation was outside the scope of the allocated time and budget for the project.

The integration with CPL was among the smoothest parts of ProvDMS's implementation. Some minor differences in testing and using CPL-integrated systems on different client and server platforms exist. Open-SUSE 12.3 was used for development and testing of ProvDMS, and Red Hat Enterprise Linux 6 for the production version.

One of the earliest hitches involved interactions between PHP and exec'd C++ calls. In order for CPL to provide PoP, it must pull some information from the executing environment. This works perfectly for client-side execution of CPL code, but once the CPL code is executed via PHP exec calls, certain environment variables are no longer retrievable. These variables are necessary to save information about provenance sessions, and thus the provenance back end cannot continue. A quick hot-fix to pass in proper environment information evaded the pull from environment variables.

Features and Usage

ProvDMS was built not just to trace the provenance of experiments, but also to be a one-stop access point for all sensor data-related activities for the FRPs. It provides the following interface features:

Experiment creation. ProvDMS allows users to select subsets of stations, data loggers, and channels as a definition of a new experiment. This information is parsed and saved as CSV files on the server. On request, users can export this data. On creation, each experiment is defined as a provenance object by the provenance back end—creating all finer-granularity objects in addition.

Experiment derivation. Users upload and define experiments as derivations of previous experiments. This means users can save the state of their data and any associated files in ProvDMS, allowing them to trace the deriva-

Figure 6. The wrapper's interaction with C++.

```
Function prov_new_experiment ( $experiment )
{
    global $comman;
    global $userID;

    $retInfo = Array() ;
    $retStatus = null ;

    $exp = new Provenance_Object ( $experiment [
        'name' ], "Experiment" );
    $exp->addPrperty ( 'time_begin' , $experiment [
        'time_begin' ] ) ;
    $exp->addProperty( 'time_end' , $experiment [
        'time_end' ] );

    $params = ' ' ;
    $params . = ' -c "create_object"';
    $params . = ' -n "' . $exp->getName() . '"';
    $params . = ' -t "' . $exp->getType() . '"';
    $params . = ' -o "' . $exp->getOriginator() . '"';
    $params . = ' -p "' . $exp->getProperties() . '"';
    $params . = ' -u "' . $userID . '"';
    $params . = ' -s "Yes"';

    //
    Soft-create, make new version if already exists

    exec ( $command . $params , $retInfo , $retStatus );
}
```

Figure 7. The creation of provenance objects.

```
bool hook_create_object( const char * user, Prov_Params
    params)
{
    int pid = getpid();

    odbcHandler * handler = new odbcHandler( "CPL",
        true, user, pid );
    handler->new_object( params );
    delete handler;

    return true;
}
```

tion in the future.

Data status. The system provides a dashboard with Sparklines,⁹ which helps summarize the status of data on the server. Sparklines are small trending plots that have no axes labels, allowing them to be embedded inline with text, thus permitting users to pick out trends in the data easily. Sparklines can display the status of key channels from different sensors for quick assessment and detection of faults in the system.

Provenance visualization. The system provides visualization capabilities so users can easily visualize their data's lineage. The subject of provenance visualization warrants a separate discussion, covered in more detail in the next section.

Much of the early development of ProvDMS was spent ensuring it is natural and simple to use. For example, the experiment creation feature (Figure 5) is designed with effective user interaction principles to enable a simple “flow,” and it emphasizes the importance of efficiency when managing user data.

Provenance Visualization

The first question anyone should ask when beginning visualization is similar to the first question that should be asked when designing a provenance system: “What information is important?”

The developers of CPL suggest the use of their Orbiter tool to visualize

Figure 8. Logical representation of a subset of provenance data.

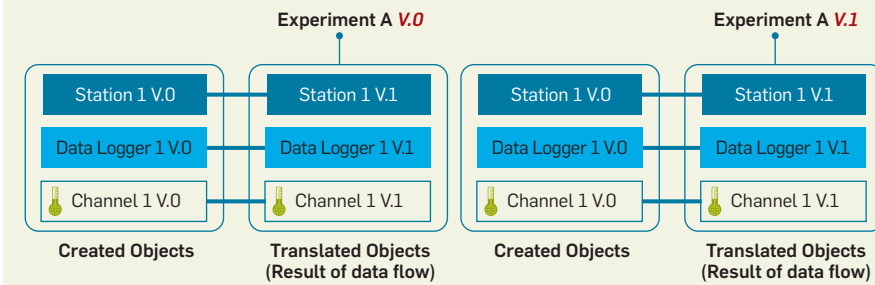
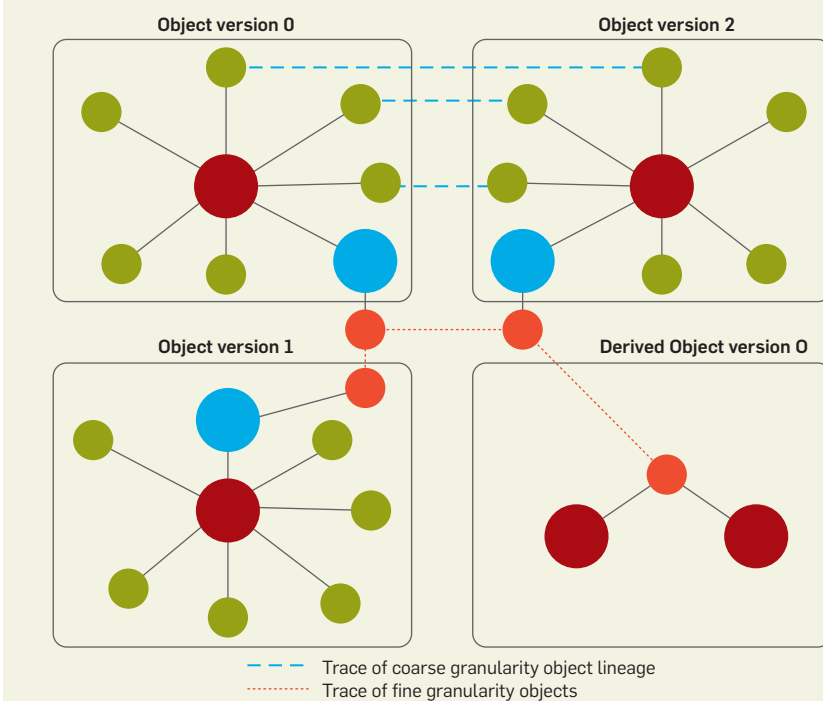


Figure 9. The force-based node-link layout.



provenance using CPL. Orbiter is an external visualization program developed in Java. It pulls information from the CPL database (an SQL back end in this case) and visualizes it using a node-link graph. It includes features for time-based visualization and node grouping for nodes with common links. It is an excellent tool for visualizing the information from the CPL database.

As easy as it would have been to make Orbiter ProvDMS's visualization tool, there are some issues. Primary among these is CPL's use of a cycle-avoidance algorithm to version and link objects without creating cycles in object provenance. Contextual infor-

mation must be displayed as part of the visualization. This means removing particular information from CPL's ancestry queries. Figure 8 shows a subset of provenance information created by ProvDMS and its integration with CPL. Two versions of the finer-granularity objects exist as a result of data-flow dependencies and the cycle-avoidance algorithm. These extra nodes must be removed for clean visualization of the provenance. This information is correct in its representation, but many of the objects are not important to users and can obfuscate the data's lineage in the visualization. For a clearer representation of the provenance, the dou-

ble versions created via the translated objects as a result of cycle avoidance must be removed.

It is important to note how specific these parsed cases are. In the figure, the experiment objects are missing the extra translation versions because these experiments are linked only via version dependencies. This means a user has created a new experiment with the same identification as a previous experiment. This is a cue for ProvDMS's integration with CPL to create a new version of this experiment. This procedure bypasses the need to link objects manually via data-flow dependencies. A situation like this increases the difficulty in parsing individual cases for visualization.

Types of Visualization

ProvDMS's visualization is Web enabled using various JavaScript visualization libraries such as the JIT (JavaScript InfoVis Toolkit) and D3.js (Data Driven Documents). ProvDMS includes a few types of visualizations:

Non-unique node-link tree. Objects in CPL's provenance implementation are inherently unique because of the Name, Type, Originator object convention. Though objects are initially created uniquely, the nature of provenance is to provide a hierarchical flow of data. Objects will undoubtedly have multiple versions at some point in their life cycles. Multiversed objects do not change their identification from one version to another. As only their version changes, the nodes are no longer uniquely identified using the same convention for this type of visualization.

Force-based node-link layout. Classical approaches to the visualization of provenance focus on tree-like views rooted from the top-level provenance object (often a selected one). CPL's objects are designed to use this type of inheritance as well, relying on descendants and ancestors for the traversal of provenance information. Even then, it can be useful to visualize the lineage of objects differently, such as employing a force-based layout. This layout still uses a node-link format—as the other ones do—but it uses a system of forces acting on each node to determine their positions. This makes the system feel more interactive, as users have the ability to apply forces to

nodes in the graphs by dragging them.

Figures 9 and 10 demonstrate some interesting results of this type of visualization. In Figure 9, a traceable flow of data lineage is visible, as well as a natural grouping of objects with similar granularity. Solid gray lines represent hierarchical connections between provenance objects that group together as information relevant to a single version of a user-defined or imported experiment. Orange-colored nodes represent the top-level experiment objects that are parents of all associated finer-granularity objects such as stations, data loggers, and channels (colored blue, red, and green, respectively). In Figure 10, the innermost node and all of the finer-granularity nodes' connections create a pseudo "weight" that encompasses the entirety of the grouping of objects. Each object has its associated weight, which affects the layout

of all connected nodes. The grouping tends to act as a single node in the visualization.

Unique, contextual node-link tree.

The current implementation of visualization in ProvdMS uses this approach in its provenance-visualization module. Similar to the first approach, this one uses a node-link tree to visualize the provenance in a hierarchical fashion. Nodes are expanded asynchronously, pulling information from the provenance database as they do. Contextual information can be shown for certain objects. In this manner, even finer granularity can be visualized by processing provenance object properties in addition to the objects themselves. Figure 11 is an example of this visualization, using contextual node-link trees. Two nodes are expanded to show meta-information at a finer granularity level than their parent nodes.

Experiment nodes are the coarsest objects, while information specific to provenance objects, shown in rectangles, is at the finest granularity level.

Conclusion

This attempt to bring provenance to scientific research has highlighted some of the challenges and potential solutions for applying provenance to generalized data streams. Although we successfully built a system to handle provenance for ORNL's FRPs, this specific use case makes it less general than many other provenance systems. The availability of CPL as a library has been beneficial. The success with using CPL can be attributed to ProvdMS being independent of the provenance back end, providing the required flexibility in system design. The C++ and PHP-wrapper code developed during the project was contributed back to the

Figure 10. A close-up of one of the groupings in the force-based node-link layout.

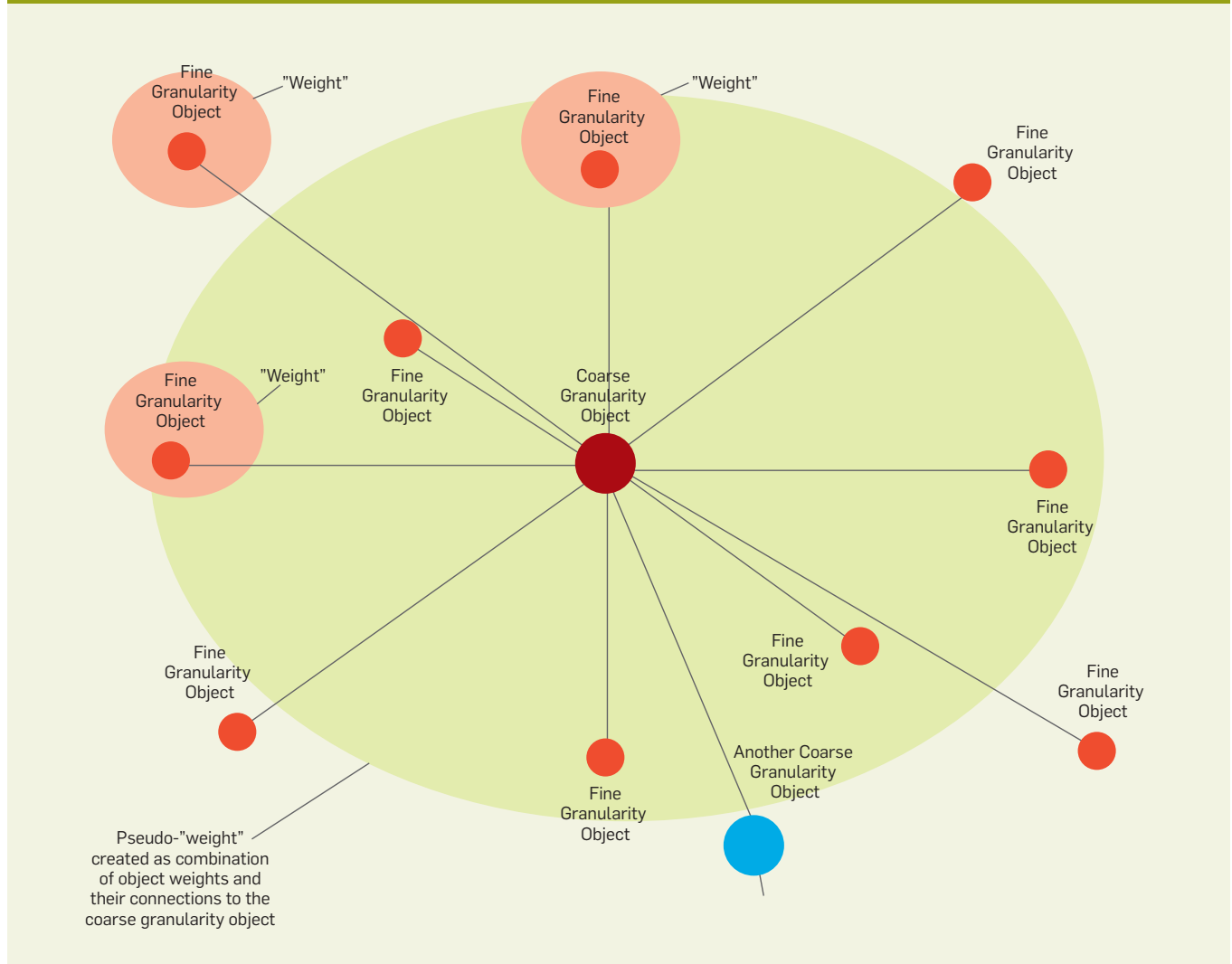
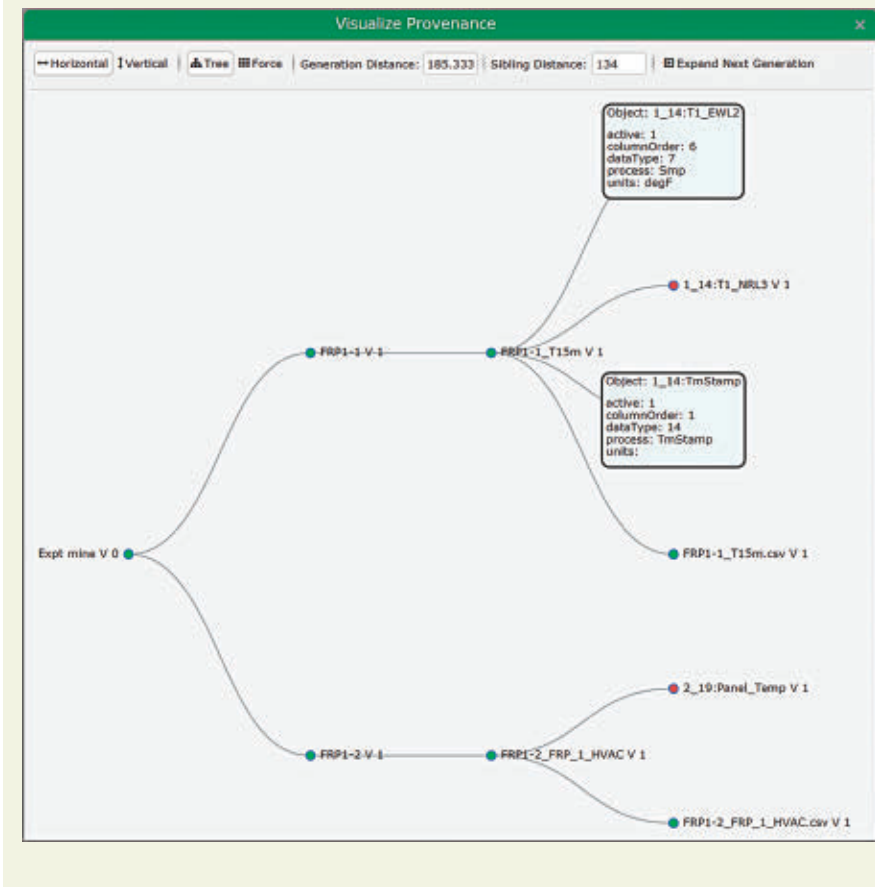


Figure 11. ProvDMS's current visualization using contextual node-link trees.



authors of CPL for future integration.

Research efforts are under way in automated sensor-data validation, estimation for missing or corrupt data, and machine-learning estimations of sensor health with plans to integrate work flows with ProvDMS. The systems will connect to the underlying layers of ProvDMS, allowing integrated tracking of the provenance for data validity, fault detection, and quality assurance.

Despite challenging design decisions, usability both guided and restricted the abilities of ProvDMS. We limited the features and the granularity of collected provenance to ensure minimal restrictions and little additional training required of the researchers. The result is a simple interface for users to keep track of their data and experiments manually. The modular design of ProvDMS will allow the addition of newer provenance-collection methods as the system evolves. The knowledge derived from our experience with ProvDMS's design and use should soon lead to further improvements.

In the end, ProvDMS can serve as

an example of implementing and using provenance of a common data archetype in an environment normally devoid of information-tracking methods. ProvDMS should demonstrate the power of such systems for enabling reproducible science.

Acknowledgments

We thank the authors of CPL—Peter Macko and Margo Seltzer of Harvard University—for their support and guidance on the use of CPL during the project. This work was funded by fieldwork proposal RAEB006 under the Department of Energy Building Technology Activity Number EB3603000. We also thank Edward Vineyard for his support of this project.

Oak Ridge National Laboratory is managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC05-00OR22725. This manuscript has been authored by UT-Battelle, LLC, under Contract Number DEAC05-00OR22725 with the U.S. Department of Energy. The U.S. government retains and the publisher, by

accepting the article for publication, acknowledges the U.S. government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. government purposes. 

Related articles on queue.acm.org

Hazy: Making it Easier to Build and Maintain Big-data Analytics

Arun Kumar, Feng Niu and Christopher Ré
<http://queue.acm.org/detail.cfm?id=2431055>

The Invisible Assistant

Gaetano Borriello
<http://queue.acm.org/detail.cfm?id=1147532>

Self-Healing Networks

Robert Poor, Cliff Bowman and Charlotte Burgess Auburn
<http://queue.acm.org/detail.cfm?id=864027>

References

1. Alvarez, S., Vazquez-Salceda, J., Kifor, T., Varga, L. Z., and Willmott, S. Applying provenance in distributed organ transplant management. *Provenance and Annotation of Data*. Springer, 2006, 28–36.
2. Chapman, A. and Jagadish, H.V. Issues in building practical provenance systems. *IEEE Data Engineering Bulletin* 30, 4 (2007), 38–43.
3. Macko, P. and Seltzer, M. A general-purpose provenance library. In *Proceedings of the 4th Usenix Workshop on the Theory and Practice of Provenance*, 2012.
4. Scheidegger, C., Koop, D., Santos, E., Vo, H., Callahan, S., Freire, J. and Silva, C. Tackling the provenance challenge one layer at a time. *Concurrency and Computation: Practice and Experience* 20, 5 (2008), 473–483.
5. Silva, C.T., Freire, J. and Callahan, S.P. Provenance for visualizations: reproducibility and beyond. *Computing in Science & Engineering* 9, 5 (2007), 82–89.
6. Simmhan, Y.L., Plale, B. and Gannon, D. A survey of data provenance in e-science. *ACM Sigmod Record* 34, 3 (2005), 31–36.
7. Simmhan, Y.L., Plale, B. and Gannon, D. Query capabilities of the Karma provenance framework. *Concurrency and Computation: Practice and Experience* 20, 5 (2008), 441–451.
8. Szomszor, M., Moreau, L. Recording and reasoning over data provenance in Web and grid services. *On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*. Springer, 603–620.
9. Tufte, E. *Sparklines: theory and practice*, 2004; http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=00010R&topic_id=1.

Zachary Hensley is a senior computer science major with a focus on software and scientific applications at Tennessee Technological University. He previously interned at Oak Ridge National Laboratory. The ProvDMS project was his first experience of extensive research into data provenance.

Jibonananda (Jibo) Sanyal is a staff scientist at the Building Technologies Research and Integration Center at Oak Ridge National Laboratory. His current research focus includes massively parallel and scalable energy simulations on high-performance computing systems and subsequent big-data analysis.

Joshua New has been a computer scientist with Oak Ridge National Laboratory since 2009. He coordinates the energy-modeling efforts of ORNL's Building Technology Research Integration Center for EnergyPlus, OpenStudio, and advanced simulation capabilities leveraging high-performance computing and AI for building energy models via ensemble simulations and big data mining.



ICMI 2014

The 16th International Conference on Multimodal Interaction

November 12–16th, 2014

Bogazici University, Istanbul, Turkey

- ✓ **Multimodal Interaction Processing**
- ✓ **Interactive Systems and Applications**
- ✓ **Modelling Human Communication Patterns**
- ✓ **Data, Evaluation and Standards for Multimodal Interactive Systems**
- ✓ **Urban Interactions**

<http://icmi.acm.org/2014>

Organising Committee

General Chairs

Albert Ali Salah (*Boğaziçi University, Turkey*)
Jeffrey Cohn (*University of Pittsburgh, USA*)
Björn Schuller (*TUM / Imperial College London, UK*)

Program Chairs

Oya Aran (*Idiap Research Institute, Switzerland*)
Louis-Philippe Morency (*University of Southern California, USA*)

Workshop Chairs

Alexandros Potamianos (*University of Crete, Greece*)
Carlos Busso (*University of Texas at Dallas, USA*)

Demo Chairs

Kazuhiro Otsuka (*NTT Comm. Science Labs., Japan*)
Lale Akarun (*Boğaziçi University, Turkey*)

Multimodal Grand Challenge Chairs

Dirk Heylen (*University of Twente, The Netherlands*)
Hatice Gunes (*Queen Mary University of London, UK*)

Doctoral Consortium Chairs

Justine Cassell (*Carnegie Mellon University, USA*)
Marco Cristani (*University of Verona, Italy*)

Publication Chairs

Alessandro Vinciarelli (*University of Glasgow, UK*)
Zakia Hammal (*Carnegie Mellon University, USA*)

Publicity Chair

Nicu Sebe (*University of Trento, Italy*)

Sponsorship Chair

Aytül Erçil (*Sabancı University, Turkey*)

Local Organization Chair

Hazim Ekenel (*Istanbul Technical University, Turkey*)

Important Dates

Grand challenge proposals	January 15th, 2014
Special session proposals	March 22nd, 2014
Workshop proposals	March 15th, 2014
Long and short paper submissions	May 9th, 2014
Doctoral consortium submissions	July 1st, 2014
Demo proposals	July 15th, 2014

DOI:10.1145/2560217.2560218

Redundant software (and hardware) ensured Curiosity reached its destination and functioned as its designers intended.

BY GERARD J. HOLZMANN

Mars Code

ON AUGUST 5, 2012, 10:18 P.M. PST, a large rover named Curiosity made a soft landing on the surface of Mars. Given the one-way light-time to Mars, the controllers on Earth learned about the successful touchdown 14 minutes later, at 10:32 P.M. PST. As can be expected, all functions on the rover, and on the spacecraft that brought it to its destination 350 million miles from Earth, are controlled by software. This article discusses some of the precautions the JPL flight software team took to improve its reliability.

To begin the journey to Mars you need a launch vehicle with enough thrust to escape Earth's gravity. On Earth, Curiosity weighed 900 kg. It weighs no more than 337.5 kg on Mars because Mars is smaller than Earth. Curiosity began its trip atop a large Atlas V 541 rocket, which, together with fuel and all other parts needed for the trip, brought the total launch weight to a whopping 531,000 kg, or 590 times the weight of the rover alone.

Within two hours following launch, though, most parts of the launch vehicle had been discarded. At that point, the remaining main parts of the spacecraft included the cruise-stage, the backshell with a large parachute inside, the descent-stage with its intricate sky crane mechanism, the rover, and a large heat shield (see Figure 1).

The cruise-stage was equipped with solar panels to help power the spacecraft during its nine-month trip to Mars, as well as a star tracker to help with navigation, and thrusters to perform small course corrections. All were cast off approximately 10 minutes before the spacecraft entered the Martian atmosphere.

The remaining parts were now all contained within the backshell and protected by the heat shield. The backshell, large enough to hold a small car, had its own set of thrusters to make small course adjustments during the hypersonic entry into the Martian atmosphere. During entry, the backshell cast off several large chunks of ballast mass (weighing some 320 kg) to adjust the center of gravity for the landing at the command of the rover computer that controls the entire mission.

Approximately three minutes before landing the parachute deployed to slow the spacecraft from 1,500 km/h to 300 km/h. The heat shield was ejected, and less than a minute before touchdown the descent stage dropped away from the backshell (see Figure 2). From this point on it was up to the descent stage to guide the rover, with wheels deployed, to the surface (see Figure 3), disconnect itself, and fly away a safe distance to crash. All steps in this sequence were again con-

This image depicts the “fill-packet” transmitted by the Curiosity rover many times each sol (a day on Mars) whenever there is no useful telemetry to send to Earth. The fill packet lists 50 members of the NASA JPL flight software team as well as an in memoriam list of another 18, including the crew of the Challenger and Columbia shuttles and the astronauts killed in a pre-launch test for Apollo 1, and inspirational remarks from astronomer Carl Sagan.

ILLUSTRATION BY BRIAN GREENBERG/ANDRIJ BORYS ASSOCIATES

» key insights

- **The software that controls an interplanetary spacecraft must be designed to a high standard of reliability; any small mistake can lead to the loss of the mission and its unique opportunity to expand human knowledge.**
- **Extraordinary measures are taken in both hardware and software design to ensure spacecraft reliability and that the system can be debugged and repaired from millions of miles away.**
- **Formal methods help verify intricate software subsystems for the potential of race conditions and deadlocks; new model-checking techniques automate the verification process.**

trolled by one of two available computers located within the body of the rover itself.

With each new mission flown to Mars, the size and complexity of both spacecraft hardware and software has increased. The Mars Science Laboratory (MSL) mission, for instance, uses more code than all previous missions to Mars combined, from all countries that have tried to do it. This rapid growth in the size of the software is clearly a concern, but one not unique to this application domain. Unlike most

other software applications, though, the embedded software for a spacecraft is designed for a one-of-a-kind device with an uncommon array of custom-built peripherals. The code targets just one user (the mission), and for the most critical parts of the mission the software is used just once, as in the all-important landing phase, which lasts only minutes. Moreover, the software can be frustratingly difficult to test in an accurate representation of the environment in which it must ultimately operate, yet there are no second chances. The penalty for even a small coding error can be not just the loss of a rare opportunity to expand our knowledge of the solar system, it can also mean the loss of a significant investment and put a serious dent in the reputation of the responsible organization.

Reducing Risk

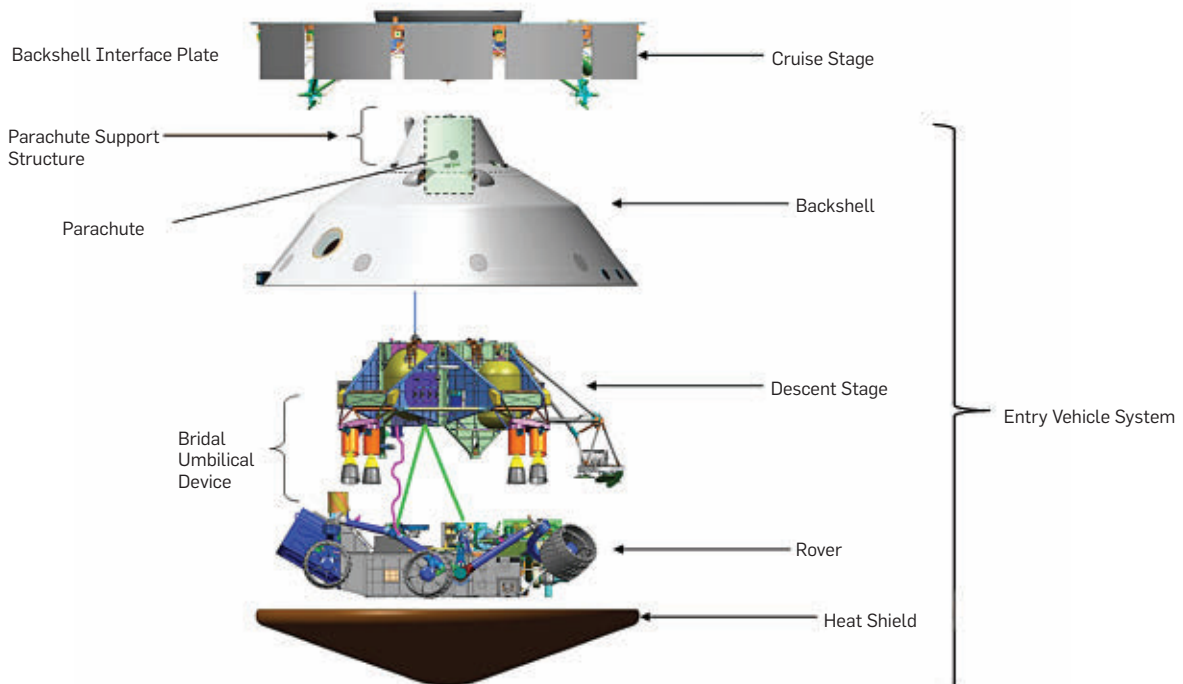
There are standard precautions that can help reduce risk in complex software systems. This includes the definition of a good software architecture based on a clean separation of concerns, data hiding, modularity, well-defined interfaces, and strong fault-protection mechanisms.¹⁸ It also

includes a good development process, with clearly stated requirements, requirements tracking, daily integration builds, rigorous unit and integration testing, and extensive simulation.

This article does not revisit these well-known principles of software design. Instead, it focuses on a different set of precautions the flight software team took in the development of the MSL mission software that is perhaps less common. We restrict ourselves here to three specific topics: First, the coding standard we adopted, which is distinguished by being sparse, risk-based, and supported by automated compliance-checking tools; second, the redefined code-review process we adopted, which allowed us to thoroughly scrub large amounts of code efficiently, again leveraging the use of tools; and third, logic model-checking tools to formally verify mission-critical code segments for the existence of concurrency-related defects.

Risk-based coding rules. No method can claim to prevent all mistakes, but that does not mean we should not try to reduce their likelihood. Before we can do so, though, we have to know what types of mistakes occur

Figure 1. Spacecraft parts.



most often in this domain. Finding the data is not difficult. Most anomalies that have affected space missions are carefully studied and documented, with most information publicly available. We used it to categorize the root causes of each software anomaly to produce a list of the primary areas of concern.

Among them are basic coding and design errors, especially those caused by an undisciplined use of multitasking. Other frequently occurring errors originate in the use of dynamic memory-allocation techniques, which in the early days of space exploration often meant the use of dynamic memory overlays. Finally, the data also shows even standard fault-protection techniques can have unintended side effects that can also cause missions to fail.

The coding standard we developed based on this study differs from many others in that it contained only risk-related, as opposed to style-related, rules.^{9,13} Our view is that coding style (for instance, where curly brackets are placed and how a loop statement is formatted) can be adjusted easily to the preferences of a viewer (or reviewer) using standard code-reformatting tools. Risk-reduction, though, is a consideration that should trump formatting decisions. We used two criteria for inclusion of rules in our new JPL coding standard: First, the rule had to correlate directly with observed risk based on our taxonomy of software anomalies from earlier missions; and second, compliance with the coding rule had to be verifiable with tool-based checks.

Compliance with a coding standard need not be an all-or-nothing proposition; not all code is equally critical to an application. The coding standard we developed therefore recognizes different levels of compliance that apply to different types of software (see Figure 4).

Level-one compliance, or LOC-1, sets a minimal standard of workmanship for all code written at JPL. There are just two rules at this level: The first says all code must be language compliant; that is, it cannot rely on compiler-specific extensions that go outside the language definition proper. For flight software the language standard used at JPL is ISO-C99. The second rule at

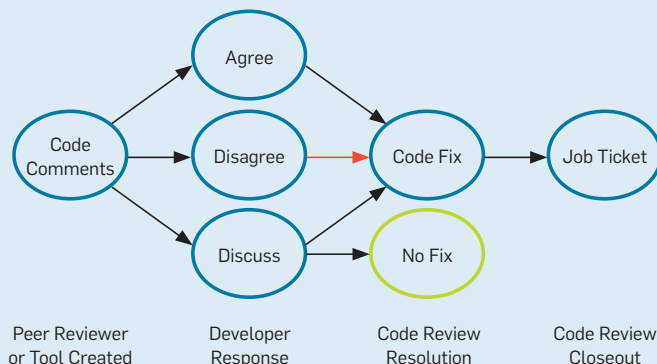
Figure 2. MSL descent stage.



Figure 3. MSL sky crane.



Figure 4. Life cycle of a code comment; orange arrow indicates where the developer disagrees with a code change but is overruled in the final review.



this level requires that all code can pass both the compiler and a good static source code analyzer without triggering warnings. For this test, the compiler is used with all warnings enabled.

LOC-2 compliance adds rules that

are meant to secure predictable execution in an embedded system context. One important rule defined at this level is that all loops must have a statically verifiable upper bound on the number of iterations they can perform.

To reach LOC-3 compliance, one of the most important rules concerns the use of assertions. We originally formulated the rule to require all functions with more than 10 lines of code contain at least one assertion. We later revised it to require that the flight software as a whole, and each module within it, had to reach a minimal assertion density of 2%. There is compelling evidence that higher assertion densities correlate with lower residual defect densities.¹⁴ The MSL flight software reached an overall assertion density of 2.26%, a significant improvement over earlier missions. This rate also compares favorably with others reported in the literature.^{1,7} One final departure from earlier practice was that on the MSL mission all assertions remained enabled in flight, whereas before they were disabled after testing. A failing assertion is now tied in with the fault-protection system and by default places the spacecraft into a predefined safe state where the cause of the failure can be diagnosed carefully before normal operation is resumed.

LOC-4 is the target level for all mission-critical code, which for the MSL mission includes all on-board flight software. Compliance with this level of the standard restricts use of the C pre-processor, as well as function pointers and pointer indirections. The cumulative number of coding rules that must be complied with to reach this level remains relatively low, with no more than 31 risk-related rules.

Figure 5. Coding standard levels of compliance.

LOC-1: language compliance	(2 rules)
LOC-2: predictable execution	(10 rules)
LOC-3: defensive coding	(7 rules)
LOC-4: code clarity	(12 rules)
LOC-5: all MISRA <i>shall</i> rules	(73 rules)
LOC-6: all MISRA <i>should</i> rules	(16 rules)

Safety-critical and human-rated software is expected to comply with the higher levels of rigor defined in LOC-5 and LOC-6. These two highest levels of compliance add all rules from the well-known MISRA C coding guidelines¹⁶ not already covered at the lower levels.

We worked with vendors of static source code analysis tools, including Coverity, Codesonar, and Semmle, to develop automatic compliance checkers for the majority of the rules in our coding standard. Compliance with all risk-based rules could therefore be verified automatically with multiple independent tools on every build of the MSL software.

One additional precaution we undertook starting with the MSL mission was to introduce a new certification program for flight-software developers, allowing us to, for instance, discuss the detailed rationale for all coding rules and reinforce knowledge of defensive coding techniques. The certification program is concluded with an exam, passage of which is required for all developers who write or maintain spacecraft software.

Tool-based code review. Not all software defects can be prevented by even the strongest coding rules, meaning it is important to devise as many methods as possible to intercept the defects that slip through and use them as early and often as possible. One standard mechanism for scrutinizing software is peer code review. Traditionally, in a peer-code-review session, expert developers are invited to provide feedback in a guided code walkthrough. This process can work exceptionally well, but only for relatively small amounts of code. If more than a few hundred lines of code are examined in a single session, the effectiveness of the session, measured by number of flaws exposed, decreases rapidly. Reviewing a few million lines of code in this manner would severely strain the system, if not the reviewers.⁸

Peer reviewers can excel at identifying design flaws but are much less reliable at the more down-to-earth job of checking for mundane issues like rule-compliance and avoidance of common coding errors. Fortunately, this is where static source-code-analysis

tools can prove their value. A static analyzer will not tire of checking for the same types of defects over and over, night after night, patiently reporting all violations. We have therefore made extensive use of this technology.

A wide range of commercial static source-code-analysis tools is on the market, each with slightly different strengths. We found that running multiple analyzers over the same code can be very effective; there is surprisingly little overlap in the output from the various tools. This observation prompted us to run not just one but four different analyzers over all code as part of the nightly integration builds for the MSL mission.

The analyzers we selected—Coverity, Codesonar, Semmle, and Uno—had to be able to identify likely bugs with a reasonably low false-positive rate, handle millions of lines of code efficiently, and allow for the definition of custom checks (such as verifying compliance with the rules from our coding standard). The output of each tool was uniformly reformatted with simple post-processing scripts so all tool reports could be made available within a single vendor-neutral code-review tool we developed, called *Scrub*. The *Scrub* tool was designed to integrate the output of the static analyzers and any other type of background checkers with human-generated peer code review comments in a single user-interface.⁸

In peer code reviews, the reviewers are asked to add their observations to the code in the *Scrub* tool, which is prepopulated with static analysis results from the most recent integration build of the code. The module owner is required to respond to each report, whether generated by a human peer reviewer or by one of the static analysis tools. To respond, the *Scrub* tool allows the module owner to choose from three possible responses: *agree*, meaning the module owner accepts the comment and agrees to change the code to address the concern; *disagree*, meaning the module owner has reason to believe the code as written should not be changed; and *discuss*, meaning the comment or report is unclear and needs clarification before it can be addressed (see Figure 5).

The peer code reviews, and the re-

sponses to all comments and reports, are done offline, outside meetings. Just one face-to-face meeting per module code review is used to resolve disagreements, clarify reports, and reach consensus on the changes to the code that have to be made.

In 145 code reviews held between 2008 and 2012 for MSL flight software, approximately 10,000 peer comments and 30,000 tool-generated reports were discussed.²⁰ Approximately 84% of all comments and tool reports led to changes in the code to address the underlying concerns. There was less than 2% difference in this rate between the peer-generated and the tool-generated reports. Explicit *disagree* responses from the module owner occurred in just 12.3% of the cases. The responses were overruled in the final code review session in 33% of those cases, leading to a required fix anyway. A *discuss* response was given for just 6.4% of all comments and reports, leading to a change in the code in approximately 60% of those cases.

These statistics from the MSL code-review process illustrate that the large majority of comments and tool reports led to immediately agreed-upon changes to the code and did not require discussion in the code review close-out meetings. The time saved allowed us to push the code-review process further than would have been possible otherwise. Critical modules, for instance, could now be reviewed multiple times before the code was finalized for launch.

Model checking. The strongest type of check we have in our arsenal for analyzing multithreaded code is logic model checking. The code for the MSL mission makes significant use of multithreading, with 120 parallel tasks being executed under the control of a real-time operating system. The potential for race conditions therefore always exists and has been a significant cause of anomalies on earlier missions. To thoroughly analyze the code for race conditions, we made extensive use of the capabilities of the logic model checker *Spin*,¹⁰ together with an extended version of a model extraction tool for C code.¹²

Spin was developed in the Computing Science Research group of Bell Labs starting in the early 1980s and has been freely available since 1989. We earlier



Peer reviewers can excel at identifying design flaws but are much less reliable at the more down-to-earth job of checking for mundane issues like rule-compliance and avoidance of common coding errors.



used this tool on the verification of key parts of the control software for a number of spacecraft, including Cassini,²¹ Deep Space One,^{5,6} and the Mars Exploration rovers.¹¹ We also used it in the recent investigation of possible triggers for unintended acceleration in Toyota vehicles.¹⁷ In almost all these cases, the verification effort succeeded in identifying unsuspected software defects, especially concurrency-related issues that would be very difficult to uncover by other means.

The model checker *Spin* specifically targets verification of distributed-systems software with asynchronous threads of execution. Its internal verification algorithm is based on Vardi and Wolper's automata-theoretic verification method.²³ Informally, *Spin* takes the role of a demonic process scheduler, trying to find system executions that violate user-defined requirements. Simple examples of the type of requirements that can be proven or disproven this way are the validity of program assertions and the absence of deadlock scenarios. But the model checker can also reach farther by verifying more complex requirements on feasible or infeasible program executions that can be expressed in linear temporal logic.¹⁹

We analyzed several critical software components for the MSL mission, including a dual-CPU boot-control algorithm (the algorithm that controls which of two available CPUs will take control of the spacecraft when it boots), the nonvolatile flash file system, and the data-management subsystem. Several vulnerabilities identified through these analyses could be eliminated from the code before the mission was launched, effectively helping reduce the risk of in-flight surprises. The basic procedure of software model checking, using the tools we developed, can be illustrated with a small example. (Because NASA rules prevent us from publishing actual flight code from the rover, we use equivalent public-domain code for this example.)

It can be unreasonably difficult to prove manually that a concurrent algorithm is correct under all possible execution scenarios. We take as our example a non-blocking algorithm for two-sided queues presented in Dettlefs et al.² together with a four-page

Figure 6. Semantics of the DCAS instruction.

```

boolean DCAS (val *addr1, val *addr2,
              val old1, val old2,
              val new1, val new2)
{
    atomically {
        if (*addr1 == old1 && *addr2 == old2)
        {
            *addr1 = new1;
            *addr2 = new2;
            return true;
        } else
        {
            return false;
        }
    }
}

```

Figure 7. C code for pushRight and popRight routines.

```

1 Node *Dummy, *LH, *RH;
2
3 val
4 pushRight(val v)
5 {   Node *nd, *rh, *lh, *rhR;
6
7     nd = (Node *) spin_malloc(sizeof(Node));
8
9     if (!nd) return FULL;
10
11     nd->R = Dummy;
12     nd->V = v;
13
14     while (true)
15     {   rh = RH;
16         rhR = rh->R;
17         if (rhR == rh)
18         {   nd->L = Dummy;
19             lh = LH;
20             if (DCAS(&RH, &LH, rh, lh, nd, nd))
21                 return OKAY;
22         } else
23         {   nd->L = rh;
24             if (DCAS(&RH, &rh->R, rh, rhR, nd, nd))
25                 return OKAY;
26         }
27     }
28
29 val
30 popRight(void)
31 {   Node *rh, *lh, *rhL;
32     val result;
33
34     while (true)
35     {   rh = RH;
36         lh = LH;
37
38         if (rh->R == rh)
39             return EMPTY;
40
41         if (rh == lh)
42         {   if (DCAS(&RH, &LH, rh, lh, Dummy, Dummy))
43             return rh->V;
44         } else
45         {   rhL = rh->L;
46             if (DCAS(&RH, &rh->L, rh, rhL, rhL, rh))
47             {   result = rh->V;
48                 rh->R = Dummy;
49                 rh->V = null;
50                 return result;
51             }
52         }
53     }
54 }

```

summary of a proof of correctness. A few years following its publication an attempt was made to formalize that proof with a theorem prover²² as part of a master's thesis project.³ The formalization revealed that both the original proof and the algorithm were flawed. A correction to the algorithm could be proven correct with the theorem prover.⁴ Each proof attempt, for both the original algorithm and the corrected version, reportedly took several months.

Lamport¹⁵ later formalized the original algorithm in +CAL, showing the flaws could be found more quickly through a model checker. Lamport noted the proof with the TLA+ model checker could be completed in less than two days, most of which was needed to define a formal model of the original algorithm in the language supported by the model checker.

As shown here, a model extractor can help avoid the need for manual construction of a formal model as well, allowing us to perform these types of verification on multithreaded code fragments in minutes instead of days. We use the original algorithm from Detlefs² to show how this verification approach works. With it, finding the flaw in the implementation of the algorithm requires no more than typing in a few lines of text and executing a single command.

The algorithm uses an atomic Double-word Compare-And-Swap, or DCAS, instruction; Figure 6 gives the semantics of this instruction as defined in Detlefs.² Figure 7 reproduces two C routines from Detlefs² for adding an element to the right of the queue and for deleting an element from the same side. The routines for adding or deleting elements from the left side of the queue are symmetric. The node structure used has three fields: a left pointer L, a right pointer R, and an integer value V.

To verify the code we first define a simple test driver that exercises the code by adding and deleting elements (see Figure 8). For simplicity, this example uses only the pushRight() and popRight() routines.

In the example test driver in Figure 8, the writer initializes the queue on line 74, and the reader waits until this step is completed on lines 57–59. The

reader contains an assertion on line 64 to verify the values sent by the writer are received in the correct order, without omissions.

We can perform the test using different threads for the reader and the writer, though these tests alone cannot establish the correctness of the algorithm. A model checker is designed to perform this type of check more rigorously. If there is any possible interleaving of the thread executions that can trigger an assertion failure, the model checker is guaranteed to find it. To use the model checker we define a small configuration file that identifies the parts of the code we are interested in. This configuration file allows us to define an execution context for the system we want to verify by extracting the relevant parts of the code and

placing them into an executable system that is then analyzed.

Figure 9 shows the complete configuration file needed to verify this application. The first four lines identify four functions in source file `dcas.c` we are interested in extracting as instrumented function calls. The next two lines identify `sample_reader` and `sample_writer` as active threads that will call these functions. The last three lines in the configuration file define the required header file `dcas.h` that holds the definition of data structure `Node` and the name of the source file (`dcas.c`) to which the verifier must be linked for additional routines, including a C encoding of the function that defines the semantics of the DCAS instruction (also shown in Figure 6).

The verification of the algorithm can now be performed with a single command, using the model-extraction tool *Modex* and the model checker *Spin* (see Figure 10).

The command takes approximately 12 seconds of real time to execute, of which only 0.02 seconds is needed for the verification itself. The rest of the runtime is taken by the model extractor to generate the verification model from the source code, for *Spin* to con-

vert that model into optimized C code, and finally for the C compiler to produce the executable that performs the verification. None of these steps requires further user interaction.

A replay of the error-trail reveals a race condition that can lead to an assertion violation and therefore shows the algorithm to be faulty (see figures 11, 12, and 13). Statements executed by the writer process are marked with *W* and statements executed by the reader process with *R*. First consider Figure 11. After the initial call to `initialize` in the `sample_writer` routine (line 74 in Figure 8), the writer initiates its first call to `pushRight` on line 77, with value 0. This value is then stored by executing lines 7 through 19 in the `pushRight` routine.

The next statement in the execution of `pushRight` would now be a call on DCAS to complete the update, but that call is delayed. Meanwhile, the `sample_reader` is free to proceed with calls to `popRight` to poll the queue for new elements (see Figure 12). The first call (line 62 in Figure 8) succeeds and retrieves the stored value 0. The remaining steps in Figure 12 illustrate the execution of the `popRight` routine for that call.

Figure 8. C code for a sample test driver.

```

53 void
54 sample_reader(void)
55 {   int i, rv;
56
57     while (!RH)
58     {   /* wait */
59     }
60
61     for (i = 0; i < 10; i++)
62     {   rv = popRight();
63         if (rv != EMPTY)
64         {   assert(rv == i);
65             } else
66             {   i--;
67             }
68     }
69
70 void
71 sample_writer(void)
72 {   int i, v;
73
74     initialize();
75
76     for (i = 0; i < 10; i++)
77     {   v = pushRight(i);
78         if (v != OKAY)
79         {   i--;
80         }
81     }

```

Figure 9. Modex configuration file.

```

%X -e pushRight
%X -e popRight
%X -e initialize
%X -e dcas_malloc
%X -a sample_reader
%X -a sample_writer
%D
#include "dcas.h"
%O dcas.c

```

Figure 10. Verification steps.

```

$ time modex -run dcas.c
MODEX Version 2.0 - 2 September 2011
c_code line 111 precondition false:
(Psample_reader->rv==Psample_reader->i)
wrote model.trail
...
pan: elapsed time 0.02 seconds

7.69 user 4.02 system 0:12.04 elapsed 97% CPU
$

```

Figure 11. Part 1, partial execution of pushRight by the test writer.

```

74 W: initialize()
76 W: i = 0
76 W: (i<10)
77 W: # v = pushRight(i) ::
7 W:   nd = (Node *) spin_malloc(sizeof(Node));
9 W:   !(!nd)
11 W:   nd->R = Dummy;
12 W:   nd->V = v;
14 W:   (true)
15 W:   rh = RH;
16 W:   rhR = rh->R;
17 W:   (rhR == rh)
18 W:   nd->L = Dummy;
19 W:   lh = LH;

```

This call should not succeed because the `pushRight` call, initiated by the writer in Figure 11, has not yet completed its update. But the trap has now been set. The `sample_reader` thread now moves on to the next call, after incrementing the value of `i`. This second call to `popRight` completes the same way it did before and again returns the value 0, resulting in the failure (see Figure 13).

The model-extraction method used here is defined in such a way it allows for very simple types of instrumentation in basic applications. The model extractor always preserves the application's original control flow. However, it also supports the definition of more advanced abstraction functions in con-

figuration files (similar to the one in Figure 9) that can be used to reduce the complexity of extracted models. The default conversion rule, which defines a one-to-one mapping of statements from the source code into the model, allows for direct verification of a surprisingly large set of multithreaded C programs and algorithms.

The MSL mission made extensive use of this automated capability to verify critical multithreaded algorithms, directly using their implementation in C. For larger subsystems, we also manually constructed *Spin* verification models in a more traditional way and analyzed them. The largest such MSL subsystem was a critical data-management mod-

ule implemented in approximately 45,000 lines of C. The design of this subsystem was converted manually into a *Spin* verification model of approximately 1,600 lines, in close collaboration with the module designer. In most cases, the model-checking runs successfully identified the existence of subtle concurrency flaws that could be remedied in the software. For the file system software in particular, the model-checking runs became a routine part of our regression “tests,” executed after every change in the code, often surprising us with the ease with which it could identify newly committed coding errors.

Conclusion

The MSL spacecraft performed flawlessly in delivering Curiosity to the surface of Mars in August 2012 where it is currently exploring the planet (see Figure 14). The rover has meanwhile achieved its primary mission, which was to determine if our neighbor planet could in principle have supported life in the distant past.

Every precaution was taken to optimize the chances of success, and not just in the development of the software. Critical hardware components were duplicated, including the rover's main CPU. But though it is not difficult to see how duplication of an essential hardware component helps improve system reliability, seeing how one can use redundancy to improve software reliability is less simple.

We gave two examples of how software redundancy was nonetheless used on the MSL mission. The first—emphasis on use of assertions throughout the code—may sound obvious but is rarely recognized as a pro-

Figure 12. Part 2, call to `popRight` by the test reader.

```
57 R: !(RH)
61 R: i = 0
61 R: (i < 10)
62 R: # rv = popRight() ::
34 R: (true)
35 R: rh = RH;
36 R: lh = LH;
38 R: !(rh->R == rh)
41 R: (rh == lh)
42 R: DCAS(&(RH), &(LH), rh, lh, Dummy, Dummy)
43 R: return rh->V;
```

Figure 13. Part 3, second call to `popRight` by the reader, with the writer still stalled in its first call to `pushRight`, leading to the assertion violation.

```
62 R: rv = popRight(i) # rv is 0
63 R: (rv != EMPTY) # true
64 R: assert(rv == i) # true
61 R: i++; # i is now 1
61 R: (i<10) # true
62 R: rv = popRight() # rv is again 0
63 R: (rv != EMPTY) # true
64 R: assert(rv == i) # false
```

Figure 14. First MSL wheel tracks on Mars.



tection mechanism based on redundancy. An assertion is always meant to be satisfied, meaning that technically its evaluation is almost always redundant. But sometimes the impossible does happen, as when, say, external conditions change in unforeseen ways. Assertions prove their value by detecting off-nominal conditions at the earliest possible point in an execution, thus allowing fault-protection monitors to take action and prevent damage.

The second example of software redundancy was used to protect the critical landing sequence. This was the only phase of the mission in which both the main CPU and its backup were used simultaneously, with the backup in hot standby. Running the same landing software on two CPUs in parallel offers little protection against software defects. Two different versions of the entry-descent-and-landing code were therefore developed, with the version running on the backup CPU a simplified version of the primary version running on the main CPU. In the case where the main CPU would have unexpectedly failed during the landing sequence, the backup CPU was programmed to take control and continue the sequence following the simplified procedure. The backup version of the software was aptly called “second chance,” and to everyone’s relief proved itself redundant by never being called on to execute.

Acknowledgments

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, under a contract with the National Aeronautics and Space Administration. Credit for the nearly flawless performance of the MSL flight software to date goes to the superb software development team that created, reviewed, analyzed, tested, and retested the code, working countless hours. 

References

- Chalin, P. *Ensuring Continued Mainstream Use of Formal Methods: An Assessment*. Roadmap and Issues Group, D.S.R., TR 2005-001, Concordia University, Montréal, Canada, 2005.
- Detlefs, D.L., Flood, C.H., Garthwaite, A.T. et al. Even better DCAS-based concurrent dequeues. In *Distributed Algorithms, LNCS Vol. 1914*, M. Herlihy, Ed. Springer Verlag, Heidelberg, 2000, 59–73.
- Doherty, S. *Modelling and Verifying Non-blocking Algorithms that Use Dynamically Allocated Memory*. Master’s Thesis, Victoria University, Wellington, New Zealand, 2004.
- Doherty, S., Detlefs, D.L., Groves, L. et al. DCAS is not a silver bullet for nonblocking algorithm design. In *Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, P.B. Gibbons and M. Adler, Eds. (Barcelona, Spain, June 27–30). ACM Press, New York, 2004, 216–224.
- Gluck, P.R. and Holzmann, G.J. Using Spin model checking for flight software verification. In *Proceedings of the 2002 Aerospace Conference* (Big Sky, MT, Mar. 9–16). IEEE Press, Piscataway, NJ, 2002.
- Havelund, K., Lowry, M., Park, S. et al. Formal analysis of the remote agent: Before and after flight. *IEEE Transactions on Software Engineering* 27, 8 (Aug. 2001), 749–765.
- Hoare, C.A.R. Assertions: A personal perspective. *IEEE Annals of the History of Computing* 25, 2 (Apr.–June 2003), 14–25.
- Holzmann, G.J. Scrub: A tool for code reviews. *Innovations in Systems and Software Engineering* 6, 4 (Dec. 2010), 311–318.
- Holzmann, G.J. The power of ten: Rules for developing safety critical code. *IEEE Computer* 39, 6 (June 2006), 95–97.
- Holzmann, G.J. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, Boston, 2004.
- Holzmann, G.J. and Joshi, R. Model-driven software verification. In *Proceedings of the 11th Spin Workshop, LNCS 2989* (Barcelona, Spain, Apr. 1–3). Springer Verlag, Berlin, 2004, 76–91.
- Holzmann, G.J. and Smith, M.H. Automating software feature verification. *Bell Labs Technical Journal* 5, 2 (Apr.–June 2000), 7–87.
- Jet Propulsion Laboratory. *JPL Coding Standard for Flight Software*; http://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_C.pdf
- Kudrjavets, G., Nagappan, N., and Ball, T. Assessing the relationship between software assertions and faults: An empirical investigation. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering* (Raleigh, NC, Nov. 7–10). IEEE Press, Piscataway, NJ, 2006, 204–212.
- Lamport, L. Checking a multithreaded algorithm with +CAL. In *Proceedings of Distributed Computing: 20th International Conference* (Stockholm, Sweden, Sept. 18–20). Springer-Verlag, Berlin, 2006, 151–163.
- Motor Industry Software Reliability Association. *MISRA-C Guidelines for the Use of the C Language in Critical Systems*. MIRA Ltd., Warwickshire, U.K., 2012; <http://www.misra-c.com/>
- NASA. *NASA Engineering and Safety Center, Technical Assessment Report*. National Highway Traffic Safety Administration (NHTSA), Toyota Unintended Acceleration Investigation, Appendix A: Software, Washington, D.C., Jan. 18, 2011; http://www.nhtsa.gov/staticfiles/nvs/pdf/NASA_FR_Appendix_A_Software.pdf
- Ong, E.C. and Leveson, N. Fault protection in a component-based spacecraft architecture. In *Proceedings of the International Conference on Space Mission Challenges for Information Technology* (Pasadena, CA, July 13–16). Jet Propulsion Laboratory, Pasadena, CA, 2003.
- Phueli, A. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (Providence, RI, Oct. 31–Nov. 1). IEEE Computer Society, Washington, D.C., 1977, 46–57.
- Redberg, R. and Holzmann, G.J. *Reviewing Code Review*. LaRS Report, Jet Propulsion Laboratory, Pasadena, CA, Nov. 2013.
- Schneider, F., Easterbrook, S.M., Callahan, J.R., and Holzmann, G.J. Validating requirements for fault-tolerant systems using model checking. In *Proceedings of the International Conference on Requirements Engineering* (Colorado Springs, CO, April 6–10). IEEE Computer Society, Washington, D.C., 1998, 4–13.
- SRI International, Computer Science Laboratory. *The PVS Specification and Verification System*; <http://pvs.csl.sri.com/>
- Vardi, M. and Wolper, P. An automata-theoretic approach to automatic program verification. In *Proceedings of the First IEEE Symposium on Logic in Computer Science* (Cambridge, MA, June 16–18). IEEE Computer Society, Washington, D.C., 1986, 332–344.

Gerard J. Holzmann (gholzmann@acm.org) is a senior research scientist and a fellow at NASA’s Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA.

© 2014 ACM 0001-0782/14/02 \$15.00

DOI:10.1145/2560217.2560219

The idea is to identify security-critical software bugs so they can be fixed first.

BY THANASSIS AVGERINOS, SANG KIL CHA, ALEXANDRE REBERT, EDWARD J. SCHWARTZ, MAVERICK WOO, AND DAVID BRUMLEY

Automatic Exploit Generation

ATTACKERS COMMONLY EXPLOIT buggy programs to break into computers. Security-critical bugs pave the way for attackers to install trojans, propagate worms, and use victim computers to send spam and launch denial-of-service attacks. A direct way, therefore, to make computers more secure is to find security-critical bugs before they are exploited by attackers.

Unfortunately, bugs are plentiful. For example, the Ubuntu Linux bug-management database listed more than 103,000 open bugs as of January 2013. Specific widely used programs (such as the Firefox Web browser and the Linux 3.x kernel) list 7,597 and 1,293 open bugs in their public bug trackers, respectively.^a Other projects, including those that are closed-source, likely involve similar statistics. These are just the bugs we know; there is always the persistent threat of zero-day exploits, or attacks against previously unknown bugs.

Among the thousands of known bugs, which should software developers fix first? Which are exploitable?

^a All bug counts exclude bugs tagged as “wishlist,” “unknown,” “undecided,” or “trivial.”

How would you go about finding the unknown exploitable ones that still lurk?

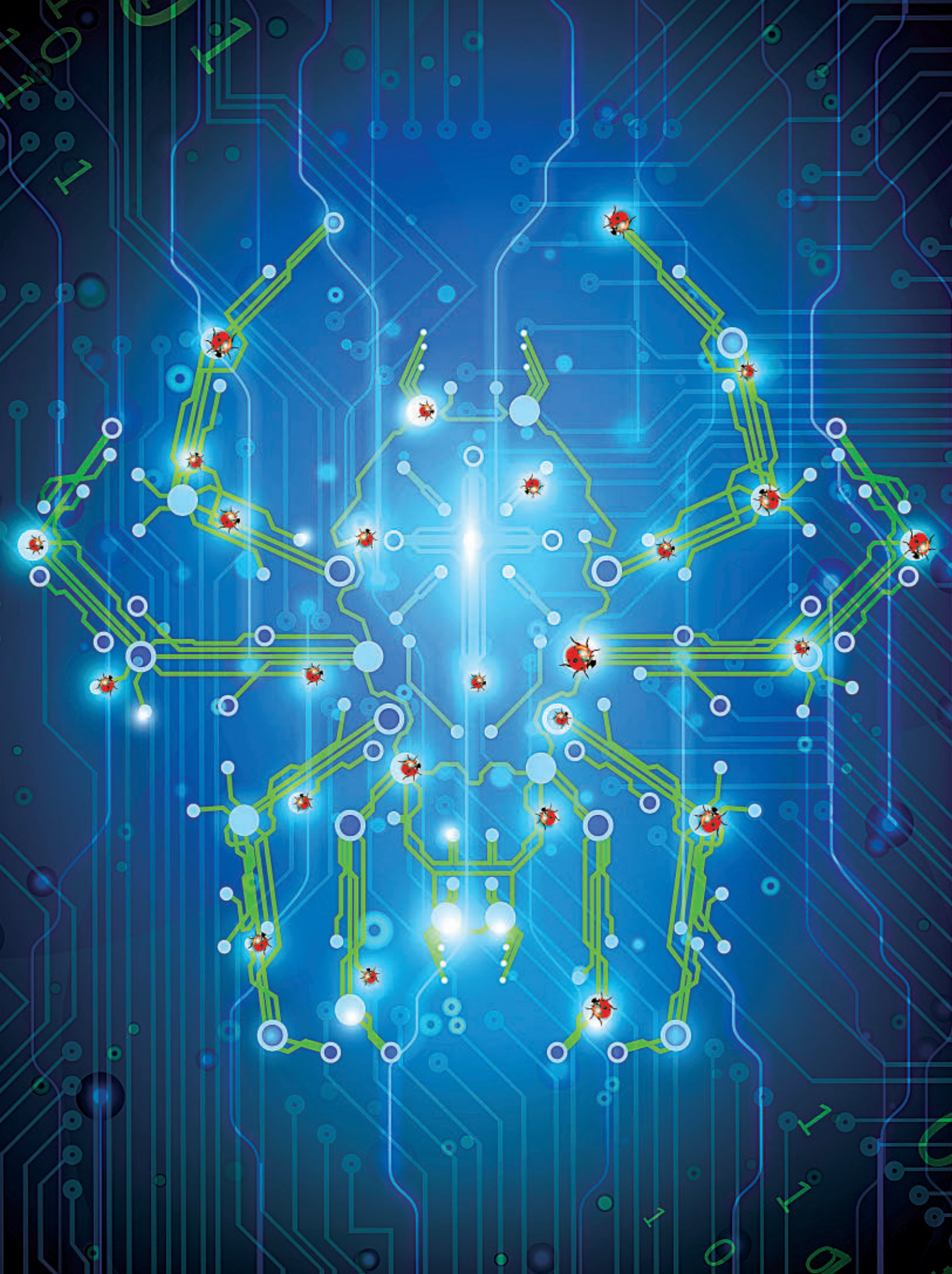
Given a program, the automatic exploit generation (AEG) research challenge is to both automatically find bugs and generate working exploits. The generated exploits unambiguously demonstrate a bug is security-critical. Successful AEG solutions provide concrete, actionable information to help developers decide which bugs to fix first.

Our research team and others cast AEG as a program-verification task but with a twist (see the sidebar “History of AEG”). Traditional verification takes a program and a specification of safety as inputs and verifies the program satisfies the safety specification. The twist is we replace typical safety properties with an “exploitability” property, and the “verification” process becomes one of finding a program path where the exploitability property holds. Casting AEG in a verification framework ensures AEG techniques are based on a firm theoretic foundation. The verification-based approach guarantees sound analysis, and automatically generating an exploit provides proof that the reported bug is security-critical.

Verification involves many well-known scalability challenges, several of which are exacerbated in AEG. Each new branch potentially doubles the number of possible program paths, possibly leading to an explosion of paths to check for exploitability. Traditional verification takes advantage of source code, models, and other abstractions to help tackle the state explosion and scale. Unfortunately, ab-

» key insights

- This research formalizes the notion of an exploit, allowing for automated reasoning about exploitation.
- The technology can be used to identify and prioritize security-critical bugs.
- Improvements for verifying programs safe may also lead to improvements for automatically generating exploits.



stractions often leak by not perfectly encapsulating all security-relevant details, and the leaky points tend to affect the quality of security analysis. For example, writing 12B to an array declared to be 11B long is wrong in C but is also unlikely to be exploitable because most compilers would pad the array with extra bytes to word-align memory operations.

In order to provide high fidelity, most AEG work analyzes raw executable code. Executable code analysis is needed because many exploits rely on low-level details that are abstract in source code (such as CPU semantics and memory layout). Executable code analysis is also attractive because it is widely applicable; users typically have access to the executable code of the programs they run (as opposed to source code) and thus can audit the code for security-critical bugs.

Throughout this article, we focus on AEG as a defensive tool for prioritizing exploitable bugs. However, we are also cognizant of the obvious offensive computing implications and applications as well. Governments worldwide are developing computer-warfare capabilities, and exploits have become a new type of ammunition. At present, exploit generation in practice is mostly a manual process. Therefore, techniques that help re-

duce the time and effort for exploit generation can potentially affect a nation's operational capabilities. AEG research is in its infancy and not yet at the point of automatically churning out weapons-grade exploits for an arbitrary program. Most reported research results generate exploits against bugs up to a few thousand lines deep in execution and for relatively straightforward bugs, while typical offensive needs include exploits for complicated bugs and large programs like Internet Explorer and Adobe Reader. Nonetheless, current AEG results show promise, and a conservative defensive security position must consider the possibility of real-world offensive AEG capabilities.

This article describes our AEG research at Carnegie Mellon University, its successes, as well as its current limitations. We focus primarily on control-flow hijack exploits that give an attacker the ability to run arbitrary code. Control-flow hijacks are a serious threat to defenders and coveted by attackers.^{3,35} Although most current research focuses on control-flow hijacks due to their immediate danger, AEG is not limited to only this class of attacks. Exploitable bugs are found in programs in all languages, and the verification-based approach to AEG still applies.

Exploiting Programs

Suppose a developer is interested in finding and fixing exploitable bugs in the `/usr/bin` directory of the latest Debian operating system. For instance, in June 2012 we downloaded the then-current Debian 6.0.5, with (in our installation) 1,168 executables in `/usr/bin` to analyze for exploitable bugs.

A typical approach to finding exploitable bugs is to first find them and then determine which ones are exploitable. One popular way to find bugs is to perform “black-box fuzzing.” Fuzzing is a program-testing technique that runs a program on inputs from a fixed alphabet, often either modifying at random a known input or trying extreme values (such as 0 and the native maximum integer), and the “black-box” refers to the program itself, which is not analyzed at all. The fuzzer chooses the inputs and observes the program, looking for hangs, crashes, buggy outputs, or other indications of a bug.

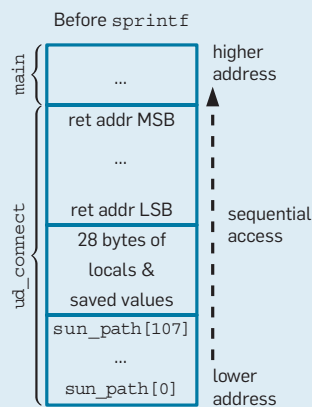
We fuzzed each program using the following script:

```
for letter in {a..z} {A..Z}; do
  timeout -s 9 1s <program>
  -$letter <path>
done
```

The script tries all single-letter com-

Our running example of a buffer overflow in `acpi-listen`.

```
1. int main(int argc, char **argv) {
2.   char *name; int i;
3.   for (;;) {
4.     i = getopt(argc, argv, "c:s:t:vh");
5.     if (i == -1) break;
6.     switch (i) {
7.       case 'c': ...; break;
8.       case 's': name = optarg; break;
9.       ...
10.    }
11.  }
12.  sock_fd = ud_connect(name);
13.  ...
14. }
15. int ud_connect(const char *name) {
16.   int fd;
17.   struct sockaddr_un {
18.     sa_family_t sun_family;
19.     char sun_path[108];
20.   } addr;
21.   ...
22.   sprintf(addr.sun_path, "%s", name);
23.   ...
24.   return fd;
25. }
```



```
00000000 31 c9 f7 e1 51 68 2f 2f 73 68 68 2f 62 69 6e 89 |1...Qh//shh/bin.|
00000010 e3 b0 0b cd 80 41 41 41 41 41 41 41 41 41 41 41 |.....AAAAAAAAAAAA|
00000020 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 |AAAAAAAAAAAAAAAAAA|
*
00000080 41 41 41 41 41 41 41 41 74 f2 ff bf |AAAAAAAt...|
```

mand-line options from a to Z, followed by a valid 6,676B filename. The `timeout` command limited total execution time to one second, after which the program was killed.

The script took about 13 minutes to fuzz all programs on our test machine, yielding 756 total crashes. We identified 52 distinct bugs in 29 programs by analyzing the calling context and faulting instruction of each crash. Which bugs should a developer fix first? The answer is the exploitable ones. For now, we forgo several important issues relevant in practice we tackle later (such as whether the buggy program is a realistic attack target and whether additional operating system defenses would protect the otherwise exploitable program from attack).

We first describe simple manual exploit generation to introduce terminology and give a flavor of how exploits work. We focus on control-flow hijack exploits, which have been a staple class of exploits in the computer-security industry for decades.^{3,35} Well-known examples of control-flow hijacks range from exploits in the Morris worm in 1988 to the more recent Stuxnet and Flame worms (though the latter exploits are much more complicated than those described here).

The figure here shows a bug discovered in `acpi_listen` (now patched in Debian testing) we use as our running example. A buffer overflow occurs on line 22. The program reads in a command-line argument; if it is `-s` (line 8), it assigns the subsequent argument string to the `name` variable. On line 22, the `sprintf` function copies `name` into `sun_path`, a field in a local instance of the networking `sockaddr_un` data type, a standard data structure in Unix for sockets.

The bug is that `sun_path` is a fixed-size buffer of 108B, while the command-line argument copied through `name` into `sun_path` can be any length. The C standard says the execution behavior is undefined if more than 108B are written. When executed, something will happen; with the fuzzing script described earlier, the program crashed. Unfortunately, this crashing bug can be exploited.

All control-flow hijack exploits



The twist is we replace typical safety properties with an “exploitability” property, and the “verification” process becomes one of finding a program path where the exploitability property holds.



have two goals: hijack control of the instruction pointer (IP) and then run an attacker's computation. For `acpi_listen`, some of the details an attacker must understand in-depth include: the hardware execution model (such as how instructions are fetched from memory and executed; how function calls are implemented; how writing outside the allocated space can hijack control of the IP; and how to redirect the IP to run the attacker's code). Since any discussion of creating exploits against vulnerable C programs assumes a basic understanding of these facts, we offer the following overview.

During runtime, computer hardware implements a fetch-decode-execute loop to run a program. The hardware maintains an IP register that contains the memory address of the next instruction to be executed. During the fetch phase, the hardware loads the data pointed to by the IP register. The data is then decoded as an instruction that is subsequently executed. The IP is then set to the next instruction to be executed. Control is hijacked by taking control of the IP, which is then used to fetch, decode, and execute the attacker's computation.

A straightforward exploit for `acpi_listen` hijacks control by overwriting data used to implement function returns. Exploits can also overwrite other control data (such as function pointers and the global offset table, as in Muller²⁹), but we omit these details here. Function calls, returns, and local variables are not supported directly by hardware. The compiler implements the semantics of these abstractions using low-level assembly instructions and memory. An attacker must be proficient in many details of code execution (such as how arguments are passed and registers are shared between caller and callee). For simplicity, we assume a standard C calling convention known as `cdecl`. Functions using it implement a stack abstraction in memory where functions push space for local variables, arguments to future calls, and other data onto the stack immediately after being called. A function return pops the allocated space off the stack. Thus, the stack grows a bit for each call and shrinks a bit on each return.

When f calls g , f first puts g 's arguments onto the stack, then invokes g , typically through a `call` assembly instruction. The semantics of `call` includes pushing f 's return address onto the stack; that is, the address in f where execution (normally) continues once g terminates. Upon entrance, g creates space for its variables and other run-time information (such as saved register values). After g completes, g returns control to f by shrinking the created stack space for g and popping off the saved address into the IP register, typically through a `ret` instruction. A critical detail is that the popped value, regardless of whether it was the original value pushed by f or not, is used as the address of the next instruction to execute. If an attacker can overwrite that address, the attacker can gain control of execution.

The stack frame just before `sprintf` is called on line 22 in the Figure. The flow of execution for creating the depicted stack includes six steps:

Return address pushed onto the stack. When `main` called `ud_connect`, `main` pushed the address of the next instruction to be executed (corresponding to line 13) onto the stack;

Control transfer. `main` transferred control to `ud_connect`;

Local variable space allocated. `ud_connect` allocated space for its local variables. On our computer, 108B were allocated for `sun_path` and an additional 28B for other data (such as additional local variables and saved register values);

Function body executed. The body of `ud_connect` ran. When `sprintf` is called, a similar flow pushes a new return address on the stack and new space onto the stack for `sprintf`'s local variables;

Local variable space deallocated. When `ud_connect` returns, it first deallocates the local variable space, then pops off the saved return address into the IP register; and

Return to caller. Under normal operation, the return address points to the instruction for line 13, and `main` resumes execution.

The crux of a control-flow hijack is that memory is used to store both control data (such as return addresses) and program-variable values, but the control data is not protected from

Governments worldwide are developing computer-warfare capabilities, and exploits have become a new type of ammunition.

being overwritten in a variable update. Control-flow hijacks are an instance of a channeling vulnerability that arise when the control and data planes are not rigorously separated. For this particular example, an out-of-bound write can clobber the return address. When `sprintf` executes, it copies data sequentially from `name` up the stack, starting from the address for `sun_path`, as shown. The copy stops only when a zero integer, or ASCII NULL, is found, which is not necessarily when `sun_path` runs out of space. A long name will clobber the saved local variables and eventually the saved return address. Since an attacker controls the values in `name`, and those values overwrite the return address, the attacker ultimately controls which instructions are executed when `ud_connect` returns.

Attackers must analyze the program to figure out exactly how many bytes to write, what constraints may be on the bytes, and what would be a good value with which to overwrite the return address. For `acpi_listen`, a string of length 140 will overwrite the return address. The first 108B will be copied into space allocated for `sun_path`. The next 28B on the stack are intended to hold local variables and saved register values. The final 4B overwrite the saved return address.

When `ud_connect` returns, the overwritten return address is popped off the stack into the IP register. The machine continues executing the instruction starting at the overwritten address. While this example overwrites the return address, a variety of other control data structures can be used to seize control; examples include function pointers, heap metadata, and C++ virtual function tables.

Control is typically hijacked to run an attacker-supplied computation. The most basic attack is to inject executable code into the vulnerable process. More advanced techniques (such as command injection, return-to-libc, and return-oriented programming) are also possible^{29,33} (and in some cases can be automated as well³¹), but we omit such discussion here.

A natural choice for the computation is to execute the command-line shell `/bin/sh` so the attacker is able to subsequently run any command

with the privileges of the exploited process. In fact, executing a shell is so popular that colloquially any attacker code is called “shellcode.” A classic approach is to give executable code as input to the program and redirect control flow to the given executable code. The executable code itself can be created by mimicking the assembly for `execve("/bin/sh", args, NULL)`. Attackers introduce the shellcode to the vulnerable program as a normal string program input that is eventually decoded and executed as code.

The final step of the attack is to overwrite the return address with the address of the shellcode. On our machine, `sun_path` is at memory address `0xbffff274`. The complete exploit for `acpi_listen` (generated automatically by our AEG tools) is shown in the figure, where:

Shellcode. The first bytes of the command line argument are the shellcode; the shellcode is 21B, and, in this case, the first 21B are copied into bytes 0–20 of `sun_path`;

Padding. The next 115B of input can be any non-zero, or non-NULL ASCII, value; the bytes are copied into bytes 21–107 of `sun_path` and the additional space for other locals; and

Shellcode address. The last 4B of input are the hex string `0x74 0xf2 0xff 0xbf`. They overwrite the return address. When the return address is popped, the bytes become the address `0xbffff274` (because x86 is little endian), which is the address of the shellcode after it is copied to `sun_path`.

The figure shows the stack frame after supplying this string as a command-line argument following `-s`. When `ud_connect` returns, the address `0xbffff274` is popped into the IP register, and the hardware fetches, decodes, and executes the bytes in `sun_path` that, when interpreted as executable code, runs `/bin/sh`. When the shellcode runs, the attacker is able to run any command with the same privileges as the exploited program.

Research Vision

Manual exploit generation requires a developer to reason about an enormous number of details (such as size of the stack, location of control flow critical data, like return address, and precise semantics of each instruction).

Our research vision is to automate it.

AEG uses verification techniques to transform the process of finding and deciding exploitability to reasoning in logic. At a high level, AEG consists of three steps: It first encodes what it means to exploit a program as a logical property; it then checks whether the exploitability property holds on a program path; and finally, for each path the property holds, it produces a satisfying input that exploits the program along the path.

These steps are the cornerstones of AEG research. First, what exploitability properties do we encode, and how? In industry, an exploit could mean control-flow hijack, while an intelligence agency might also include information disclosures, and a safety board could include denial of service for critical services. Any single property may have many encodings, with some more efficient for automated tools to check than others. Second, what techniques and algorithms should a programmer employ to check a program? The general problem of checking programs for properties is called “software model checking,”²⁴ encompassing a number of techniques (such as bounded model checking, symbolic execution, and abstract interpretation). Third, what does it take to implement real systems, and how do these systems perform on real software?

The theory of AEG can be described with a small number of operations on a well-defined programming language that interacts with its environment in a few predictable, easy-to-model ways. However, a real system must also contend with hundreds of CPU instructions and the tricky and complex ways programs interact with their environments. Sometimes even pedestrian yet necessary details are difficult to get right; for example, it took our team almost a year to stop finding bugs in our internal semantics for the x86 shift instructions (such as `shl`). The developers of Microsoft’s SAGE tool reported similar difficulties for the same instructions.⁴

Current AEG research primarily uses symbolic execution²⁵ to check program paths for exploitability properties. At a high level, symbolic execution represents all possible inputs as

a set of symbolic input variables. Symbolic execution then picks a program path through a predefined path-selection algorithm. The path is then “executed,” except, instead of executing on a real, concrete input, a symbolic input stands in for any possible concrete value. Symbolic execution builds up a path formula in terms of the symbolic inputs based on the instructions executed. The path formula is satisfied, meaning made true, by any concrete input that executes the desired path. If the path formula is unsatisfiable, there is no input that executes the path, and the path is called infeasible. The satisfiability check itself is done through automated solvers (such as Satisfiability Modulo Theories, or SMT).¹⁵ By construction, free variables correspond to program inputs, and any satisfying assignment of values to free variables (called a model) is an input that executes the selected path. SMT solvers enumerate satisfying answers when needed.

In `acpi_listen`, the symbolic inputs are the first two arguments `argv[1]` and `argv[2]`. (Although we have shown source code for `acpi_listen` for clarity, our AEG tool Mayhem requires only the program executable.¹²) Executing the `-s` option program path generates the constraint that the first 3B of `argv[1]` correspond to the NULL-terminated string `-s`. At each subsequent branch point, symbolic execution adds more constraints to the formula. Next, `acpi_listen` calls `sprintf`, which copies bytes from `name` to `addr`. `sun_path` until it encounters a NULL character. Symbolic execution captures this logic by adding the constraint that each copied byte is non-NULL. Symbolically executing the `-s` program path where `argv[1]` is three symbolic bytes and `argv[2]` is 140 non-NULL symbolic bytes generates the constraints:

$$\begin{aligned} \text{argv}[1][0:2] &= \text{"-s"} \wedge \forall i \in [0, 139]. \\ \text{argv}[2][i] &\neq 0 \wedge \text{argv}[2][140] = 0 \end{aligned} \quad (1)$$

Note that a formula may have many satisfying answers; for example, bytes 0–139 of `argv[2]` can be “A,” “B,” or any other non-NULL character.

Each feasible path can be checked for exploitability by adding a set of con-

History of AEG

Symbolic execution was invented around 1975 independently by several researchers.^{5,23,25} Around 2005, the field exploded. Hundreds of papers have now been published describing advanced techniques and applications; see Cadar and Sen¹¹ for a description and the main challenges of symbolic execution. Modern tools (such as KLEE,⁹ EXE,¹⁰ SAGE,²⁶ and others^{7,13,32,37}) find inputs that can crash or hang a system. Such inputs may well be viewed as exploits in safety-critical systems where uptime is critical. More generally, work in symbolic execution is directly applicable to making AEG more efficient. As of 2012, most symbolic-execution work followed one execution path at a time. Since then, more work has looked at generalizing over multiple paths (such as to loops³⁰). Others have also investigated alternatives to symbolic execution that tame path explosion (such as Brumley and Jager⁶ and Flanagan and Saxe^{16,18,26}). More generally, any verification technique that can produce example inputs (such as bounded model checking) is likely usable for AEG.

Modern AEG research dates to at least Ganapathy et al.,¹⁷ who explicitly connected verification to exploit generation, modeling how format string specifiers are parsed by functions like `printf` that take a variable number of arguments and use the model to automatically generate exploits. They also demonstrated automatically generating an exploit against a key integrity property for a cryptographic co-processor.¹⁷ However, they considered only API-level exploits, which do not include running shellcode or the conditions necessary to reach a vulnerable API call site.

In 2007, Medeiros²⁸ and Grenier et al.²¹ proposed techniques based on pattern matching for AEG.

In 2008, Brumley et al.⁸ developed automatic patch-based exploit generation (APEG). The APEG challenge is, given a buggy program P and a patched version P' , generate an exploit for the bug present in P but not present in P' . The idea is the difference between P and P' reflects where the original bug occurs and under what conditions it might be triggered. Attackers have long known the value of analyzing patches to find non-public bugs; for example, attackers have been known to joke Microsoft's "patch Tuesday" is followed by "exploit Wednesday." Our techniques automatically found the differences between P and P' and generated inputs that triggered the bugs in P using symbolic execution. One main security implication is that attackers can potentially use APEG to exploit bugs before patches can be distributed to a large number of users. We generated exploits for five Microsoft security patches, including

triggering an infinite loop in the TCP/IP driver and stealing files on Microsoft Web servers. One limitation was that our work on APEG only proposed, but did not implement, techniques for executing shellcode for memory-safety bugs.⁸

Heelan's 2009 thesis²² was the first to comprehensively describe and implement techniques for automatically generating control-flow hijack exploits that execute shellcode. In Heelan's problem setting, the attacker is given an input that executes an exploitable program path, and the goal is to output a working control-flow-hijack exploit. This setting is the same as in our running example where we first fuzzed to find bugs, then checked exploitability. Heelan proposed using symbolic execution and taint analysis to derive the conditions necessary to transfer control to shellcode and demonstrated a tool that produced exploits for several synthetic and for one real vulnerability. He also used a technique called return-to-register to improve exploit robustness. Heelan's thesis also presented a history of AEG work through 2009.

In 2011, we proposed AEG techniques that find bugs and generate exploits, demonstrating them on 16 vulnerabilities.¹ The initial work performed symbolic execution on source code to find bugs, then used dynamic binary analysis to generate control-flow hijack exploits. Included were a number of optimizations for searching the state space (such as preconditioned symbolic execution and the buggy-path first optimizations discussed earlier). In 2012, we introduced Mayhem, a tool and set of techniques for AEG on executable code.¹² With Mayhem, we proposed techniques for actively managing symbolically executed program paths without exhausting memory and reasoning about symbolic memory addresses efficiently. Both papers^{1,12} targeted control-flow hijacks for buffer overflows and format-string vulnerabilities. Mayhem generated exploits for seven Windows and 22 Linux vulnerabilities. Disregarding one long-running outlier, the average exploit-generation time in all experiments was 165 seconds. As of July 2013, Mayhem was able to generate exploits for buffer overflows, format strings, command injection, and some information-leak vulnerabilities.

AEG¹ and Mayhem¹² were designed to demonstrate a bug is exploitable but do not try to bypass defenses that may otherwise protect a system. In 2011, we proposed techniques for bypassing the DEP and ASLR defenses implemented in Windows 7 and Linux, as well as exploit hardening and maintenance.³¹

straints that are satisfied only by exploiting inputs. Most research tackles control-flow hijack exploits, where the exploitability constraints specify the IP register holds a value that corresponds to some function f of user input i (such as, f may be a call to `tolower` on the input i) and the resulting IP points to shellcode:

$$IP = f(i) \wedge \text{mem}[IP] = \langle \text{shellcode} \rangle \quad (2)$$

Now let a_r be the memory address for the return address and a_s be the ad-

dress of our shellcode. The full formula to reach and exploit the `acpi_listen` bug is:

$$\begin{aligned} &(\text{Equation 1}) \wedge \text{mem}[ar] = as \\ &\wedge \text{mem}[as:as+\text{len}(\text{shellcode})-1] = \\ &\langle \text{shellcode} \rangle \end{aligned} \quad (3)$$

The $\text{mem}[a_r]$ constraint requires the return address to contain the address of the shellcode a_s . The final constraint requires the shellcode to start at address a_s . The variable a_s is left unconstrained since the shellcode could potentially

be placed anywhere in memory. In our experiment, our AEG tool Mayhem¹² found the exploitable path and solved the exploitability formula in 0.5 seconds. Mayhem is also able to enumerate satisfying answers to automatically generate multiple exploits.

Managing state explosion. AEG is a type of software verification, albeit for a very special property. As such, it inherits benefits but also well-known scalability challenges (such as path explosion and the NP-hardness of solving SMT queries in general). They are often

amplified in AEG because AEG techniques reason about both low-level code and large inputs, along with a few abstractions. However, specific characteristics of AEG also afford researchers unique opportunities.

Consider the affect of path prioritization on this program:

```
int x = get_int();
if((x % 2) == 0) {
    if(x > 10) vuln1();
    else if(x == 3) vuln2();
    else safe();
} else { safe(); }
```

Let x_o be ao explore the program and find the vulnerability:

$$\begin{aligned} (x_o \% 2) = 0 \wedge \neg(x_o > 10) \wedge \neg(x_o = 3) \\ (x_o \% 2) = 0 \wedge \neg(x_o > 10) \wedge x_o = 3 \\ \neg((x_o \% 2) = 0) \\ (x_o \% 2) = 0 \wedge x_o > 10 \end{aligned}$$

The first formula for the first path is satisfiable (such as when $x_o = 4$), indicating the path can be executed but is safe (unexploitable). The second formula corresponds to the infeasible path up to `vuln2()` and is unsatisfiable because the constraint $(x_o \% 2) = 0$ and $x_o = 3$ cannot both be true simultaneously. Since `vuln2` will never be executed, it can never be exploited. The third formula corresponds to a feasible, safe path. Only the fourth formula corresponding to the path up to `vuln1()` is satisfiable, where a satisfying assignment (such as $x_o = 42$) corresponds to an exploit. In general, the number of paths and formulas is infinite for programs with loops and exponential in terms of number of branches for any acyclic portion, making effective path selection a fundamental issue in AEG research.

Path-selection heuristics guide execution so vulnerable paths are selected early in exploration. Symbolic execution research is filled with a variety of approaches. For example, KLEE has options for depth-first traversal of the control-flow graph, as well as a randomized strategy.⁹ Microsoft uses generational search,²⁰ which prioritizes symbolically executing program paths that branch off a known path taken by a fixed concrete seed input. Godefroid et al.'s research²⁰ suggests generational search is more effective



A sound AEG technique says a bug is exploitable if it really is exploitable, while a complete technique reports all exploitable bugs.



than either breadth-first search or depth-first search.

Two techniques that proved effective in our experiments at Carnegie Mellon are “preconditioned symbolic execution” and “buggy-path first.”¹ Preconditioned symbolic execution first performs lightweight analysis to determine the necessary conditions to exploit any lurking bugs, then prunes the search space of paths that do not meet these conditions. For example, a lightweight program analysis may determine the minimum length input string needed to trigger possible buffer overflows, and paths corresponding to inputs smaller than the minimum length can be pruned or skipped.

The idea of buggy-path first is that any bug is a sign of programmer confusion, increasing the likelihood of an exploitable bug being nearby. For example:

```
char buf[1024];
memset(buf, 0, strlen(input));
...
strncpy(buf, input,
        strlen(input));
```

The second line contains a mistake where potentially more than 1,024B of `buf` are zeroed. This bug would likely not lead to a control-flow hijack, but does signal confusion that the length of input is somehow related to the size of `buf`. Buggy-path first would prioritize further exploration of the buggy path over other possible paths and thus discover the subsequent exploitable code more quickly in our tests. Note that a unique aspect of buggy-path first is that execution continues under the assumption the bug has been triggered (such as in the example when nearby stack variables may have been zeroed inadvertently).

A second core challenge of AEG research is optimizing SMT satisfiability checks. In theory, each satisfiability check is an NP-hard problem instance, but in practice many queries are resolved quickly. For example, in an experiment involving 5.6 million SMT queries, 99.98% of all solved queries took one second or less. Domain-specific optimizations in symbolic execution (such as arithmetic and logical simplifications, strength reduction,

concrete execution, and caching) all help speed queries.^{9,10,13,19,32}

In 2006 when we started using symbolic execution and SMT solvers, we treated the SMT solver as a black box, focusing only on the symbolic executor. In hindsight, that approach was naive. In our research group we now believe it is more fruitful to view the SMT solver as a search procedure and use optimizations to guide the search. For example, one recurring challenge in AEG is checking satisfiability of formulas that operate on memory with symbolic memory addresses. A symbolic memory address occurs when an index into an array or memory is based on user input, as in:

```
...; y=mem[i%256]; if(y==2) vuln(); ...
```

Without more information, the SMT solver must do a case split over all possible values of i that may reach downstream statements (such as `vuln`). Case splits can quickly push an SMT solver off an exponential cliff. Symbolic memory references often crop up in commonly occurring library calls (such as conversion functions like `tolower` and `toascii`) and parsing functions (such as `sscanf`). Many symbolic executors mitigate the case split by concretizing symbolic addresses to a specific value (such as by picking $i=42$).

Unfortunately, in our experiments with dozens of exploitable bugs we found concretization overconstrains formulas, leading our initial AEG techniques to miss 40% of known exploitable bugs in our test suite;¹² for example, AEG may need to craft an input that becomes valid shellcode after being processed by `tolower` (such as `tolower` is f in Equation 2). In Mayhem, we proposed a number of optimizations for symbolic memory;¹² for example, one performs a type of strength reduction where symbolic memory accesses are encoded as piecewise linear equations.¹²

Example application: Exploiting /usr/bin. Recall we fuzzed Debian /usr/bin and found 52 distinct bugs in 29 programs, including `acpi_listen`. One goal was to determine which bugs are exploitable.

We ran our binary-only AEG tool called Mayhem¹² on each crash to de-

Automatically generating an exploit provides proof that the reported bug is security-critical.

termine if we could automatically generate an exploit from the crashing path. We also manually checked whether it was possible to exploit the bug. Five of the 52 bugs were vulnerable to a control-flow hijack, and Mayhem generated exploits for four of them. The exploit for `acpi_listen` took 0.5 seconds to generate, and the remaining three took 8, 12, and 28 seconds, respectively.

These results on /usr/bin offer three insights: First, current AEG tools like Mayhem are sound but incomplete. A sound AEG technique says a bug is exploitable if it really is exploitable, while a complete technique reports all exploitable bugs. Unfortunately, Rice's theorem implies developing a sound and complete analysis for any nontrivial program property is in general undecidable. Second, AEG can be very fast when it succeeds. And finally, there is ample room for improving AEG in particular and symbolic execution and software model checking in general. For example, we analyzed why Mayhem failed on the last vulnerability, finding the problem was a single constraint that pushed the SMT solver (we use Z3) off an exponential cliff. Perhaps comically, manual analysis showed the constraint was superfluous but was not recognized as such by the automatic formula optimizer. Once the constraint was removed from the formula, exploit generation took less than five seconds.

Real-World Considerations

Security practitioners often focus only on exploits for programs on the attack surface of a system.²⁷ The attack surface consists roughly of the set of programs, files, protocols, services, and other channels available to an attacker; examples include network daemons, programs called from Web servers on untrusted inputs, privileged programs, and media players. Our example `acpi_listen` is not on the attack surface. We chose `acpi_listen` because it highlights the steps of AEG, yet disclosing the exploit would do little damage because it is not on the attack surface. Interestingly, the `acpi_listen` vulnerability is remarkably similar to a recent PHP vulnerability that performs an unchecked copy on the same data structure.¹⁴

Overall, AEG techniques are valuable because they show whether a pro-

gram can be exploited regardless of whether it is on the attack surface or not. For example, a program not on the attack surface in one deployment may be on the surface for another. More generally, programs on the attack surface are simply a subset of all programs; if we can handle all programs we can surely handle the subset on the attack surface. Current techniques have found exploits on the attack surface, albeit not in widely used large applications like Internet Explorer. For example, as we wrote this article we ran Mayhem on additional examples that are on the attack surface, finding a number of zero-day exploits for media applications (such as `ezstream` and `imview`) and network applications (such as `latd` and `ndtpd`).

Another consideration is additional layers of defense that might protect otherwise exploitable programs. Two popular operating-system-level defenses against control-flow hijacks are data-execution prevention, or DEP, and address space layout randomization, or ASLR.

DEP marks memory pages either “writable” or “executable” but forbids a memory page from being both. DEP prevents an exploit that requires writing and then executing shellcode on a memory page from working (such as the shellcode mentioned earlier). Unfortunately, attackers have developed techniques to bypass DEP. One such method is called return-to-libc where the attacker shellcode executes code already present in memory (such as by running `system("/bin/sh")` in `libc` directly) rather than writing new code to memory. Return-oriented programming, or ROP, uses instruction sequences already present in memory, called “gadgets.” Shacham et al.³³ showed it is possible to find a Turing-complete set of gadgets in `libc`.

ASLR prevents control-flow hijacks by randomizing the location of objects in memory. Recall that to exploit `acpi_listen`, the attacker needs to know the address of the shellcode. ASLR randomizes addresses so vulnerable programs likely crash instead of successfully redirecting control to the shellcode. ASLR is an important defense but does not fix the underlying vulnerabilities and thus may provide limited protection; for example,

Windows and Linux systems running on 32b processors may have insufficient randomness to provide strong security,³⁴ though 64b architectures can address this problem. Particular deployments of ASLR may have weaknesses as well; for example as of January 2013 the program image of Linux executables is often not randomized. Even when randomized well, additional vulnerabilities may disclose information that can subsequently be used in a control-hijack exploit.

Schwartz et al.³¹ proposed exploit hardening, which takes an exploit that works against an undefended system and hardens it to bypass defenses. One step in exploit hardening is to automatically generate ROP payloads (to bypass DEP) that take advantage of small portions of unrandomized memory (to bypass ASLR on the 2013 implementations of ASLR on Windows 7 and Linux). In particular, Schwartz et al. showed ROP payloads can be generated for most programs in Windows and Linux that have at least 20KB of unrandomized code, which is true for many programs. Exploit hardening can be paired with AEG to check the end-to-end security of a program running on a specific system.

Finally, DEP and ASLR defend only against memory overwrite attacks. Other vulnerabilities (such as information disclosure, denial of service, and command injection) are also critical in practice; for example, DEP and ASLR do not protect against exploits for the command-injection vulnerability found by Mayhem in `ezstream`.

Conclusion

AEG is far from being solved. Scalability will always be an open and interesting problem. As of February 2013, AEG tools typically scale to finding buffer overflow exploits in programs the size of common Linux utilities. In Mayhem, one current bottleneck is driving the symbolic executor to the buggy portion of the state space. As a result, programs with deep bugs are typically beyond the scope of our current Mayhem AEG tool. Examples include large programs with bugs deep in the program (such as Internet Explorer and Adobe Reader), as well as those with large protocol state (such as first authenticate, then send mul-

tiply fragmented messages to exploit a bug). In addition, programs with complex functions (such as hashes) are often a bottleneck for SMT solvers. One promising data point is that Microsoft’s SAGE tool routinely finds bugs in large applications,²⁰ though automatically generating exploits for those bugs is an open challenge with huge potential rewards.

More fundamentally, AEG must expand to involve a wider variety of exploitability properties and scale to new program domains. While buffer overflows continue to be exploited^{3,35} integer overflows, use-after-free, heap overflows, and Web vulnerabilities are also important (and popular) targets;³ for example, heap overflows against modern operating systems like Windows 8 pose difficult challenges (such as modeling internal heap metadata and new heap allocators with built-in defenses). In our experience, the problem is often not coming up with some formalism, but with the right formalism and optimizations that make AEG efficient and practical on real-world programs and vulnerabilities.

Except for a few examples (such as Ganapathy et al.’s¹⁷ exploits against a particular cryptographic API), most work in AEG has focused on exploiting programs in type-unsafe languages, though type safety is no panacea. Information flow, command injection, and many other common exploitable bugs can all occur in typical type-safe languages. Moreover, the runtime environment itself may have security-critical flaws. For example, the most commonly exploited vulnerabilities in 2011 were in Java.²

AEG can be modeled as a verification task; therefore, the better programmers and researchers get at software verification, the better they will likely get at automatically generating exploits. Some security researchers are pessimistic about the practicality of AEG in many application settings,³⁶ rightfully pointing out significant scalability hurdles and the lack of exploits against vulnerabilities like use-after-free. We are more optimistic. Eight years ago, AEG techniques were restricted to analyzing a single API call. Today, AEG can both automatically find and generate exploits in common binaries. In an effort to improve

security in Debian, we started a project in 2013 to check all programs in `/usr/bin` for exploitable bugs and so far have found more than 13,000 with more than 150 exploitable. Advancements will continue to be fueled by better tools, techniques, and improvements in verification and security.

Acknowledgments

This research is partially supported by grants and support from the National Science Foundation, the Defense Advanced Research Projects Agency, Lockheed Martin, Northrop Grumman, and the Prabhu and Poonam Goel Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies. 

References

- Avgerinos, T., Cha, S.K., Lim, B.T.H., and Brumley, D. AEG: Automatic Exploit Generation. In *Proceedings of the Network and Distributed System Security Symposium* (San Diego, CA, Feb. 6–9). Internet Society, Reston, VA, 2011, 283–300.
- Faulchender, D., Bawany, S., Blackbird, J., Blakemore, E., Fauthhaber, J., Fayyaz, S., Felstead, D., Henry, P., Goel, N.K., Jones, J., Kuo, J., Lauricella, M., Malcolmson, K., Ng, N., Oram, M., Pecceij, D., Probert, D., Rains, T., Simorjay, F., Stewart, H., Thomlinson, M., Wu, S., and Zink, T. *Microsoft Security Intelligence Report 12* (July–Dec. 2011). Microsoft, Redmond, WA; <http://www.microsoft.com/security/sir/archive/default.aspx>
- Bilge, L. and Dumitras, T. Before we knew it: An empirical study of zero-day attacks in the real world. In *Proceedings of the ACM Conference on Computer and Communications Security* (Raleigh, NC, Oct. 16–18). ACM Press, New York, 2012, 833–844.
- Bounimova, E., Godefroid, P., and Molnar, D. *Billions and Billions of Constraints: Whitebox Fuzz Testing in Production*. Technical Report MSR-TR-2012-55. Microsoft, Redmond, WA, May 2012; <http://research.microsoft.com/apps/pubs/?id=165861>
- Boyer, R. S., Elspas, B., and Levitt, K. N. SELECT—A formal system for testing and debugging programs by symbolic execution. In *Proceedings of the International Conference on Reliable Software* (Los Angeles, Apr). ACM Press, New York, 1975, 234–245.
- Brumley, D. and Jager, I. *Efficient Directionless Weakest Preconditions*. Technical Report CMU-CyLab-10-002. Carnegie Mellon University, Pittsburgh, PA, July 14, 2010; https://www.cylab.cmu.edu/research/techreports/2010/tr_cylab10002.html
- Brumley, D., Jager, I., Avgerinos, T., and Schwartz, E.J. BAP: A binary analysis platform. In *Proceedings of the International Conference on Computer Aided Verification* (Snowbird, UT, July 14–20). Springer, Berlin, Heidelberg, Germany, 2011, 463–469.
- Brumley, D., Poosankam, P., Song, D., and Zheng, J. Automatic patch-based exploit generation is possible: Techniques and implications. In *Proceedings of the IEEE Symposium on Security and Privacy* (San Francisco, May 18–21). IEEE Press, Los Alamitos, CA, 2008, 143–157.
- Cadar, C., Dunbar, D., and Engler, D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the USENIX Symposium on Operating System Design and Implementation* (San Diego, CA, Dec. 8–10). USENIX Association, Berkeley, CA, 2008, 209–224.
- Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., and Engler, D.R. EXE: Automatically generating inputs of death. In *Proceedings of the ACM Conference on Computer and Communications Security* (Alexandria, VA, Oct. 30–Nov. 3). ACM Press, New York, 2006, 322–335.
- Cadar, C. and Sen, K. Symbolic execution for software testing: Three decades later. *Commun. ACM* 56, 2 (Feb 2013), 82–90.
- Cha, S.K., Avgerinos, T., Rebert, A., and Brumley, D. Unleashing Mayhem on binary code. In *Proceedings of the IEEE Symposium on Security and Privacy* (San Francisco, May 21–23). IEEE Press, Los Alamitos, CA, 2012, 380–394.
- Chipounov, V., Kuznetsov, V., and Candea, G. The S2E platform. *ACM Transactions on Computer Systems* 30, 1 (Feb. 2012).
- CERT/NIST. *PHP socket_connect() Stack Buffer Overflow. National Vulnerability Database*. Entry CVE-2011-1938. National Institute of Standards and Technology, Gaithersburg, MD, May 31, 2011; <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-1938>
- De Moura, L. and Björner, N. Satisfiability Modulo Theories: Introduction and applications. *Commun. ACM* 54, 9 (Sept. 2011), 69–77.
- Flanagan, C. and Saxe, J.B. Avoiding exponential explosion: Generating compact verification conditions. In *Proceedings of the ACM Symposium on Principles of Programming Languages* (London, U.K., Jan. 17–19). ACM Press, New York, 2001, 193–205.
- Ganapathy, V., Seshia, S.A., Jha, S., Rebs, T.W., and Bryant, R.E. Automatic discovery of API-level exploits. In *Proceedings of the International Conference on Software Engineering* (St. Louis, MO, May 15–21). IEEE Press, Los Alamitos, CA, 2005, 312–321.
- Godefroid, P. Compositional dynamic test generation. In *Proceedings of the ACM Symposium on the Principles of Programming Languages* (Nice, France, Jan. 17–19). ACM Press, New York, 2007, 47–54.
- Godefroid, P., Klarlund, N., and Sen, K. DART: Directed automated random testing. In *Proceedings of the ACM Conference on Programming Language Design and Implementation* (Chicago, June 12–15). ACM Press, New York, 2005, 213–223.
- Godefroid, P., Levin, M.Y., and Molnar, D. SAGE: Whitebox fuzzing for security. *Commun. ACM* 55, 3 (Mar. 2012), 40–44.
- Grenier, L. (Pusscat and LinOxx). Byakugan: Automating exploitation. In *ToorCon Seattle* (Seattle, WA, May 2007); <http://seattle.toorcon.net/>
- Heelan, S. *Automatic Generation of Control Flow Hijacking Exploits for Software Vulnerabilities*. M.Sc. thesis. University of Oxford, Oxford, U.K., Sept. 3, 2009; http://solo.bodleian.ox.ac.uk/primo_library/libweb/action/dlDisplay.do?vid=OXVU1&docId=oxfale-ph017069721
- Howden, W.E. Methodology for the generation of program test data. *IEEE Transactions on Computers* C-24, 5 (May 1975), 554–560.
- Jhala, R. and Majumdar, R. Software model checking. *ACM Computing Surveys* 41, 4 (Oct. 2009).
- King, J.C. Symbolic execution and program testing. *Commun. ACM* 19, 7 (July 1976), 385–394.
- Kuznetsov, V., Kinder, J., Bucur, S., and Candea, G. Efficient state merging in symbolic execution. In *Proceedings of the ACM Conference on Programming Language Design and Implementation* (Beijing, June 11–16). ACM Press, New York, 2012, 193–204.
- Manadhata, P.K. and Wing, J.M. An attack surface metric. *IEEE Transactions on Software Engineering* 37, 3 (May–June). IEEE Press, Los Alamitos, CA, 2011, 371–386.
- Medeiros, J. *Automated Exploit Development, The Future of Exploitation Is Here*. Technical Report. Grayscale Research, 2007; http://www.grayscale-research.org/new/pdfs/toorcon_whitepaper.pdf
- Muller, T. *ASLR Smack & Laugh Reference Seminar on Advanced Exploitation Techniques*. Technical Report. RWTH Aachen University, Aachen, Germany, Feb. 2008.
- Saxena, P., Poosankam, P., McCamant, S., and Song, D. Loop-extended symbolic execution on binary programs. In *Proceedings of the International Symposium on Software Testing and Analysis* (Chicago, July 19–23). ACM Press, New York, 2009, 225–236.
- Schwartz, E.J., Avgerinos, T., and Brumley, D.Q. Exploit hardening made easy. In *Proceedings of the USENIX Security Symposium* (San Francisco, Aug. 8–12). USENIX Association, Berkeley, CA, 2011, 379–394.
- Sen, K., Marinov, D., and Agha, G. CUTE: A Concolic Unit Testing Engine for C. In *Proceedings of the ACM International Symposium on Foundations of Software Engineering* (St. Petersburg, Russia, Aug. 18–26). ACM Press, New York, 2005, 263–272.
- Shacham, H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the ACM Conference on Computer and Communications Security* (Alexandria, VA, Oct. 29–Nov. 2). ACM Press, New York, 2007, 552–561.
- Shacham, H., Page, M., Pfaff, B., Goh, E., Modadugu, N., and Boneh, D. On the effectiveness of address-space randomization. In *Proceedings of the ACM Conference on Computer and Communications Security* (Washington, D.C., Oct. 25–29). ACM Press, New York, 2004, 298–307.
- van der Veen, V., dutt-Sharma, N., Cavallaro, L., and Bos, H. Memory errors: The past, the present, and the future. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses* (Amsterdam, The Netherlands, Sept. 12–14). Springer, Berlin, Heidelberg, Germany, 2012, 86–106.
- Vanegue, J., Heelan, S., and Rolles, R. SMT solvers for software security. In *Proceedings of the USENIX Workshop on Offensive Technologies* (Bellevue, WA, Aug. 6–7). USENIX Association, Berkeley, CA, 2012.
- Wang, X., Chen, H., Jia, Z., Zeldovich, N., and Kaashoek, M.F. Improving integer security for systems with KINT. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, Oct. 8–10). USENIX Association, Berkeley, CA, 2012, 163–177.

Thanassis Avgerinos (thanassis@cmu.edu) is a Ph.D. candidate in the Electrical and Computer Engineering Department at Carnegie Mellon University, Pittsburgh, PA, and a founder of ForAllSecure.com.

Sang Kil Cha (sangkilc@cmu.edu) is a Ph.D. candidate in the Electrical and Computer Engineering Department at Carnegie Mellon University, Pittsburgh, PA.

Alexandre Rebert (alexandre@cmu.edu) is a Ph.D. student in the Electrical and Computer Engineering Department at Carnegie Mellon University, Pittsburgh, PA, and a founder of ForAllSecure.com.

Edward J. Schwartz (edmcman@cmu.edu) is a Ph.D. candidate in the Electrical and Computer Engineering Department at Carnegie Mellon University, Pittsburgh, PA.

Maverick Woo (pooh@cmu.edu) is a systems scientist in CyLab at Carnegie Mellon University, Pittsburgh, PA.

David Brumley (dbrumley@cmu.edu) is an assistant professor in the Electrical and Computer Engineering Department at Carnegie Mellon University, Pittsburgh, PA, and CEO of ForAllSecure.com.

A solution to the persistent problem of preventing collusion in Vickrey auctions.

BY SILVIO MICALI AND MICHAEL O. RABIN

Cryptography Miracles, Secure Auctions, Matching Problem Verification

IN THIS ARTICLE, we extend the methods of Rabin et al.^{10,11} in a major way and provide a solution to the long-standing important problem of preventing collusion in second-price (Vickrey) auctions. The new tools presented are deniable revelation of a secret value and uncontrollable deniable bidding. In Rabin et al.,^{10,11} new highly efficient

methods for proving correctness of announced results of computations were introduced. These proofs completely conceal input values and inter-

mediate results of the computation. One application was to enable an Auctioneer to announce outcome of a sealed bid auction and provide verification of correctness of the outcome, while keeping bid values information-theoretically secret. We quickly survey these methods for completeness of the discussion and because of their wide applicability. Another example of an application is to prove to participants of a stable matching process such as the assignment residents to hospitals, of the correctness of the announced assignment without revealing any preferences of residents with respect to hospitals and vice-versa.

» key insights

- **Practically efficient secrecy of values preserving proofs of correctness of computations are useful for many financial and social processes.**
- **In particular, they supply a solution to the long-standing open problem of countering collusion of bidders in second-price (Vickrey) auctions.**
- **An important feature of these new methods is their understandability by a wide audience of potential users.**

By way of motivation, let us outline the main application given in this article for the extended method for secrecy preserving proofs of correctness. We consider Vickrey auctions where bidders $B_1, \dots, B_n$ submit sealed bids $b_1, \dots, b_n$ for a single item IT to a seller/auctioneer AU. At an announced end of bidding time T_1 , the AU opens the bids and determines that, say, b_w was the highest bid value and b_s was the second highest bid. Bidder B_w will get the item IT and pay to AU the second-highest bid value b_s .

This bidding mechanism, absent collusion, makes it worth while for every bidder to bid his true value for the item IT. It thus assures the AU a return of the second highest private true value for the IT.¹⁴

When setting up the auction, AU specifies a reserve price r . If none of the bids is $\geq r$, then the IT is not sold. If the second price is smaller than r , then the winner (if there is one) pays r for the IT.

The possibility of collusion completely subverts the above advantage to the AU from the second price auction. Assume that all bidders form a Cartel to collude against AU. They determine ahead of closing time T that B_1 's true value b_1 (as claimed by him) for the item IT is the largest among all true values as claimed by bidders. They agree that in the actual auction, B_1 will bid b_1 and each of the other bidders $B_2, \dots, B_n$ will bid r . They also agree that if B_1 gets the IT, then he will make certain side payments to Cartel members $B_2, \dots, B_n$. They also specify fines to be paid by cartel members who deviate from the agreement. Now, if all Cartel members keep to their agreement, then B_1 will get the IT and pay r to the AU. Thus, all of the seller's potential gain from conducting the auction is wiped out. Because of possibility of collusion, second-price auctions are rarely used despite their theoretical advantage.^{5, 6, 12, 13}

We shall show how the use of cryptography enables prevention of collusion in *one-time* second-price auctions by making cartel agreements unenforceable and making it worthwhile for colluders to break those agreements. In repeated auctions involving the same bidders, the participants have an incentive to voluntarily keep collusion agreements so as to gain in the long run. The extent to which our methods can be

applied to these cases and to other auctions is under study.

Using the methods of Rabin et al.^{10,11} and the new tools of *deniable revelation of a secret value and uncontrollable deniable bidding*, we design an auction mechanism with the following properties.

1. Bidders submit sealed bids $b_1, \dots, b_n$ to AU in an uncontrollable and deniable manner. This means that a bidder cannot be compelled by anybody to submit a specified bid value. Also, he cannot be compelled to reveal any information about his submitted bid.

2. The AU assigns to every bidder B_i a secret identifier id_i . Identifiers are known to AU but NOT known to bidders.

3. After the closing time of the auction, the AU determines that bidder B_w is the highest/winning bidder and that B_s is the second highest bidder with bid value b_s .

4. AU proves to the bidders, referring only to identifiers, that the bid by the bidder with identifier value id_w (say identifier value 10325) is the highest bid. Also that the bid by the bidder with identifier value id_s (say identifier value 21131) is the second highest bid.

5. The proof in 4 is information-theoretic hiding with respect to all bid values and with respect to the correlation between identifiers and bidders. Thus at this stage, bidders know nothing about who bid what and even the winner and second highest bidder do not know about their status as such.

6. The AU proves to B_w that his identifier is the above-mentioned id_w , that is, that he is the winner of the IT. AU proves to B_w that the bid value associated with the above-mentioned identifier id_s is b_s . AU collects from the winner B_w the price b_s . That is, the winner gets IT and pays to the AU the second highest bid value b_s (Vickrey). These proofs to B_w are again secrecy preserving with respect to the actual identity of B_s and any other bid value except b_s .

7. The AU proves to B_s that his identifier is id_s . The AU proves to every bidder $B_j, j \neq s$, that his identifier is different from id_s .

8. The proofs of 6–7 are again secrecy preserving and deniable by the bidders involved.

9. Every bidder B_i , if he so desires, can arrange that if he wins the item

IT, then the fact he won will remain secret/unknown to the other bidders. This assumption holds, for example, for digital goods but may be difficult to implement for some physical goods such as radio spectrum. This issue is fully discussed later.

We shall prove that these properties 1–9 enable the Auctioneer to prevent collusion by promising, when announcing the auction, a kickback payment to the second highest bidder, whoever he may turn out to be.

The implementation of properties 1–8 requires of the Auctioneer proofs of correctness of announced results of computations while keeping input values and intermediate results secret. A new highly efficient tool for doing this was presented in Rabin et al.^{10,11}

A new construct of a deniable proof of value presented in this paper is employed in implementing the properties 1 and 8.

Sealed Bid Auctions Implementation by Encryption and Secure Bulletin Board

In this article, we assume that the Auctioneer employs an electronic Secure Bulletin Board (SBB) with the following properties. The SBB is controllable by the AU who can post data. Posted data is time stamped and signed by the AU. Data cannot be erased. The SBB is viewable by all participants in the auction and they are assured that they all view the same content. Detailed implementations of a SBB use standard algorithmic tools and are not discussed herein.

Much of the data posted on the SBB will be in “sealed envelopes” created by bidders or by the AU. In Definition 3, we specify the Pedersen Commitment function which will be used in detailed proofs of the secrecy properties of our bidding mechanism. In practice, we implement sealed envelopes and commitments by an encryption function $E(\cdot)$, say a 128-bit AES (Advanced Encryption Standard) used in authenticated encryption mode such as GCM.

Previous Results and Background

The method of value-secrecy preserving proofs of correctness in Rabin et al.^{10,11} and in the present work was motivated by the ground-breaking methodology of Zero Knowledge Proofs (ZKP) innovated

in Goldreich et al.³ and Goldwasser et al.,⁴ and the subject of thousands of subsequent papers. ZKP and other methods of verification of truth of claimed statements are, however, not sufficiently efficient for providing practical solutions for the auction-verification problems treated in Rabin et al.^{10,11} and herein.

By way of example, in Parkes et al.⁸ a method using Paillier homomorphic encryption⁷ was employed for verification of claimed results of an auction while keeping bid values secret. Verification of a hundred-bidder second-price auction required several hundred minutes. By comparison, the new method of Rabin et al.¹⁰ verifies a hundred-bidder second-price auction in two milliseconds. The use of multi-party computations (see Ben-David et al.¹) provides secrecy of bids but no verification of correctness of announced results. It is also by far slower than that of Rabin et al.^{10,11} and that presented in the present work.

The main innovation of Rabin et al.^{10,11} is to work directly with the input values to a computation and its intermediate results as numbers rather than going down to the bit level. Furthermore, numbers are randomly represented by two-coordinate vectors.

The papers by Rabin et al.^{10,11} consider a generalized form of straight line computations on elements of a finite field F_p . For our applications, a 128-bit prime p is completely adequate. Thus, the field operations $(x + y) \bmod p$ and $(x \times y) \bmod p$ are rapidly executable on an ordinary 64-bit processor.

A number of players $P_1, \dots, P_n$ secretly submit to an *Evaluator Prover* EP input values $x_1, \dots, x_n$ taken from F_p (i.e., $x \in \{0, 1, \dots, p-1\}$). The EP performs a computation on these inputs and announces the results of that computation.

DEFINITION 1. *A Generalized Straight Line Computation (GSLC) on inputs $x_1, \dots, x_n \in F_p$ with K outputs $x_{L+1}, \dots, x_{L+K}$ is a sequence*

$$\text{GSLC} = x_1, \dots, x_n, x_{n+1}, \dots, x_L, x_{L+1}, \dots, x_{L+K} \quad (1)$$

where for all $m > n$ there exist $i, j < m, L$, such that $x_m = (x_i + x_j) \bmod p$, or $x_m = (x_i \times x_j) \bmod p$, or $x_m = x_i$, or $x_m = \text{TruthValue}(x_i \leq x_j)$.

An example of a GSLC for the output

We show how the use of cryptography enables prevention of collusion in one-time second-price auctions by making cartel agreements unenforceable and making it worthwhile for colluders to break those agreements.

$$x_{(2n-1)} = x_1 + \dots + x_n \text{ is}$$

$$\begin{aligned} & x_1, \dots, x_n, x_{(n+1)}, \dots, x_{(2n-1)}, \\ & \text{where } x_{(n+1)} = x_1 + x_2, x_{(n+2)} \\ & \quad = x_{(n+1)} + x_3, \text{ etc.} \end{aligned} \quad (2)$$

Random vector representations of values $x \in F_p$. We now come to the main construct for enabling Secrecy Preserving Proofs for the correctness of the results $x_{L+1}, \dots, x_{L+K}$ of the GSLC(1).

DEFINITION 2. *Let $x \in F_p$ be a value. A random vector representation $\text{RR}(x)$ of x is a vector $X = (u, v)$ where $u, v \in F_p$; u was chosen randomly (notation $u \leftarrow F_p$) and $x = (u + v) \bmod p$. For a vector $X = (u, v)$ we denote $\text{val}(X) = (u + v) \bmod p$.*

The method for creating a $\text{RR}(x) = (u, v)$ of x is to randomly choose $u \leftarrow F_p$ and set $v = (x - u) \bmod p$. Note that from u (or v) by itself, no information about x can be deduced.

Commitment functions. We shall use the Pedersen commitment function⁹ for values $u \in F_p$. Let G be a group of prime order $q > p$ for which computing the discrete log function is intractable. Let g, h in G be two generators such that $\log_g(h) = e$ (i.e., $g^e = h$) is not known and by the intractability assumption not computable in, say, a thousand years.

DEFINITION 3. *Let $u \in F_p$, the commitment $\text{COM}(u, r)$ to u , using the help value $r \in [0, q-1]$, is $\text{COM}(u, r) = g^u \times h^r$.*

Note that under a random choice of the help value r , $\text{COM}(u, r)$ is a random element of G . Consequently, the commitment function $\text{COM}(u, r)$ is *information-theoretically hiding*. Since computation of $\log_g(h) = e$ is intractable, the commitment function is *computationally binding*. The latter means that for no commitment value C is it possible to compute two different pairs $(u, r) \neq (v, s)$ such that $C = \text{COM}(u, r) = \text{COM}(v, s)$. The reason is that $\log_g(h) = e$ is efficiently computable from the equation $g^u \times h^r = g^v \times h^s$. Consequently, a player who has created and posted a commitment $\text{COM}(u, r)$ can open it only in one way to reveal the original value u .

Even the above strong binding property of the Pedersen commitment leaves it exposed to an attack by imitation. Assume that one bidder in an auction

has committed to his bid using a value u committed to as $C = \text{COM}(u, r) = g^u \times h^r$. Another bidder who sees the posted C will post $D = C \times g \times h^s$. When the first bidder decommits the value u by revealing u and r , the second bidder will open D by revealing $u + 1$ and $r + s$, thus decommitting the value $u + 1$ and raising the bid by 1. In the following, such an attack will be enabled if there is collusion between the auctioneer and the second bidder.

To counter exposure to imitation, we assume that an independent agent, such as NIST, has created and signed randomly chosen pairs (g_i, h_i) , $i = 0, 1, \dots$, of generators of the group G . When setting up the auction, the AU and every participant are assigned a different pair of generators from the above list to be used for their commitments.

Proving claimed correctness of an addition $x + y = z$. We can now show how the EP can prove to a Verifier correctness of an equation $x + y = z$. The EP prepares random representations $X = (u_1, v_1)$, $Y = (u_2, v_2)$, and $Z = (u_3, v_3)$, of the values x, y , and z . Note that

$$\text{val}(X) + \text{val}(Y) = \text{val}(Z) \quad (3)$$

if and only if there exists a $w \in F_p$ such that $X + Y = Z + (w, -w)$. The EP prepares commitments

$$\begin{aligned} \text{COM}(X) &= [\text{COM}(u_1, r_1), \text{COM}(v_1, s_1)], \\ \text{COM}(Y) &= [\text{COM}(u_2, r_2), \text{COM}(v_2, s_2)], \\ \text{COM}(Z) &= [\text{COM}(u_3, r_3), \text{COM}(v_3, s_3)] \end{aligned} \quad (4)$$

The EP posts the commitments (4) or sends them to the Verifier VER and claims that the hidden vectors X, Y, Z satisfy (3).

When challenged by VER to prove this claim, the EP posts or sends to Verifier the above value w . The Verifier now presents to EP a randomly chosen challenge $c \leftarrow \{1, 2\}$.

Assume that $c = 1$. The EP decommits/reveals to Verifier $u_j, r_j, j = 1, 2, 3$. The Verifier checks the commitments, that is, computes $\text{COM}(u_j, r_j), j = 1, 2, 3$, and compares to the posted first coordinates of $\text{COM}(X), \text{COM}(Y), \text{COM}(Z)$.

The Verifier next checks that $u_1 + u_2 = u_3 + w$. If $c = 2$ was chosen, then the Verifier asks for the second coordinates of X, Y, Z , and checks that $u_1 + u_2 =$

$u_3 - w$. The following two theorems are immediately obvious.

THEOREM 1. *If (3) is not true for the vectors committed in $\text{COM}(X), \text{COM}(Y), \text{COM}(Z)$, then Verifier will accept with probability at most $1/2$ the claim that (3) holds.*

PROOF. Under our assumption about the COM function being computationally binding, the EP can open the commitments for $u_j, v_j, j = 1, 2, 3$, in only one way. Now, if (3) does not hold, then at least one of the equations $u_1 + u_2 = u_3 + w$, or $v_1 + v_2 = v_3 - w$ is not true. So the probability that a random challenge $c \leftarrow \{1, 2\}$ will not uncover the falsity of the claim (3) is less than $1/2$.

THEOREM 2. *The above interactive proof between EP and Verifier reveals nothing about the values $\text{val}(X), \text{val}(Y), \text{val}(Z)$ beyond, if successful, that the claim that (3) is true (subject to probability at most $1/2$ of Verifier accepting a false claim).*

PROOF. We note that the interactive proof involves only the revelation of either all the first coordinates or all the second coordinates of X, Y, Z . Assume that Verifier's challenge was $c = 1$. The only revealed values were random u_1, u_2, u_3, w which satisfy $u_1 + u_2 = u_3 + w$. Because the commitment function $C(\cdot, \cdot)$ is information-theoretically hiding, the unopened second coordinates in the commitments (3) of $\text{COM}(X), \text{COM}(Y), \text{COM}(Z)$ are consistent with any three values $v_{1,1}, v_{2,2}, v_{3,3}$, satisfying $v_{1,1} + v_{2,2} = v_{3,3} - w$. Thus, the interactive proof is consistent with any three vectors X_1, Y_1, Z_1 satisfying the sum equality (3).

A probability of $1/2$ of the Verifier being cheated is of course not acceptable. The probability of being cheated is exponentially reduced by simultaneously employing k repetitions of the process.

Simultaneous verification of several additions. Consider the GSLC (2) which involves n inputs $x_1, \dots, x_n$, and has as output their sum $x_1 + \dots + x_n$. The EP will present to Verifier $2n - 1$ commitments $\text{COM}(X_j), 1 \leq j \leq 2n - 1$, for random representation for the values $x_j, 1 \leq j \leq 2n - 1$. The interactive proofs for correctness of all $n - 1$ claimed equalities $\text{val}(X_{n+1}) = \text{val}(X_1) + \text{val}(X_2), \text{val}(X_{n+2}) = \text{val}(X_{n+1}) +$

$\text{val}(X_3)$, etc., will be done simultaneously for all equations. The EP will present to Verifier $n - 1$ values $w_1, \dots, w_{n-1}$. The Verifier will then randomly choose a challenge $c \leftarrow \{1, 2\}$. The same challenge c will be used by EP and Verifier to check all the $n - 1$ equalities. It is clear that if not all $n - 1$ claimed equations are true, then the probability that Verifier will accept is at most $1/2$. Also, the argument of Theorem 2 that the interactive proof is information-theoretic value-hiding holds without change.

Proving claimed correctness of multiplications. For proving correctness of the operations of multiplication $x_m = x_i \times x_j$ in the SLC (1), the EP will have posted on the SBB for the Verifier commitments $\text{COM}(X_m), \text{COM}(X_i), \text{COM}(X_j)$ for random representations of the values x_m, x_i, x_j . The EP has to prove to Verifier that

$$\text{val}(X_i) \times \text{val}(X_j) = \text{val}(X_m) \quad (5)$$

Let $X_i = (u_1, v_1)$, $X_j = (u_2, v_2)$, and $X_m = (u_3, v_3)$. The EP prepares auxiliary vectors $Z_0 = (u_1 u_2, v_1 v_2)$, $Z_1 = (u_1 v_2 + w_1, p - w_1)$, $Z_2 = (u_2 v_1 + w_2, p - w_2)$, where w_1, w_2 are randomly chosen values. The EP augments the commitments presented to Verifier into:

$$\begin{aligned} &\text{COM}(X_m), \text{COM}(X_i), \text{COM}(X_j), \\ &\text{COM}(Z_0), \text{COM}(Z_1), \text{COM}(Z_2) \end{aligned} \quad (6)$$

Clearly (5) holds if the following Aspects 0–4 hold true for the vectors committed in (6):

Aspect 0: $Z_0 = (u_1 u_2, v_1 v_2)$.

Aspect 1: $\text{val}(Z_1) = u_1 v_2$.

Aspect 2: $\text{val}(Z_2) = u_2 v_1$.

Aspect 4: $\text{val}(X_m) = \text{val}(Z_0) + \text{val}(Z_1) + \text{val}(Z_2)$.

In the interactive proof/verification, either Aspects 0 and 4 are checked together, or Aspect 1 or Aspect 2 is separately checked. The Verifier randomly chooses with probability $1/2$ to verify Aspect 0 and the addition in Aspect 4. He randomly chooses $c \leftarrow \{1, 2\}$. Say $c = 1$. The EP reveals the first coordinates of X_m, X_i, X_j and Z_0 . Aspect 0 is verified. Aspect 4 is verified in the manner of verification of additions. If the EP's claim is false with respect to Aspect 0 or Aspect 4, then the probability of Verifier accepting is at most $3/4 = 1 - (1/2) \times (1/2)$.

The Verifier chooses to check either Aspect 1 or Aspect 2, each with probability $1/4$. Say Aspect 1 was chosen by Verifier. The EP reveals the first coordinate u_1 of X_i and the second coordinate v_2 of X_j and both coordinates of Z_1 and checks the equality of Aspect 1. Note that if Aspect 1 is false and is chosen for verification, then Verifier will never accept. Similarly for Aspect 2. Consequently, if (5) is false and the proof of correctness (6) presented by EP to Verifier is false in Aspect 1, or Aspect 2, then the probability that Verifier will accept is at most $3/4$. Altogether we have:

THEOREM 3. *If the product claim (5) is false then the probability that the Verifier will accept EP's proof of correctness is at most $3/4$.*

REMARK. To achieve the information-theoretic value hiding property of the above interactive proof of correctness, we require an additional step in EP's construction of the posted proof (6). We note that the same x_i can appear in the GSLC (1) as left factor and as right factor. One example arises if the GSLC has an operation $x_m = x_i \times x_j$. In this case, verifying Aspect 1 will reveal both coordinates of X_i and hence reveal the value $x_i = \text{val}(X_i)$. When preparing a proof of correctness of the GSLC (1), the EP creates for every x_i involved in multiplications two random vector representations X_i^L and X_i^R .

The proof of correctness of the multiplication $x_m = x_i \times x_j$ will be:

$$\text{COM}(X_m), \text{COM}(X_i^L), \text{COM}(X_j^R), \\ \text{COM}(Z_0), \text{COM}(Z_1), \text{COM}(Z_2),$$

where now $X_i^L = (u_1, v_1)$, $X_j^R = (u_2, v_2)$. It is clear that even if $i = j$, and Aspect 1 is checked, u_1 and v_2 are independent random values from F_p . Similarly if SLC contains another multiplication $x_k = x_s \times x_j$, it, as well as $x_m = x_i \times x_j$, is verified with respect to Aspect 1. For the first multiplication, X_i^R will be employed, and for the second multiplication, X_j^L will be used. Thus again independent random first coordinate of X_i^R and second coordinate of X_j^L are revealed.

Proving claimed inequalities $x_m = \text{TruthValue}(x_i \leq x_j)$. Such inequalities $x \leq y$ are proved for cases $x, y < p/2$. It is clear that for such x, y , we have $x \leq y$ iff $y - x < p/2$. Example: Let $p = 17$, $x = 7$, $y = 5$.

Then $x \leq y$ is false and $y - x = 15 > 17/2$.

So the EP can prove correctness of inequalities if he can, when true, prove for a commitment $\text{COM}(X)$ that $\text{val}(X) < p/2$.

In Rabin et al.,^{10,11} Lagrange's theorem that every integer x is the sum of four squares of integers: $x = w_1^2 + w_2^2 + w_3^2 + w_4^2$ is employed to enable the EP to create a Value Hiding Proof of the GSLC (1) by use of which he can achieve [Rabin et al.,^{10,11} Theorem 1]:

THEOREM 4. *Let commitments $\text{COM}(X_1), \dots, \text{COM}(X_n)$ to input values $x_1, \dots, x_n$ be posted and let the EP perform the GSLC (1) and post the K output values $x_{(L+1)}, \dots, x_{(L+k)}$. The claimed correctness of the output values can be interactively proven by the EP and a Verifier while keeping all inputs and intermediate values information-theoretically secret. If the Prover's claim is true then the Verifier will always accept the claim. If the Prover's claim is false then the probability that Verifier will accept the claim is at most $3/4$.*

Amplification of Verifier's Confidence

In the previous section, we saw how the EP has expanded the GSLC (1) into a sequence of commitments to be called a Value Hiding Proof (VHP-GSLC). The Value Hiding Proof is employed by EP and VER in an input and intermediate value-hiding interactive proof of correctness of the output values of the GSLC as claimed by the EP. We have shown that the probability of the VER to accept a false claim is at most $3/4$. In applications, a $3/4$ probability of being cheated is of course unacceptable. The solution is of course duplication of the interactive proof in k translations of the GSLC (1). A successful verification of correctness of all k translations by VER will assure him that the probability of him having been cheated by the EP is smaller than $(3/4)^k$.

In practice, the EP may be called upon to interactively prove correctness of announced results to different Verifiers upon K different occasions. So, what is needed is for the EP to prepare and post $K \times k$ Value Hiding Proofs of the GSLC. Next we give an algorithm for doing that.

Making multiple copies of a sequence of hidden values. The reader who is mainly interested in the overall

structure of our results may skip the details of this section and just take for granted its conclusion that many copies of posted hidden values can be made and their value consistency can be proved without revelation of actual values.

In the general case, as well as in the application to securing Vickrey auctions, the EP will have a sequence of m hidden input values $y_1, \dots, y_m$. Some of these inputs were supplied by players $P_1, \dots, P_n$ (in the case of auctions by bidders) and some of these inputs are created by the EP as part of the GSLC computation and proofs that he will conduct.

To begin with, the AU posts on the Secure Bulletin Board $3k$ rows:

$$\text{COM}(Y_1^{(j)}), \dots, \text{COM}(Y_m^{(j)}), \quad 1 \leq j \leq 3k. \quad (7)$$

Each of these $3k$ rows consists of m commitments to vector representations of the m values $\text{val}(Y_i^{(j)}) = y_i$, $1 \leq i \leq m$. For some column indices i , the $3k$ commitments $\text{COM}(Y_i^{(j)})$, $1 \leq j \leq 3k$, to vector representations of the value y_i were provided by one of the players $P_1, \dots, P_n$. For the other column indices i , the $3k$ commitments were supplied by the EP. For a proof of correctness of announced output results, the question arises: How can the EP prove to a Verifier that for each column index i the posted commitments $\text{COM}(Y_i^{(j)})$, $1 \leq j \leq 3k$, all contain vector representations of the same value. That is, how can the EP prove that the rows in (7) are *pairwise value consistent* in the following sense.

DEFINITION 4: *Two rows of commitments*

$$\text{COM}(X_1), \dots, \text{COM}(X_m) \\ \text{COM}(Y_1), \dots, \text{COM}(Y_m) \quad (8)$$

are called *value consistent* if $\text{val}(X_i) = \text{val}(Y_i)$, $1 \leq i \leq m$.

Assume that the EP wants to prove for two posted commitments $\text{COM}(X)$ and $\text{COM}(Y)$, where $X = (u_1, v_1)$ and $Y = (u_2, v_2)$, a claim that $\text{val}(X) = \text{val}(Y)$. He reveals to VER the pair $(w, -w)$ such that $X = Y + (w, -w)$. As in the verification of addition, the Verifier now presents to EP a randomly chosen challenge $c \leftarrow \{1, 2\}$. If $c = 1$, then the EP reveals to VER the first coordinates u_1 and u_2 . The VER checks that $u_1 = u_2 + w$. Similarly if $c = 2$. Clearly, if the EP's claim is false, then the probability that

VER will accept the claim is at most $1/2$.

The same procedure will apply to proving/verifying a claim that the two rows (8) are value consistent. Here, EP posts m vectors $(w_i, -w_i)$, $1 \leq i \leq m$. The VER uses one random challenge $c \leftarrow \{1, 2\}$ to require from the EP to either reveal/open all first coordinates in all commitments or to reveal/open all second coordinates.

We now come to the procedure whereby the EP proves to a VER the value consistency of the initially posted $3k$ rows of commitments (7) and creates additional N rows of commitments to be used in multiple proofs of correctness of announced results of the GSLC.

THEOREM 5. (*Rabin et al.*,^{10,11} Theorem 8) *Let the EP choose an L (say $L = 10$) and prepare and post $M = 10 \times k \times L$ new rows (9):*

$$\text{COM}(X_1^{(j)}), \dots, \text{COM}(X_m^{(j)}), \quad 1 \leq j \leq M, \quad (9)$$

so that each row of (9) is pairwise value consistent with every one of the $3k$ rows (7). That is, for every input index i , $\text{val}(X_i^{(j)}) = y_i$, $1 \leq j \leq M$.

Upon demand, EP can conduct an information-theoretic value-hiding interactive proof convincing a Verifier that:

1. Among the initially posted $3k$ rows (7) at least a majority of $2k$ rows are pairwise value consistent. By definition, the m values $y_1, \dots, y_m$ of the vectors committed to in that $2k$ majority are the input values to the process.

2. In the additional M rows (9) posted by EP, at least $(1 - 1/L)M$ rows are pairwise value consistent with at least $2k$ pairwise value consistent rows of (7).

3. The probability that the Verifier will accept claims 1–2 when not both are true is at most $(1/2 + 1/e^2)^k + (1/2 + 1/e^2)^{3k} < 2(1/2 + 1/e^2)^k$. $\square$

The interactive proof involves EP opening one coordinate in every one of the $3km$ pairs (7) and opening one coordinate in each commitment in $6kL$ rows of (9). Thus this interactive proof leaves $4kL$ untouched rows of (9) with the assurance that at least $(1 - 1/L)4kL$ of these rows are pairwise value consistent with the m values initially committed to in (7). The untouched rows can be employed in $N = 4L = 40$ proofs

In practice, the EP may be called upon to interactively prove correctness of announced results to different Verifiers upon K different occasions.

of correctness of outputs, where each proof uses k rows extended to Value Hiding Proofs. Every such interactive proof employing k rows reduces probability of Verifier being cheated to $(1/10 + 3/4)^k$.

In the following treatment of Vickrey auctions, we shall assume the availability of any needed number of value-consistent rows of commitments to input values without repeating the details as to how these rows were obtained from the initial input rows.

Deniable Revelation of a Value

We want to show that the EP can post commitments $\text{COM}(X)$ to vector presentations of a value x and reveal the value x to a player P in a manner that P can subsequently deny knowledge of the value x . Furthermore, even though the commitments are publicly posted on the SBB and viewable by other players, P cannot open any of these commitments. Consequently, the value x remains information-theoretically hidden from everybody except for the EP and P .

Our algorithm requires a step where P privately meets with the EP in a manner unobserved by anybody else and that P does not carry away from the meeting a record of the value x . The question whether this private meeting can be replaced by exchanges of encrypted messages is a topic for further research.

THEOREM 6. *Assume that the EP has posted on the SBB $20k$ commitments:*

$$(P, \text{COM}(X^{(j)})), \quad 1 \leq j \leq 20k, \quad (10)$$

where P is a name of a player, to random representations of a value x , that is, $\text{val}(X^{(j)}) = x$, $1 \leq j \leq 20k$. Note that these posted commitments are publicly associated with the player P .

The EP reveals the value x to P and claims to him that the posted commitments (10) are to vector representations of this x . The EP can prove to P that the commitments are to random representations of the value x in a manner that (a) If more than $2k$ of the above $20k$ commitments are *not* to vector representations of x , then the probability that P will accept the false claim is at most d^k ; (b) P cannot be compelled to reveal that value x to another party or prove to another party that the commitments are to the value x .

PROOF. Player P meets privately with the EP. The EP claims to P that the hidden value is x . Player P randomly chooses $10k$ commitments out of the $20k$ commitments $(P, \text{COM}(X^{(i)}))$.

For each of the $10k$ commitment $(P, \text{COM}(X^{(i)}))$ chosen by P , the EP privately claims to P that $X^{(i)} = (u^{(i)}, v^{(i)})$. Note that this is what EP claims, without opening the commitment $\text{COM}(X^{(i)})$.

Player P checks that for every claimed value of a vector $X^{(i)}, (u^{(i)} + v^{(i)}) \bmod p = x$.

Next, P chooses, for each of the $10k$ selected $\text{COM}(X^{(i)})$, independently a random challenge $c_i \in \{1, 2\}$ and presents c_i to EP. If $c_i = 1$, then EP opens/reveals to P the first commitment of the chosen pair $\text{COM}(X^{(i)})$. Player P checks that for $\text{COM}(X^{(i)})$, the revealed coordinate value matches the above value $u^{(i)}$ as claimed by EP. Similarly for the case $c_i = 2$. Player P accepts that (10) are $20k$ commitments to representations of the value x only if all the above $10k$ checks are true.

The opening of the commitment to the coordinate c_i is done by EP on the SBB so that the identity of the opened commitment is publicly known.

Why is knowledge of the value x deniable by P ? Player P was privately shown both coordinates, $u^{(i)}$ and $v^{(i)}$, of $10k$ vectors $X^{(i)}$. Thus he has deniability of what he saw. For each of these vectors, the EP publicly opened just one of the two posted commitments $\text{COM}(u^{(i)}), \text{COM}(v^{(i)})$, where $\text{COM}(X^{(i)}) = (\text{COM}(u^{(i)}), \text{COM}(v^{(i)}))$. Hence, nobody except for the EP can open the other coordinate and the value x remains information-theoretically hidden.

We turn to the probability that the player P will accept a false claim. For brevity of discussion, we present a heuristic argument that if, say, $k > 30$ then the value d in the above bound d^k on the probability of P being cheated is close to $\sqrt[1]{(1/e)}$, e being the natural log base 2.71 Namely, if more than $2k$ of the $20k$ vectors $X^{(i)}$ have $\text{val}(X^{(i)}) \neq x$, then the probability for a randomly chosen $X^{(i)}$ to lead P to find that EP is cheating is $>(1/10) \times 1/2$. Consequently, the probability of accepting EP's claim for a randomly chosen $X^{(i)}$ that $\text{val}(X^{(i)}) = x$ is $< 11/20$. For $10k$ choices, the probability of accepting is smaller than $(11/20)^{10k}$. But $(11/20)^{10}$ approximates $\sqrt[1]{(1/e)}$. $\square$

Uncontrollable, Deniable Bidding

We turn to describe the method implementing uncontrollable, deniable bidding by use of deniable revelation of a value. In the following sections, the auctioneer AU will play the role of an evaluator prover EP vis-à-vis the bidders in the auction. The terms AU and EP will be used interchangeably.

Step 6.1 Assume a one-time single-item Vickrey auction. The auctioneer AU, who will later also act as a prover, announces the auction and a reserve price r below which the item will not be sold. AU announces a time T for closing of the auction participation phase. AU also announces a time $T_1 > T$ for completion of submission of bids.

Step 6.2 Assume that bidders $B_1, \dots, B_n$ have decided to participate in the auction. As each bidder B_i declares to AU prior to time T his intention to participate in the auction, the AU assigns to B_i a randomly chosen identification number $id_i \in F_p$ and a randomly chosen value $x_i \in F_p$. The value x_i will be subsequently used to enable B_i to submit his bid in an uncontrollable, deniable way.

The EP posts for every B_i $20k$ pairs $(B_i, \text{COM}(X_i^{(j)}))$, $1 \leq j \leq 20k$, to random vector representations of the value x_i , that is, $\text{val}(X_i^{(j)}) = x_i$, $1 \leq j \leq 20k$.

In a private meeting, EP reveals to B_i the value x_i in a deniable way as discussed earlier.

Step 6.3 To bid the value b_i bidder B_i computes, while still privately meeting with EP, the $z_i \in F_p$ such that $x_i + z_i = b_i \bmod p$.

Bidder B_i prepares $3k$ commitments $\text{COM}(Z_i^{(j)})$, $1 \leq j \leq 3k$, to random vector representations of the value z_i , that is, $\text{val}(Z_i^{(j)}) = z_i$, $1 \leq j \leq 3k$. He digitally signs these commitments and hands them over to EP who posts them on the SBB.

Now B_i erases from his device the values x_i and b_i but retains the value z_i and the data required for opening/decommitting $\text{COM}(Z_i^{(j)})$, $1 \leq j \leq 3k$.

Note that at this point in time, before the closing of the auction, B_i has made his chosen bid b_i , but the EP does not know what that bid value is because he does not know the value z_i .

THEOREM 7. *The above process implements a sealed-bid uncontrollable and deniable submission of a bid by bidder B_i .*

PROOF. The bid value b_i equals the sum $x_i + z_i$. While EP knows the value x_i , he will know the value z_i only after bidder B_i will reveal it to the EP at the closing of the auction at time T_1 . Thus we have a sealed-bid auction.

Bidder B_i cannot be compelled to make a specified bid because until his private meeting with the EP he does not know the value x_i . After he made his bid he can perhaps be made, or volunteer, to reveal the value z_i . But the value x_i was revealed to B_i in a deniable way. As shown earlier and in Theorem 6, B_i can claim anything about that value but can prove nothing about it. Thus this deniability extends to his bid value $x_i + z_i = b_i$.

Conducting the Second Price Auction

The purpose of the following procedure is to enable the EP to prove to bidders who won and what price the winner should pay by referring only to id numbers assigned by the EP to bidders. The procedure keeps bid values information-theoretically secret as well as the correlation between id numbers and actual bidders.

Step 7.1 The EP chooses for every bidder B_i a random identifier id_i . The identifiers are known only to the EP. At time T , the announced end of auction participation phase, the AU will post on the Secure Bulletin Board (SBB) the following data:

$$\begin{aligned} &\langle B_1, \text{COM}(ID_1^{(j)}), \text{COM}(Y_1^{(j)}), \\ &\quad \text{COM}(Z_1^{(j)}), \dots, \\ &\langle B_n, \text{COM}(ID_n^{(j)}), \text{COM}(Y_n^{(j)}), \\ &\quad \text{COM}(Z_n^{(j)}), \dots \rangle, \quad 1 \leq j \leq 3k \end{aligned} \quad (11)$$

where $ID_1^{(j)}$ is the j th random vector representation of the identifier id_1 ; $\text{COM}(Y_1^{(j)})$ is the j th random vector representation of the value x_1 chosen as explained at the end of the section "Deniable Revelation of a Value"; and $Z_1^{(j)}$ is the j th random vector representation of the value z_1 . Similarly for the other subscripts $2, \dots, n$.

Step 7.2 After time T of closing the submission of sealed-bid auction and posting of the $3k$ rows (11), every bidder B_i opens his $3k$ commitments $\text{COM}(Z_i^{(j)})$, for the EP.

The EP chooses $M = 10kL$, $L = 10$, and randomly chooses M permutations π

of the indexes $\{1, \dots, n\}$. The EP creates for every bidder B_i M commitments $\text{COM}(VB_i)$ to random vector representations VB_i of the value B_i (the names of the bidders are ASCII code words reduced to numbers).

The EP now creates and posts M new rows R_{3k+h} , $1 \leq h \leq M$ each row R_{3k+h} , a random permutation π_h of the n quadruples:

$$\begin{aligned} &\langle \text{COM}(VB_1^{(3k+h)}), \text{COM}(ID_1^{(3k+h)}), \\ &\quad \text{COM}(Y_1^{(3k+h)}), \text{COM}(Z_1^{(3k+h)}) \rangle, \dots, \\ &\langle \text{COM}(VB_n^{(3k+h)}), \text{COM}(ID_n^{(3k+h)}), \\ &\quad \text{COM}(Y_n^{(3k+h)}), \text{COM}(Z_n^{(3k+h)}) \rangle, \end{aligned} \quad (12)$$

$1 \leq h \leq M$.

Each of the rows (12) contains $m = 4n$ commitments and, before being permuted, is pairwise value consistent with each of the rows (11) viewed as a sequence of $m = 4n$ commitments to vector representations of values.

Step 7.3 The EP acting as Prover and all bidders $B_1, \dots, B_n$ jointly acting as Verifier, now conduct the secrecy preserving proof noted earlier confirming that out of the $3k$ rows (11) at least $2k$ are pairwise value consistent and out of the new M rows (12) no more than M/L are not value consistent with at least $2k$ majority of the rows (11).

The only new point in this interactive proof is that whenever a row R_{3k+h} is chosen by the Verifier, the EP/Prover opens all the n commitments $\text{COM}(VB_i^{(3k+h)})$ revealing the names $B_1, \dots, B_n$ and ordering the quadruples according to the names.

As mentioned, $4kL$ of the rows R_{3k+h} remain untouched at the end of this Step 7.3.

Step 7.4 Now the EP proves which identifier number id_w had highest bid and which identifier number id_s had the second highest bid. Without revealing bid values and without revealing names of the bidders associated with these identifier numbers, this is done as follows:

The Verifiers $B_1, \dots, B_n$ randomly choose k rows R_{3k+h} out of the $4kL$ remaining untouched rows in Step 7.3. Slightly abusing notations, call these rows $R_1, \dots, R_k$.

The EP orders the identifier numbers $id_1, \dots, id_n$ he has assigned to the bidders $B_1, \dots, B_n$ according to size. This induces a permutation π on the indices $\{1, \dots, n\}$ so that

$$id_{\pi(1)} < id_{\pi(2)} < \dots < id_{\pi(n)} \quad (13)$$

The EP opens in each of the rows $R_1, \dots, R_k$ the n commitments $\text{COM}(ID)$. Thus the rearranged row R_j will look to the Verifiers as:

$$\begin{aligned} &\langle \text{COM}(VB_{\pi(1)}^{(j)}), id_{\pi(1)}, \text{COM}(Y_{\pi(1)}^{(j)}), \\ &\quad \text{COM}(Z_{\pi(1)}^{(j)}) \rangle, \dots, \\ &\langle \text{COM}(VB_{\pi(n)}^{(j)}), id_{\pi(n)}, \\ &\quad \text{COM}(Y_{\pi(n)}^{(j)}), \text{COM}(Z_{\pi(n)}^{(j)}) \rangle, \end{aligned} \quad (14)$$

$1 \leq j \leq k$

Recall that for every quadruple $\langle \text{COM}(VB), id, \text{COM}(Y), \text{COM}(Z) \rangle$, the bid value b of the bidder B to whom the EP secretly attached the identifier number id is $b = \text{val}(Y) + \text{val}(Z)$.

Using the k rows (14) as inputs and noting that the pairwise value consistency for these rows has already been established for the Verifiers, the EP can interactively prove to the Verifiers that for identifier numbers id_w and id_s the bid value represented in the quadruple containing id_w is the highest and the bid value represented in the quadruple containing id_s is the second highest. The interactive proofs are as in Rabin et al.^{10,11} and as detailed in the section discussing previous results.

Step 7.5 Informing the winner, the second highest bidder and the other bidders. The EP now privately proves to the winning bidder that his associated identifier number is the id_w of step 7.4, thereby proving to him that he is the Winner. The EP reveals to the winner in a deniable way that the bid value associated with the identifier number id_s is b_s and collects that payment from the Winner.

In preparation for the kick back promised by the EP/AU to the second highest bidder, the EP privately and deniably proves to the second highest bidder that his identifier number is id_s .

The EP also privately proves to every other bidder that his identifier number is neither id_s nor id_w . These interactive proofs are conducted without revealing to the bidder in question his identifier number.

In the interest of brevity, we omit the detailed constructions of the above proofs. They follow the patterns and employ the tools developed in Rabin et al.^{10,11} and in previous sections of this article.

Countering Collusion

The construction of a bidding process having Properties 1–8 is now complete.

Formation of the cartel. By way of example we assume that seven bidders, $B_1, \dots, B_7$, out of the n bidders get together before the closing of the auction and, following a discussion, agree that:

- Bidder B_i will bid according to strategy s_i , $1 \leq i \leq 7$.
- If a cartel bidder B_i is the winner, he will make side payments $p_j^{(i)}$ to each player $B_j, j \neq i$, in the cartel.

REMARK. Clauses (a)–(b) enable, for example, an agreement that B_1 will be the highest bidder among $B_1, \dots, B_7$, and that if he wins he will make promised side payments to $B_2, \dots, B_7$. On the other hand, if one of $B_2, \dots, B_7$ wins by deviating from the agreement then he will make punitively high side payments to the other cartel members.

THEOREM 8. *If the auction mechanism satisfies conditions 1–9 then collusion is avoidable.*

PROOF. We assume that the bidders $B_1, \dots, B_7$ are independent self-interested entities and that the auction for the item IT with reserve price r is a one-time event.

When announcing the auction, the AU promises in a binding way that the second price bidder B_j among all bidders $B_1, \dots, B_n$, whoever he will turn out to be, will get a kick back payment of $(b_j - r)/k$, where b_j is his bid and r is the announced reserve price. Say $k = 10$.

Now, every cartel member B_i argues for himself as follows. In the proof of correctness of the auction result, all bid values will remain information-theoretically secret. Each of the cartel members can arrange it so that if he wins, the fact that he won will remain unknown to me (bidder B_i). Because that winner is self-interested, he will not make the side payment to me without any danger of reprisal. Also if I win, this fact will remain unknown to everyone except to me and to the AU, hence I shall not need to make any side payments. On the other hand, if I bid $b_i =$ my true private value for IT, then if I win I shall get the IT at the second highest bid value. If I am the

second highest bidder, then I shall get a kick back payment of $(b_i - r)/k$, where b_i = my private true value for the IT. The fact that I got the kick back payment will remain secret and be deniable by me.

The above argument can be strengthened to cover certain instances where the identity of the winner does not remain secret. Namely, the winner B_w has to pay the AU the second highest bid value b_s . But the identity of the second highest bidder B_s is secret from everyone except for the AU and B_s himself. The bidder B_s is informed that he is the second highest in a deniable way. The winner B_w learns from the AU in a deniable way of the payment b_s he has to make. Thus he can claim to other cartel members anything about that payment and consequently cheat them about the level of side payments he has to make.

The only concern that a cartel member B_i planning to deviate from the cartel agreement may have is that if another cartel member was designated as winner and that if he (B_i) bids his true value, he may turn out to be the winner and be subject to a fine payment. If B_i , based on his estimate of bids by other bidders, concludes that this is a likely outcome, then he will deviate only if he knows that his becoming the winner can be concealed. Possible concealment strategies are described following this Theorem.

Conclusion. Cartels are useless and the best strategy for every bidder is to bid his private true value. $\square$

Keeping the winner's identity secret. The possibility of doing so depends on laws governing auctions and on special circumstances of a bidder, auctioneer, and the nature of the item IT up for auction.

For example: If the Auctioneer is a government agency, then there are often transparency requirements with respect to who gets the IT. Similarly, if a bidder is a government agency. The same restrictions may apply to publicly held corporations. In the latter case, the corporation may circumvent restrictions by use of an entity registered in another jurisdiction.

If the IT is a financial instrument, then transfer to the winner may be

secretly done and subsequently known only to tax authorities.

Consider an auction of a large plot of land. If a bidder is a developer intending to build on it, then if he wins the fact is not concealable. If the bidder is an investor who intends to later on resell for a profit, then if he wins he can ask the AU to transfer right to sell to a confidentiality protecting trust. That trust will arrange the transfer of title to a subsequent buyer while keeping the winner's identity concealed.

All in all the possibility of keeping the winner's identity concealed depends on a myriad of legal and practical factors governing the auction in question. It gives rise to creative solutions. It is in the interest of the auctioneer and a winning bidder to cooperate in implementing a solution when legal and possible.

Verification of Stable Matching Solutions

In 1962, Gale and Shapley² formulated the stable matching problem and provided an efficient algorithm for its solution. A number of players $H_1, \dots, H_m$ are looking at a pool of candidates $G_1, \dots, G_k$.

In one example, the players are women, the candidates are men, and $m = k$. Every woman has her ordering of preference of men as spouses, and similarly for every man. A matching is a permutation $\mu: [1, n] \rightarrow [1, n]$ assigning to H_i the spouse $G_{\mu(i)}$. The matching is *stable* if there is no pair of indexes i, j such that H_i prefers $G_{\mu(j)}$ to $G_{\mu(i)}$ and $G_{\mu(j)}$ prefers H_i to H_j . If the latter happens, then H_i can drop $G_{\mu(i)}$ and $G_{\mu(j)}$ will move to H_i .

In another important example, the players are hospital departments (say surgery departments) and the candidates are graduating medical interns looking to become residents. In this case, $k > m$ and every department may induct several residents. Again every department has its ordering of preference of candidates and every candidate has his ordering of preference of departments. These orderings are submitted to an agency that computes a stable matching and announces the assignments while keeping the preferences secret.

Assume that a resident G_i assigned to hospital department H_j suspects that the agency could have assigned him to

another department preferred by him because such a department got assigned a resident less desirable to it than G_i . Upon demand, G_i can get a proof that that is not the case. The proof of correctness does not reveal any preferences, only that G_i was not cheated. Similarly a department can obtain a secrecy preserving proof that no more desirable candidate is willing to move over to it.

Acknowledgments

Michael O. Rabin thanks Eric Maskin for very useful conversations and comments on the topic of Vickrey auctions, and Hal Varian for suggesting the application to matching problems. $\blacksquare$

References

1. Ben-David, A., Nisan, N., Pinkas, B. Fairplaymp a system for secure multi-party computations. In *CCS* (2008), ACM.
2. Gale, D., Shapley, L.S. College admissions and the stability of marriage. *Am. Math. Mon.* 69 (1962), 9–14.
3. Goldreich, O., Micali, S., Wigderson, A. Proofs that yield nothing but their own validity, or all languages in np have zkp systems. *J. ACM* 38 (1991), 692–729.
4. Goldwasser, S., Micali, S., Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM J. Comput.* 18 (1989), 186–208.
5. Graham, D., Marshall, R. Collusive bidder behavior in single-object second-price and English auctions. *J. Polit. Econ.* 95, 6 (1987).
6. Marshall, R., Marx, L. Bidder collusion. *J. Econ. Theor.* 133 (2007), 374–402.
7. Paillier, P. Public-key encryptions based on composite residuosity classes. In *Proceedings of EUROCRYPT 99* (1999), 223–239.
8. Parkes, D., Rabin, M.O., Shieber, S., Thorpe, C. Practical secrecy-preserving verifiably correct and trust worthy auctions. In *Proceedings of 8th International Conference on Electronic Commerce (ICEC)* (2006), 71–81.
9. Pedersen, T. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of CRYPTO 91* (1991), Springer Verlag, 129–140.
10. Rabin, M., Mansour, Y., Muthukrishnan, S., Yung, M. Strictly black-box zero-knowledge and efficient validation of financial transactions. In *Proceedings of ICALP* (2012), Springer Verlag, 738–749.
11. Rabin, M., Servidio, R., Thorpe, C. Practical secrecy-preserving, verifiably correct and trustworthy auctions. In *Proceedings of IEEE Symposium on Logic in Computer Science* (Wrocław, 2007).
12. Sandholm, T. Limitations of the Vickrey auction in computational multi-agent systems. Research Paper. Department of Computer Science, Washington University, St. Louis, MO.
13. Ungern-Sternberg, T.V. Cartel stability in sealed bid second price auctions. *J. Ind. Econ.* 36 (Mar. 1988), 351–358.
14. Vickrey, W. Counterspeculation, auctions, and competitive sealed tenders. *J. Finance* 16 (1961), 8–37.

Silvio Micali (silvio@csail.mit.edu) is the Ford Professor of Engineering in the Electrical Engineering and Computer Science Department at the Massachusetts Institute of Technology, Cambridge, MA. He is co-recipient of the 2012 ACM A.M. Turing Award.

Michael O. Rabin (morabin@gmail.com) is The Thomas J. Watson Professor of Computer Science at Harvard University, Cambridge, MA, and professor of computer science at Columbia University, New York, NY. He is co-recipient of the 1976 ACM A.M. Turing Award.

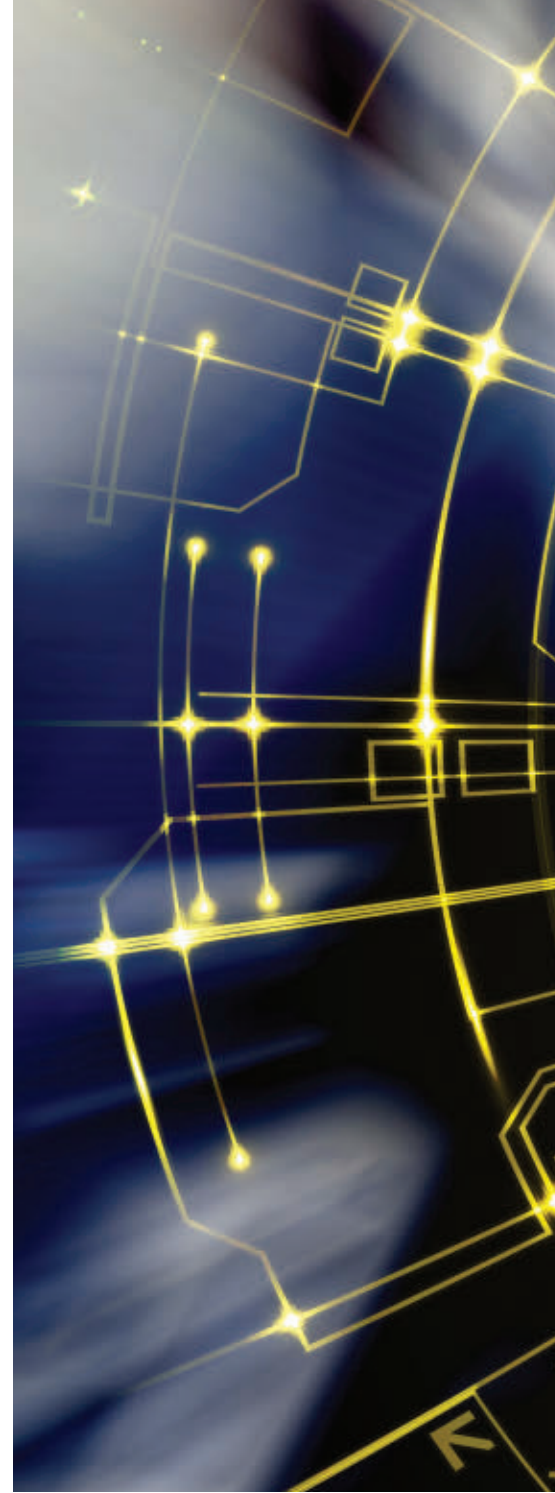
Timing analysis for hard real-time systems.

BY REINHARD WILHELM AND DANIEL GRUND

Computation Takes Time, But How Much?

A FEW YEARS ago, in the pages of this magazine, Edward Lee argued that computing needs time.²³ This article focuses on the natural assumption that computing also takes time. We examine the problem of determining how *much* time. It is the problem of verifying the real-time behavior of safety-critical embedded systems. For such systems, for example, anti-lock brakes and airbags, punctual behavior is of utmost importance: If the controlling computations take too long, quality of service degrades or the systems fail completely—your braking distance is longer or your head hits the steering wheel, respectively.

The basis for verifying the timeliness of system reactions is reliable information on the execution times of all computational tasks involved. It is the job of timing analysis, also called worst-case execution-time (WCET) analysis, to determine such information.



» key insights

- The tremendous progress in microprocessor architecture not only increased average-case performance, but also the complexity of verifying the real-time behavior of programs executed on those architectures.
- To derive useful execution time guarantees, static analyses must prove that speculation mechanisms of modern CPUs will indeed be effective during program runtime.
- This is challenging due to timing anomalies and interdependencies between architectural components.



To avoid having to solve the halting problem, all programs under analysis must be known to terminate. Loops need bounded iteration counts and recursion needs bounded depth—either given explicitly in the program, determined by some analysis, or supplied by the programmer. Furthermore, computing the exact WCET of a program is not necessary. A conservative approximation, such as an upper bound on the execution times of a program, is adequate; and if low enough also sufficient to prove overall timeliness. Figure 1 illustrates the most important notions.

In the old days, such a conservative approximation, called the *timing schema method*, was proposed by Shaw.³³ Its goal is to determine bounds on the execution times of higher-level language programs. The idea is to work along the structurally inductive definition of high-level programming languages, such as along the syntax tree of programs: It starts with bounds on the execution times of atomic program elements and then computes bounds on the execution times of complex constructs by composing the execution times of their components.

For instance, the upper bound on the execution times of a conditional `if b then  $s_1$  else  $s_2$` would be computed as: $ub(\text{if } b \text{ then } s_1 \text{ else } s_2) = ub(b) + \max\{ub(s_1), ub(s_2)\}$.

Today there are at least two reasons that render the timing schema method impractical, infeasible, or imprecise. The first one is compilers. Program transformations and optimizations performed by compilers render source code inadequate for timing analysis: Source code does not reveal the actually executed machine instructions. It does not show the control flow of the

binary program executed on the target machine. Nor does it show the register or memory allocation of program variables and intermediate results. This uncertainty about the actually executed code has already been addressed in Shaw,³³ and has been found to be difficult. Since then, advances in optimizing compilers have only increased this uncertainty.⁴

The second reason is the tremendous progress in computer architecture aiming at ever-increasing (average-case) performance. In the old times, execution times of instructions were constants. With the advent of microprocessors incorporating deep pipelines, caches, and various other speculation concepts, the ex-

ecution times of instructions became variable: The execution times of an instruction may be different for different occurrences of the instruction, that is, at different program points. Execution time may even differ for different executions of an instruction at a single program point. The variations are large: Execution times of instructions may vary by a factor of 100 or more. One could argue that, most of the time, execution is fast since the architectures are optimized for average-case performance. However, this is no foundation for deriving guarantees. On the other hand, starting the structural composition of the timing schema with extremely wide bounds can only result in overall bounds that

are virtually useless.

Let us look into the reasons for the variability of instruction timing more closely: The different execution times of an instruction result from the different states the architecture may be in when execution of the instruction starts. For instance, the time for a load instruction depends on the state of the cache(s) and, maybe, also on the occupancy of the processor memory bus; the time for a conditional branch depends on the state of the branch prediction and may include the time necessary to recovery from misprediction. The architectural state in turn is the result of the execution history, which again is determined by the input to the program and the initial architectural state. Different initial states and different control-flow paths to a program point will result in a set of possible execution states, P , before the instruction at this program point is executed. Later, we describe how a phase called *microarchitectural analysis* computes invariants that characterize these sets of states.

We could then, at least conceptually, try to “expand” the instruction by its implementation in the underlying architectural platform, which is a huge finite-state machine. For this instruction, only a subset of the possible transitions through this finite-state machine are possible, namely those starting in states in P . They would lead to a new set of states reached by executing the instruction.

All paths not starting in states in P can be ignored in the search. Unfortunately, the rest would still be too large to be exhaustively explored. This is the state-space explosion problem encountered by many attempts to exhaustively explore state spaces.

The main measure used in microarchitectural analysis to counter this complexity threat is abstraction. It allows for a compact representation of sets of execution states: Information irrelevant for timing can be dropped completely. Timing-relevant information can be conservatively approximated such that it can be efficiently represented. As we will discuss, there are limits as to how much such an abstraction may forget and still be useful.

At the end of this introduction, it should be clear that timing analysis

Figure 1. Fictitious distribution of execution times of a task exemplifying lower and upper timing bounds (LB, UB) as well as best-case and worst-case execution time (BCET, WCET).

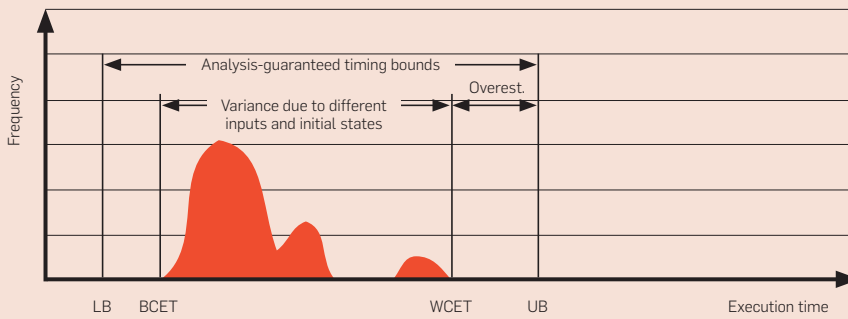
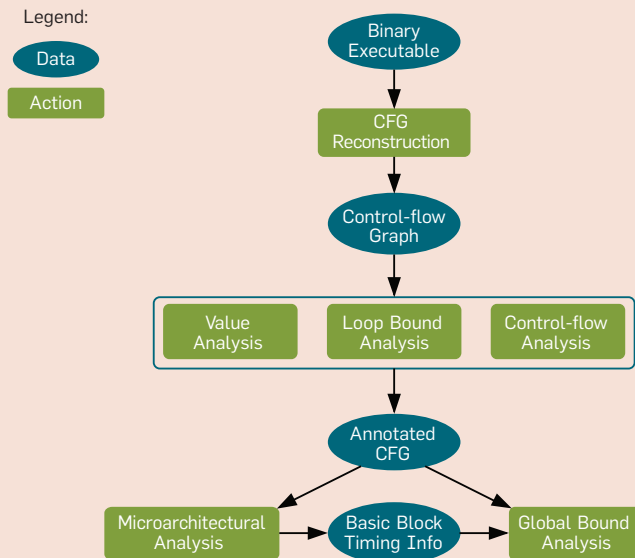


Figure 2. Main phases of a static timing analysis.



does not try to solve the halting problem. All programs under analysis are guaranteed to terminate. The complexity of several subtasks, in particular the huge state space to explore, is the problem to cope with.

Taking the Problem Apart

WCET analysis essentially is the search for a longest path through a program. This can be cast as the problem of constructing a weighted graph and finding a longest path in it: The graph nodes model program fragments, for example, basic blocks, that is, maximally long sequences of straight-line code. The graph edges model possible control flow. The node weights are upper bounds on the execution times of the program fragments, the edge weights are bounds on their traversal counts.

Following this scheme, a quasi-standard architecture for static timing analysis has emerged over the years, as shown in Figure 2. Here, we briefly explain the objective and main challenges of each subtask using the graph terminology.

1. *Control-flow reconstruction*³⁸ determines the control-flow graph itself. It reads the binary executable to be analyzed, reconstructs its control flow, and transforms it into an intermediate program representation. This is a nontrivial task due to dynamically computed control-flow successors, for example, machine code generated for switch statements; or function pointers whose values might not be determined easily. Some instruction-set architectures (ISAs) have additional surprises in store, for example those without a proper instruction for the return from subroutines.

2. *Value analysis* can be seen as an auxiliary analysis. It attempts to statically determine the values stored in registers and memory locations. Such information is needed for loop-bound analysis, to determine the execution time of arithmetic instructions whose timing depends on the values of their operands, for safely approximating effective addresses for data-cache analysis, and to resolve some more value-dependent timing issues. The general problem of value analysis is undecidable, and often several values are possible at a program point when control passes by this program point



The basis for verifying the timeliness of system reactions is reliable information on the execution times of all computational tasks involved.



several times. One out of many approximations⁶ is an interval analysis, which computes enclosing intervals for the set of possible values in registers and memory locations.

3. *Loop bound analysis*^{14,15} determines the edge weights of the graph. It identifies loops in the program and tries to determine bounds on the number of loop iterations. Similarly, recursion must be bounded. Encountered challenges are the analysis of computations on loop counters and loop exit conditions, as well as dependencies between loop counters in nested loops. As the general problem is undecidable, bounds may have to be provided by the user.

4. *Control-flow analysis*^{14,16} also known as infeasible path analysis, is an optional analysis. It determines the set of possible paths through the program more precisely in order to tighten the timing bounds. Its results are additional edge weights, but also more general path constraints. Note that loop bound analysis can be seen as a special, indispensable case of control-flow analysis.

5. *Microarchitectural analysis* determines the node weights of the graph. This subtask will be detailed below as it is the most complex one and provides the most interesting insights.

6. *Global bound analysis*^{1,24,39} also called path analysis, finally determines a longest path in the graph. One approach^{24,39} conveniently relies on integer linear programming to do so: (a) Integer variables model the traversal count of nodes and edges. (b) A set of constraints models the control flow of the program using Kirchhoff's law: The sum of incoming flow at a node must equal the sum of outgoing flow. The incoming flow at the program start node is fixed to 1. (c) Another set of constraints models loop bounds and other path constraints as determined by control-flow analysis. (d) The objective function is the scalar product of the traversal counts and weights of the nodes, that is, execution count times execution time. To compute upper bounds, the objective function is maximized.

The Core of the Problem

In modern computer architectures, speculation is overabundant, it is the

normal mode of operation, and works astonishingly well: Caches speculate on data reuse; branch prediction speculates on the outcome of comparisons, pipelining speculates on the absence of data dependencies; among others. There are even speculations on speculation: Instruction prefetching and speculative execution hope for correct branch prediction. And microarchitectural analysis, which has to determine upper bounds on the execution times of program fragments, has to pay the bill.

Microarchitectural analysis. Computer architects are content when speculation works most of the time—average-case performance is what matters to them. To derive tight timing bounds, however, microarchitectural analysis must prove that the speculation mechanisms work—when they do.

This can possibly be done in many ways. We use abstract interpretation⁶ to compute invariants at each program point that characterize the set of all states the architecture can be in when control reaches this program point. These invariants describe safe information about the contents of the caches;^{10,11} the state of the pipeline including the contents of all its queues and buffers as well as the occupancy of its units;^{12,40} and the state of off-chip buses, memories, and peripherals. The computed invariants are used to exclude transitions in the architecture. For example, a cache-miss transition can be excluded if the invariant about the cache state allows it to predict a cache hit.

If this seems easy to you, let us indicate some pitfalls that rule out some “obvious” optimizations and limit scalability.

Pitfall timing anomalies. It looks tempting to only follow the worst-case transition, for example, the cache-miss transition, when the statically available information admits several possible transitions, for example, cache hit and cache miss. However, there are timing anomalies that make it difficult to decide which transition is the worst.

Intuitively, a timing anomaly is a situation where the local worst case does not entail the global worst case. Consider Figure 3, where a cache miss

to memory block *A*—the local worst case—results in a globally *shorter* execution time than a cache hit. This is because it prevents a branch prediction that would erroneously prefetch *B*. Another example is given in Figure 4. Shortening instruction *A* leads to a longer overall schedule, because instruction *B* can now block the more important instruction *C*, which may only run on Resource 2.

In other words, greedy is not necessarily optimal when maximizing execution times. For details see Lundqvist and Stenström,²⁵ who introduced the notion of timing anomalies, or Reineke and Sen,³⁰ who present a formal definition in the context of WCET analysis.

Pitfall interdependencies. It looks tempting to decompose the architectural analysis into individual analyses of its components. However, architectural components interact in non-trivial ways.

For instance, consider caches in combination with branch prediction: If a branch is mispredicted, instructions are fetched from the wrong branch target before the misprediction is detected and fetching is redirected to the correct branch target. Those extra instruction fetches can evict blocks from the cache that might have been useful for future program execution. Conversely, a data-cache miss can delay the computation of a branch condition, which in turn can trigger a branch prediction, and ultimately speculative fetching and execution of code. If the initial data-cache access were a hit, none of this might have happened. Such (circular) interdependencies also exist between other architectural components.

In particular, the following attempt to analyze the influence of a cache on execution time is unsound:

1. Determine an execution time bound, t_h , for a program assuming that all cache accesses are hits.
2. Determine an upper bound, n , on the overall number of cache misses in any program execution.
3. Take t_h and add n times the cache-miss penalty, t_p , to obtain an upper bound on the execution time that includes cache misses.

In fact, there may be an execution of the program that takes longer, that

is, $t > t_h + n \cdot t_p$. A cache hit may entail a state change in another component such that an even greater penalty is incurred by that other component.

This all suggests that analyzing each architectural component independently of other components is either unsound or imprecise: Unsound if other components and their influence are simply disregarded; imprecise if the analysis always has to account for the worst behavior of other components.

Dire consequences. As a consequence of timing anomalies, microarchitectural analysis must consider all transitions that cannot be ruled out statically. Due to the interdependencies, the currently practiced solution is to analyze all architectural components simultaneously. If one additionally considers this analysis is performed at the granularity of processor cycles, it becomes apparent that this subtask of WCET analysis is the most complex one.

The preceding discussion also illustrates that worst-case execution time—as other nonfunctional properties like maximal stack usage—is very difficult to check experimentally, such as, by testing and measurements. Identifying safe end-of-test criteria for these program properties is an unsolved problem. In consequence, the required test effort is high, the tests require access to the physical hardware, and the results are not complete. The advantage of static analysis is that it enables full control and data coverage, it can be easily automatized, and software developers can run tools from their workstation computers.

Architectural abstractions. As mentioned earlier, abstraction is the strong asset in microarchitectural analysis. It is used to efficiently represent the architectural state at program points and to allow for an efficient state-space exploration.

A feature common to all architectural abstractions is they completely abstract from data. WCET analysis is primarily interested in how long a computation takes; the actually computed values are of no direct interest. Only if values have an influence on the execution time of an instruction they become important. For instance, it makes a difference whether the ad-

dress of a memory access maps to a fast SRAM or a slow external flash memory. Or whether the operand of a variable-latency floating-point multiplication is zero or not.

Information about values is maintained by the preceding value analysis and can be queried if necessary. Factoring out value analysis (as well as control-flow analysis and loop bound analysis) in this way is possible because it solely depends on the instruction set of the considered computing platform. How a particular instruction set is implemented by a microarchitecture, which determines the timing, is irrelevant for them. Hence, all those analyses can be performed at the instruction level, prior to microarchitectural analysis, which is performed at the cycle level. This improves analysis efficiency considerably.

Note that the whole truth may include pleasantries like exposed pipelines or delay slots. The intrepid reader may think about the ramifications of “undead code,” that is, unreachable code that gets executed speculatively; or, similarly, loop bounds that get exceeded by speculative execution.

The next step toward efficient representations is component-specific abstractions. Consider caches for instance. The timing-relevant information about caches is whether the memory block accessed at a program point is contained in the cache or not when execution reaches that program point. This essential information is called *must-* and *may-cache* information:¹⁰ Must-cache information is a set of memory blocks that *definitely* are contained in the cache whenever execution reaches that program point. May-cache information is a set of memory blocks that *may be* contained in the cache whenever execution reaches that program point. The former allows to predict hits; the latter allows to predict misses; and for the memory blocks in the set May/Must both hit and miss need to be taken into account.

Cache abstractions conservatively approximate this information. To be able to maintain useful information on cache updates, abstract caches contain more than just the must- and may-information. Compared to interval and congruence abstractions with rather simple encodings, that is, $[l, u]$ and n

mod m , respectively, the encoding and interpretation of abstract caches is more complicated. Yet abstract caches are elegant and more efficient than explicit encodings of sets of concrete cache states. For a recent overview of cache analysis and examples of cache abstractions, Grund¹¹ or refer to the earlier work.¹⁰

Regarding the success in abstraction, pipelines are a counterexample. Pipelines are much more heterogeneous, that is, they consist of a large number of small components, for example, fetch buffers, dispatchers, execution units, queues for pending load and store instructions, among others. Besides some minor abstractions, for example, abstraction of symmetrical units, no satisfactory abstraction of sets of pipeline states has been found so far. In lieu of compact abstract domains, the domain used for pipeline analysis essentially is a powerset domain: The architectural state of pipelines is approximated by sets of concrete pipelines. For an early example of such a pipeline analysis, see Ferdinand et al.;⁸ for a more complex example, see Thesing.⁴⁰

As explained earlier, all these analyses are performed simultaneously.

Much like the actual hardware is made up of components, the microarchitectural analysis is made up of abstract domains for all components, which are composed using appropriate domain constructors. Overall, microarchitectural analysis is an abstract interpretation with a huge abstract domain. The number of states considered during the analysis of an average-sized basic block for a PowerPC 7448 can grow to seven-figure numbers.

Context sensitivity. Most static program analyses rely on the tacit assumption of control-flow abstraction: Considering the exact set of possible program paths (to a program point) is intractable. A simple abstraction is to approximate the set of possible paths (to a program point) by the set of all paths through the CFG (to that program point). The loss of this abstraction stems from considering infeasible paths through the CFG.

Regarding the WCET bound, infeasible shortcuts are not that problematic but infeasible detours are. Hence, as explained, there is control flow analysis, which determines infeasible paths and thereby narrows down the set of considered paths in order to tighten the timing bounds.

Figure 3. Speculation timing anomaly.

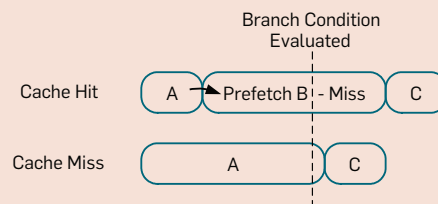
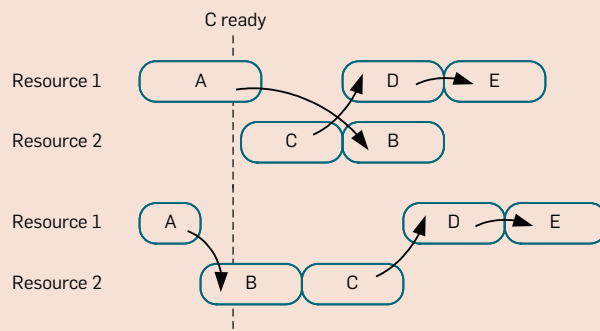


Figure 4. Scheduling timing anomaly. Arrows indicate dependencies.



In general, the WCET problem requires analyses that are highly context-sensitive. That is, they need to distinguish between different possibilities how control reaches a program point. To see why, consider the execution of loops: The first iteration typically exhibits a different architectural behavior than subsequent iterations, for example, the instructions of the loop get loaded into the cache. If one would not distinguish between iterations one would have to conservatively take instruction cache misses into account for all loop iterations. This, however, would lead to a significant overestimation of the execution time.

Similar differences can be exhibited between different call sites of functions. After a first call some of the function's code might remain in the cache. Loop bounds within the function might depend on a function parameter. Different parameter values might induce paths of different lengths through the function.

Essentially contexts allow to infer stronger invariants per (program point, context)-pair compared to a single invariant per program point.

To read more on context sensitivity and special kinds of context sensitivity that have emerged in WCET analysis, see Martin et al.²⁶

Conclusion. Modern CPUs feature an abundance of speculation mechanisms that increase (average-case) performance. To derive tight bounds on program execution times, static WCET analysis must prove that speculation mechanisms indeed work well. As we have seen, this is difficult for many reasons:

Interdependencies between architectural components forbid the analysis of individual components as well as the naive approach of assigning execution time penalties, for example, “add n times the cache miss penalty.” Analyses need to be integrated, which entails higher complexity and increased resource demand. At the same time, pruning of the analysis search space is hindered by timing anomalies and requires precise information about the architectural state. To obtain such precise information, context-sensitive analyses are required that need to distinguish between a large number of different execution histories. This is because the state of components

like caches and branch predictors may depend on events in the distant past. Indeed some may never forget about their history.²⁹

On a more abstract level, CPUs are built to exploit runtime information during the execution of a single program path. WCET analysis must statically prove successful exploitation while efficiency dictates implicit consideration of all program paths. It is fair to say that for hard real-time systems, the high sophistication of some speculation mechanisms is a waste of silicon.

Academic Development

The editorial by Puschner and Burns²⁸ compactly describes the early work on WCET analysis in the 1980s and 1990s. It summarizes the contributions of the then active research groups. The more recent survey of Wilhelm et al.⁴³ takes a more problem-centric point of view. It discusses different approaches at the subtasks of timing analysis, contributing groups, as well as the strengths and limits of the tools available at that time.

In principle, one can distinguish between methods based on measurements, simulation, or static analysis. The WCET estimates determined by measurements or simulation are unsafe as inclusion of the worst-case combination of program input and initial architectural state can rarely be guaranteed. Static methods can provide guarantees but may suffer from overestimation. Performing measurements requires the hardware and tracing equipment; simulations and static analysis require models of the architecture.

Some research groups invested into academic prototypes, some of which made the leap to commercially available tools. The most widely known tools that are still maintained are:

(a) aiT, a commercial tool by AbsInt GmbH, Saarbrücken, Germany, based on static analysis;

(b) BoundT, a commercial tool by Tidorum Ltd., Helsinki, Finland, based on static analysis;

(c) Chronos, an open source software developed by National University of Singapore, employs the SimpleScalar simulator for microarchitectural analysis;

(d) OTTAWA, an open source software developed by University of Tou-

louse, France, based on static analysis;

(e) RapiTime, a commercial tool by Rapita Systems Ltd., York, U.K., based on measurements; and,

(f) SWEET, an academic prototype of Mälardalen University, Sweden, focusing on static control-flow analysis.

For a more complete list and a deeper discussion of functionalities and limitations we refer to Wilhelm et al.⁴³

Regarding sound static analysis approaches and saving open and upcoming challenges for later discussion, the most important developments were (a) the timing schema introduced by Shaw³³ and its extension; (b) the switch to the more suitable implicit path enumeration technique (IPET), initially proposed by Li and Malik,²⁴ which is still used today for global bound analysis; (c) dissection of the timing analysis task into controllable subtasks;⁸ (d) approaches to microarchitectural analysis, leading to complete models of complex processors;⁴⁰ (e) abstractions for the huge state spaces of cache architectures;^{10,11} and (f) relaxations of the uninterrupted-execution assumption,² which we will discuss in further detail.

The WCET research group at Mälardalen University maintains a number of WCET benchmark programs^a used to evaluate and compare different types of WCET analysis tools and methods.¹³ Since 2006, the WCET community periodically performs a WCET Tool Challenge.^b The published results³⁷ give a good overview of the state of the different tools.

Industrial Adoption

Until the 1990s, timing analysis in industrial contexts was dominated by measurement-based techniques and simple counting methods. The strand of research on static methods reached a milestone in 1998 with the founding of the company AbsInt Angewandte Informatik GmbH. After preliminary discussions with the German TÜVs (technical inspection offices) the market potential of Abstract Interpretation based WCET analysis seemed to justify the commercialization effort. Airbus was among the first companies

a <http://www.mrtc.mdh.se/projects/wcet/benchmarks.html/>

b <http://www.mrtc.mdh.se/projects/WCC/>

to recognize the potential of the novel technique. Airbus was instrumental in the European IST project Daedalus (Validation of critical software by static analysis and abstract testing), in the course of which AbsInt adapted its prototypical tool chain for WCET analysis to the industrial requirements for avionics software.^{34,35,41} The first processors supported by the new commercial WCET analysis tool were Motorola PowerPC 755 and TI TMS470. Today, this tool, which implements the architectures described earlier, is known as aiT WCET Analyzer.

Initially reported results³⁵ pertain to the more complex PowerPC 755, whose variance of execution times for instructions is in the order of a factor of several hundreds. The comparison shows that the computed upper bound of a task typically is about 25% higher than the measured time for the same task, the real but non-calculable WCET being in between. Analysis time was 12 hours per program on average and the maximal memory demand was close to 3GiB. Since then, the aiT tool chain has been continuously improved by incorporating new research results and now supports some 20 processor targets.^c For simpler microcontrollers overestimation is below 10%.^d

Tool couplings to other development tools have been implemented. For instance between aiT and model-based code generators like Esterel SCADE⁹ and dSPACE TargetLink.²⁰ They enable the worst-case timing behavior to be continuously evaluated during software development. Links between analysis results and the model level enable timing information to be traced back to the model level. Errors can be detected early in the development process, thus avoiding late-stage integration problems. Another important process optimization can be realized by integrating tools for computing the worst-case execution time and the worst-case response time. A tool coupling between aiT and the SymTA/S tool from Symtvision provides a seamless approach to timing analysis: SymTA/S computes the end-to-end times and worst-case response times of the system based



The worst-case execution time estimates determined by measurements or simulation are unsafe as inclusion of the worst-case starting conditions can rarely be guaranteed. Methods based on static analysis can provide guarantees but may suffer from overestimation.



on the worst-case execution times computed by aiT.¹⁹ From a safety assurance perspective, typically model-based testing is used for showing functional program properties, and static analysis to prove the absence of non-functional program errors. Therefore it can be very beneficial to integrate model-based testing and analysis, which has been addressed by a tool coupling between aiT and the model-based testing tool BTC EmbeddedTester.¹⁸ Model-level information like execution model or environment specifications is automatically taken into account, avoiding duplicate effort for test and analysis setup. Tests and analyses can be launched seamlessly and produce unified result reports.

During the last years, most of the relevant safety standards have been undergoing major revisions, for example, DO-178, IEC-61508, and CEN-EC EN-50128. The norm ISO-26262 defining functional safety for road vehicles was published in the year 2011. All of them require to identify potential functional and nonfunctional hazards and to demonstrate that the software does not violate the relevant safety goals. All mentioned standards list the worst-case execution time to the software properties that have to be determined for real-time software. When used in the certification process of safety-critical systems, tools must be qualified according to the pertinent safety standard. To support this, AbsInt has developed Qualification Support Kits (QSK) for aiT, which can demonstrate the tool works correctly in the operational context of the user. Additionally, Qualification Software Life Cycle Data (QSLCD) reports are available that document the entire tool development process of aiT for all target processors and compilers, including all verification and quality assurance activities. QSK and QSLCD enable the tool qualification according to any safety standard to be performed in a mostly automatic way up to the highest criticality levels.²²

Customers of aiT come from all safety-critical industry sectors: Avionics and space, automotive, nuclear power plant control, healthcare technology, among others. Regrettably, most aiT customers do not agree to be referenced. Some who agreed can

c <http://www.absint.com/ait/targets.htm/>

d <http://www.absint.com/ait/precision.htm/>

be found at <http://www.absint.com/success.htm>. In 2010, aiT was used by NASA as an industry-standard tool for demonstrating the absence of timing-related software defects in the Toyota Motor Corporation Unintended Acceleration Investigation.²⁷

Open Questions and Future Challenges

In this article, we described a solution for the WCET-analysis problem. However, there are still shortcomings whose removal will increase general applicability. Here, we discuss these shortcomings as well as future threats to the viability of this solution.

For the sake of completeness, let us list the underlying assumptions of our approach. Regarding the program to be analyzed: termination, no self-modifying code, no dynamic memory allocation, and resolvable dynamic branch targets.

These are rather easy to satisfy and missing information can be supplied by the developer or derived from a higher-level model. Although the source code might be subject to regulations or norms, this is irrelevant for WCET analysis as it requires binaries as input.

Static analysis requires a truthful model of the architecture to be analyzed. Current abstract models are crafted by studying hardware documentation, by querying designers, by asking somebody who knows, and, if necessary, by performing reverse-engineering experiments. The upcoming alternative to this error-prone and laborious process is to derive abstract models from VHDL or Verilog specifications.³¹

One important assumption we make on the system level is that programs are executed in isolation. If program execution were interrupted, and other code were executed, the architectural state would be different when program execution would resume. Hence, the computed invariants on the architectural state at all following program points would be wrong. One needs additional analyses that allow to bound the interruption-caused increase in execution time.² Nonpreemptive scheduling, as found in avionics for example, is the easier choice though, at least concerning WCET analysis.



Necessary effort and achievable precision for the single-core setting meet industrial requirements. For most current multicore platforms, precision and/or complexity are unacceptable.



In fact one can state this assumption in a more general way: Analysis takes account only of events whose occurrence can be associated with specific program points. Other types of events that happen without such an association and that alter the architectural state have to be dealt with separately. Preemptive scheduling, possibly implemented using hardware interrupts, is only one example. Others include DRAM refreshes, DMA transfers, or even the parallel execution of programs on multiprocessor or multicore platforms.

The problem with parallel execution of programs is the induced interference on architectural resources that are shared between programs, for example, caches, interconnects, flash memories, peripherals, and so on. Some accesses to shared resources, in particular global variables, will be synchronized to guarantee the semantics of the co-running programs. For the rest, the order in which competing threads access shared machine resources is not statically fixed. This complicates timing analysis as any access to a shared resource might be delayed due to competing accesses. One approach to statically prove the absence of such delays is to analyze each program given abstractions of the resource access behaviors of all other co-running programs. One example is given by Schranzhofer et al.,³² but finding suitable abstractions for the general problem is unsolved.

To mitigate problems in timing analysis, one can try to reduce interferences on resources by smart configuration of the CPU or system board.^{7,21} As an alternative to tilting at windmills, that is, trying to fix designs that are less favorable for WCET analysis, one can try to design the architectures appropriately in the first place.^{42,44} Appropriately meaning that one can easily predict their behavior while they still exhibit high performance. The need for predictability was recognized early³⁶ and has since been inspected in several ways, for example, Henzinger,¹⁷ Bernardes,⁵ and Thiele.⁴² However, the understanding of predictability in the real-time community is rather implicit. A generally accepted formal definition is still to be found. The joint article by Axer et al.³ discusses predict-

ability fundamentally and at several abstraction levels of systems.

Conclusion

Timing analysis has been made difficult by the use of high-performance microprocessors, which use caches, deep pipelines, out-of-order execution, and branch prediction to improve average-case performance. These architectural components have introduced a large variability of the execution times of individual instructions by the dependence on the execution state. The solution is to safely bound the execution times of sequences of instructions occurring in the program based on information about all possible execution histories leading to these occurrences. Static program analysis is used to compute such information. Reliable and precise upper bounds can be computed. Timing-analysis tools are in routine use in the safety-critical embedded-systems industries.

The possibility to determine execution-time bounds and the precision of the results depend heavily on properties of the underlying computer architecture. Trends in computer architecture and in software design threaten the applicability of established methods. The fact the timing analysis problem could be solved in a provably correct way is considered one of the success stories of formal methods. Continuation of this story will require more predictable architectures as well as advances in analysis technology.

Acknowledgments

We thank our many colleagues who have contributed to the described approach, including Christian Ferdinand, Florian Martin, Henrik Theiling, Michael Schmidt, and Stephan Thesing, Reinhold Heckmann, Daniel Kästner, and Jan Reineke.

The development of the technology was supported by the Transfer Project 14 of the Deutsche Forschungsgemeinschaft, the European IST project Daealus, and the Transregional Research Center AVACS of the Deutsche Forschungsgemeinschaft. 

References

- Althaus, E., Altmeyer, S. and Naujoks, R. Precise and efficient parametric path analysis. In *Proceedings of the ACM SIGPLAN/SIGBED 2011 Conference on Languages, Compilers, and Tools for Embedded Systems*. ACM, NY (Apr. 2011), 141–150.
- Altmeyer, S., Davis, R.I. and Maiza, C. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems* 48, 5 (2012), 499–526.
- Axer, P. et al. Building timing predictable embedded systems. *Trans. Embedded Computing Systems* (2012).
- Balakrishnan, G. and Reps, T. WYSINWYX: What you see is not what you execute. *ACM Trans. Program. Lang. Syst.* 32, 6 (Aug. 2010), 23:1–23:84.
- Bernardes, J.N.C. On the predictability of discrete dynamical systems. In *Proc. of the American Math. Soc.* 130, 7 (2001), 1983–1992.
- Cousot, P. Abstract interpretation based formal methods and future challenges. *Informatics—10 Years Back, 10 Years Ahead* (2001), 138–156.
- Cullmann, C. et al. Predictability considerations in the design of multi-core embedded systems. In *Proceedings of Embedded Real Time Software and Systems* (May 2010), 36–42.
- Ferdinand, C. et al. Reliable and precise WCET determination for a real-life processor. In *Proceedings of the First International Workshop on Embedded Software* (London, U.K., 2001). Springer, 469–485.
- Ferdinand, C. et al. Combining a high-level design tool for safety-critical systems with a tool for WCET analysis on executables. In *Proceedings of the 4th European Congress ERTS Embedded Real-Time Software* (Toulouse, France, Jan. 2008).
- Ferdinand, C. and Wilhelm, R. Efficient and precise cache behavior prediction for real-time systems. *Real-Time Systems* 17, 2-3 (1999), 131–181.
- Grund, D. Static Cache Analysis for Real-Time Systems—LRU, FIFO, PLRU. Ph.D. thesis. Saarland University, 2012.
- Grund, D., Reineke, J. and Gebhard, G. Branch target buffers: WCET analysis framework and timing predictability. *J. Systems Architecture* 57, 6 (2011), 625–637.
- Gustafsson, J., Betts, A., Ermedahl, A. and Lisper, B. The Mälardalen WCET benchmarks: Past, present and future. In *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis*.
- Gustafsson, J., Ermedahl, A., Sandberg, C. and Lisper, B. Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium* (Washington, D.C., 2006). IEEE-CS, 57–66.
- Healy, C., Sjödin, M., Rustagi, V., Whalley, D. and van Engelen, R. Supporting timing analysis by automatic bounding of loop iterations. *Real-Time Systems* 18 (2000), 129–156.
- Healy, C. and Whalley, D. Automatic detection and exploitation of branch constraints for timing analysis. *IEEE Trans. Software Engineering* 28, 8 (2002), 763–781.
- Henzinger, T. Two challenges in embedded systems design: Predictability and robustness. *Philos. Trans. Royal Soc. Math., Phys. and Engin. Sciences*, 366, 1881 (2008), 3727–3736.
- Kästner, D. et al. Leveraging from the combination of model-based analysis and testing. *Embedded World Congress*, 2013.
- Kästner, D., Ferdinand, C., Heckmann, R., Jersak, M. and Gliwa, P. An integrated timing analysis methodology for real-time systems. *SAE World Congress*. SAE International, 2011.
- Kästner, D. et al. Integrating model-based code generators with static program analyzers. *Embedded World Congress*, 2013.
- Kästner, D. et al. Meeting real-time requirements with multi-core processors. In *Proceedings of 2012 Workshop: Next Generation of System Assurance Approaches for Safety-Critical Systems*. (Sept. 2012).
- Kästner, D. and Ferdinand, C. Efficient verification of non-functional safety properties by abstract interpretation: Timing, stack consumption, and absence of runtime errors. In *Proceedings of the 29th International System Safety Conference* (Las Vegas, 2011).
- Lee, E.A. Computing needs time. *Commun. ACM* 52, 5 (May 2009), 70–79.
- Li, Y.-T.S. and Malik, S. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference* (New York, NY, 1995). ACM, NY, 456–461.
- Lundqvist, T. and Stenström, P. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings of the 20th IEEE Real-Time Systems Symposium* (Dec. 1999), 12–21.
- Martin, F., Alt, M., Wilhelm, R. and Ferdinand, C. Analysis of loops. *Compiler Construction* (1998), 80–94.
- National Highway Traffic Safety Administration Toyota unintended acceleration investigation. Technical Report TT-10-00618, NASA Engineering and Safety Center, January 2011.
- Puschner, P. and Burns, A. Guest editorial: A review of worst-case execution-time analysis. *Real-Time Systems* 18 (2000), 115–128.
- Reineke, J. and Grund, D. Sensitivity of cache replacement policies. *ACM Trans. Embedded Computing Systems* (2013).
- Reineke, J. et al. A definition and classification of timing anomalies. In *Proceedings of 6th International Workshop on Worst-Case Execution Time Analysis* (July 2006).
- Schlickling, M. and Pister, M. Semi-automatic derivation of timing models for WCET analysis. In *Proceedings of the ACM SIGPLAN/SIGBED 2010 Conference on Languages, Compilers, and Tools for Embedded Systems* (Apr. 2010), 67–76. ACM, NY.
- Schranzhofer, A., Pellizzoni, R., Chen, J.-J., Thiele, L. and Caccamo, M. Timing analysis for resource access interference on adaptive resource arbiters. *IEEE Real-Time and Embedded Technology and Applications Symposium* (2011), 213–222.
- Shaw, A.C. Reasoning about time in higher-level language software. *IEEE Trans. Software Eng.* 15, 7 (1989), 875–889.
- Souyris, J. Industrial experience of abstract interpretation-based static analyzers. *Building the Information Society*. R. Jacquot, ed. IFIP 156 (2004), 393–400. Springer, Boston.
- Souyris, J., Pavec, E.L., Himpert, G., Jégu, V. and Borios, G. Computing the worst-case execution time of an avionics program by abstract interpretation. In *Proceedings of the 5th Intl. Workshop on Worst-Case Execution Time Analysis* (2005), 21–24.
- Stankovic, J. and Ramamritham, K. What is predictability for real-time systems? *Real-Time Syst.* 2 (1990), 247–254.
- Tan, L. The worst-case execution time tool challenge 2006. *International J. Software Tools for Technology Transfer* 11, 2 (2009), 133–152.
- Theiling, H. Extracting safe and precise control flow from binaries. In *Proceedings of the 7th International Conference on Real-Time Systems and Applications* (Washington, D.C. 2000). IEEE-CS, 23–30.
- Theiling, H. ILP-based interprocedural path analysis. *Lecture Notes in Computer Science, EMSOFT 2491* (2002), 349–363. Springer.
- Thesing, S. Safe and Precise WCET Determinations by Abstract Interpretation of Pipeline Models. Ph.D. thesis, Saarland University, 2004.
- Thesing, S. et al. An abstract interpretation-based timing validation of hard real-time avionics software systems. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks* (June 2003). IEEE-CS, 625–632.
- Thiele, L. and Wilhelm, R. Design for timing predictability. *Real-Time Systems*, 28, 2-3 (2004), 157–177.
- Wilhelm, R. et al. The worst-case execution-time problem—Overview of methods and survey of tools. *ACM Trans. Embedded Computing Systems* 7, 3 (2008), 1–53.
- Wilhelm, R. et al. Memory hierarchies, pipelines, and buses for future time-critical embedded architectures. *IEEE TCAD* 28, 7 (July 2009), 966–978.

Reinhard Wilhelm (wilhelm@cs.uni-saarland.de) is a professor at Saarland University, Saarbrücken, Germany, where he has served as chair for programming languages and compiler construction since 1978.

Daniel Grund (daniel.grund@thalesgroup.com) is currently a systems architect Thales Transportation Systems in Stuttgart, Germany. He contributed to the work described in this article while he was a researcher at Saarland University, Saarbrücken, Germany.



Association for
Computing Machinery

Advancing Computing as a Science & Profession

membership application & digital library order form

Priority Code: AD13

You can join ACM in several easy ways:

Online

<http://www.acm.org/join>

Phone

+1-800-342-6626 (US & Canada)
+1-212-626-0500 (Global)

Fax

+1-212-944-1318

Or, complete this application and return with payment via postal mail

Special rates for residents of developing countries:

<http://www.acm.org/membership/L2-3/>

Special rates for members of sister societies:

<http://www.acm.org/membership/dues.html>

Please print clearly

Name

Address

City

State/Province

Postal code/Zip

Country

E-mail address

Area code & Daytime phone

Fax

Member number, if applicable

Purposes of ACM

ACM is dedicated to:

- 1) advancing the art, science, engineering, and application of information technology
- 2) fostering the open interchange of information to serve both professionals and the public
- 3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature

ACM Code of Ethics:

<http://www.acm.org/about/code-of-ethics>

choose one membership option:

PROFESSIONAL MEMBERSHIP:

- ☐ ACM Professional Membership: \$99 USD
- ☐ ACM Professional Membership plus the ACM Digital Library:
\$198 USD (\$99 dues + \$99 DL)
- ☐ ACM Digital Library: \$99 USD (must be an ACM member)

STUDENT MEMBERSHIP:

- ☐ ACM Student Membership: \$19 USD
- ☐ ACM Student Membership plus the ACM Digital Library: \$42 USD
- ☐ ACM Student Membership PLUS Print CACM Magazine: \$42 USD
- ☐ ACM Student Membership w/Digital Library PLUS Print
CACM Magazine: \$62 USD

- ☐ **Join ACM-W:** ACM-W supports, celebrates, and advocates internationally for the full engagement of women in all aspects of the computing field. Available at no additional cost.

payment:

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc. in US dollars or foreign currency at current exchange rate.

☐ Visa/MasterCard ☐ American Express ☐ Check/money order

☐ Professional Member Dues (\$99 or \$198) \$ _____

☐ ACM Digital Library (\$99) \$ _____

☐ Student Member Dues (\$19, \$42, or \$62) \$ _____

Total Amount Due \$ _____

Card #

Expiration date

Signature

RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

**All new professional members will receive an
ACM membership card.**

Questions? E-mail us at acmhelp@acm.org
Or call +1-800-342-6626 to speak to a live representative

Satisfaction Guaranteed!

research highlights

P. 106

**Technical
Perspective**
**A New Spin on
an Old Algorithm**

By Michael W. Mahoney

P. 107

**Communication Costs of
Strassen's Matrix Multiplication**

By Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz

Technical Perspective

A New Spin on an Old Algorithm

By Michael W. Mahoney

COMMUNICATION—THE COST of moving bits between levels of the memory hierarchy on a single machine or between machines in a network or data center—is often a more precious resource than computation. Although not new, communication-computation trade-offs have received renewed interest in recent years due to architectural trends underlying high-performance computing as well as technological trends that permit the automatic generation of enormous quantities of data. On the practical side, this has led to multicore processors, libraries such as LAPACK and ScaLAPACK, schemes such as MPI and MapReduce, and distributed cloud-computing platforms. On the theoretical side, this has motivated a large body of work on new algorithms for old problems under new models of data access.

Into this fray enters the following paper by Ballard, Demmel, Holtz, and Schwartz, which considers a fundamental problem, adopting a new perspective on an old algorithm that has for years occupied a peculiar place in the theory and practice of matrix algorithms. In doing so, the work highlights how abstract ideas from theoretical computer science (TCS) can lead to useful results in practice, and it illustrates how bridging the theory-practice gap requires a healthy understanding of the practice.

The basic problem is the multiplication of two $n \times n$ matrices. This is a fundamental primitive in numerical linear algebra (NLA), scientific computing, machine learning, and large-scale data analysis. Clearly, n^3 time is a trivial lower bound—that much time is necessary to read the input and write the output. Moreover, at first glance, it seems “obvious” the ubiquitous three-loop algorithm for multiplying two matrices (given as input two $n \times n$ matrices, A and B , for each i, j, k , do: $C(i, j) += A(i, k) * B(k, j)$) shows that a constant times n^3 time is needed to solve the problem.

Back in 1969, it was surprising when Strassen presented his by-now well-known algorithm. The basic idea is two 2×2 matrices can be multiplied using 7, rather than the usual 8, multiplications. Since the same idea applies to 2×2 block matrices, the natural recursive extension can be used to multiply two $n \times n$ matrices in no more than a constant times n^ω arithmetic operations, where $\omega = \log_2 7 \approx 2.808$. Over the years, the exponent ω has been whittled down to $\omega \approx 2.373$, and many conjecture that there exist Strassen-like algorithms with $\omega = 2$.

Strassen’s algorithm highlights the distinction, extremely important in TCS, between problems and algorithms; and it demonstrates that non-obvious algorithms can have better running times, in theory at least, than the obvious algorithm. Although its running time can be better than the usual three-loop algorithm for input matrices larger than ca. 100×100 , Strassen’s algorithm has, for both technical and non-technical reasons, yet to be widely used in practice.

This paper is part of a larger body of work on minimizing communication in NLA algorithms. Previous work has shown that geometric embedding methods can be used to establish communication lower bounds for three-loop matrix multiplication algorithms in both shared-memory sequential and distributed-memory parallel models. Basically, the algorithm can be modeled as a computation directed acyclic graph (CDAG). Due to the three-loop structure of the algorithm, this graph can be embedded into a 3D cube; and from the isoperimetric properties of that embedding a lower bound on communication can be established. The main result of this paper is a new lower bound on the amount of communication for both sequential and parallel versions of Strassen-like algorithms that is lower than the lower bound of the usual three-loop algorithm.

Since the geometric embedding methods do not seem to apply to the recursive structure of Strassen-like algorithms, the new lower bound is established by considering the edge expansion of the CDAG of Strassen’s algorithm. Expanders—graphs that do not have any good partitions and that do not embed well in any low-dimensional Euclidean space—are remarkably useful structures that are ubiquitous within TCS and almost unknown outside TCS. For readers familiar with expanders, this paper will provide yet another application. For readers not familiar with expanders, this paper should be a starting point.

Finally, in a stroke that will make practitioners of numerical analysis and data analysis—as well as lower bound complexity theorists—happy, the authors also show their lower bounds are tight by providing an optimal algorithm. In the sequential case, this is attained by the standard implementation of Strassen’s algorithm; and, in the parallel case, the authors, in joint work with Benjamin Lipshitz, have developed a novel Communication Avoiding Parallel Strassen algorithm. This latter algorithm communicates asymptotically less than previous three-loop and Strassen-based algorithms; and its empirical performance exceeds all other known matrix multiplication algorithms, three-loop or Strassen-based, on large parallel machines. Remarkably, this suggests that Strassen’s algorithm should be adopted into existing parallel NLA libraries, providing a great example of how to bridge the theory-practice gap, and suggesting that Strassen’s algorithm might still see practical use—ironically, though, due to its better communication properties. 

Michael W. Mahoney (mmahoney@icsi.berkeley.edu) is at the International Computer Science Institute and the Department of Statistics at the University of California at Berkeley.

Copyright held by Author.

Communication Costs of Strassen's Matrix Multiplication

By Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz

Abstract

Algorithms have historically been evaluated in terms of the number of arithmetic operations they performed. This analysis is no longer sufficient for predicting running times on today's machines. Moving data through memory hierarchies and among processors requires much more time (and energy) than performing computations. Hardware trends suggest that the relative costs of this communication will only increase. Proving lower bounds on the communication of algorithms and finding algorithms that attain these bounds are therefore fundamental goals. We show that the communication cost of an algorithm is closely related to the graph expansion properties of its corresponding computation graph.

Matrix multiplication is one of the most fundamental problems in scientific computing and in parallel computing. Applying expansion analysis to Strassen's and other fast matrix multiplication algorithms, we obtain the first lower bounds on their communication costs. These bounds show that the current sequential algorithms are optimal but that previous parallel algorithms communicate more than necessary. Our new parallelization of Strassen's algorithm is communication-optimal and outperforms all previous matrix multiplication algorithms.

1. INTRODUCTION

Communication (i.e., moving data) can greatly dominate the cost of an algorithm, whether the cost is measured in running time or in total energy. This holds for moving data between levels of a memory hierarchy or between processors over a network. Communication time per data unit varies by orders of magnitude, from order of 10^{-9} seconds for an L1 cache reference to order of 10^{-2} seconds for disk access. The variation can be even more dramatic when communication occurs over networks or the internet. In fact, technological trends^{16, 17} are making communication costs grow exponentially over time compared to arithmetic costs. Moore's Law is making arithmetic on a chip improve at about 60% per year, but memory and network bandwidth is improving at only 26% and 23% per year.¹⁶ So even in cases where communication is not the bottleneck today, it may be in the future.

Ideally, we would be able to determine lower bounds on the amount of required communication for important problems and design algorithms that attain them, namely, algorithms that are communication-optimal. These dual problems have long attracted researchers, with one example being classical $\Theta(n^3)$ matrix multiplication (see further details below), with lower bounds proved in Hong and Kung¹⁸ and Irony et al.²⁰ and many optimal sequential and parallel algorithms obtained in, for example, Agarwal et al.¹ and Cannon¹¹.

These lower bounds have recently been extended to a large class of other classical linear algebra problems, including linear system solving, least squares, and eigenvalue problems, for dense and sparse matrices, and for sequential and parallel machines.⁹ Surprisingly, the highly optimized algorithms in widely implemented libraries like LAPACK and ScaLAPACK³ often do not attain these lower bounds, even in the asymptotic sense. This has led to much recent work inventing new, faster algorithms that do; see the citations in Ballard et al.^{9, 10} for references.

In this paper, we describe a novel approach to prove the first communication lower bounds for Strassen's $\Theta(n^{\log_2 7})$ matrix multiplication algorithm, as well as many similar fast algorithms. Specifically, we introduce expansion analysis of the computational graphs of the algorithms and show that the expansion helps determine the communication cost. These communication cost bounds are *lower* than those of classical matrix multiplication: this means that not only does Strassen's algorithm reduce computation, but it also creates an opportunity for reducing communication. In addition, the lower bound decreases as the amount of available memory grows, suggesting that using extra memory may also allow for faster algorithms.

In fact, there is an optimal parallel algorithm that attains our lower bounds for varying amounts of memory, whose performance exceeds all other known matrix multiplication implementations, classical or Strassen-based, on a large parallel machine,⁶ see Figure 1. In the rest of this paper, we focus on explaining our new lower bounds for Strassen's algorithm and their implications.

1.1. Communication models

In order to analyze the communication costs of algorithms, we consider idealized memory and communication models. In the sequential case (see Figure 2), we consider a machine with two levels of memory hierarchy: a fast memory of size M words (where computation is performed) and a slow memory of infinite size. We assume that the input initially resides in slow memory and is too large to fit in fast memory. We define the *communication cost* of a sequential algorithm

The original version of this paper is entitled "Graph Expansion and Communication Costs of Fast Matrix Multiplication" and was first published in the *Proceedings of the 2011 ACM Symposium on Parallelism in Algorithms and Architectures* and also appeared in the December 2012 issue of the *Journal of the ACM*.

Figure 1. Strong-scaling performance comparison of parallel matrix multiplication algorithms on a Cray XT4. All data corresponds to a fixed dimension $n = 94080$. The x-axis represents the number of processors p on a log scale, and the y-axis measures effective performance, or $2n^3/(p \cdot \text{time})$. The new algorithm outperforms all other known algorithms and exceeds the peak performance of the machine with respect to the classical flop count. The new algorithm runs 24–184% faster than the best previous Strassen-based algorithm and 51–84% faster than the best classical algorithm for this problem size.

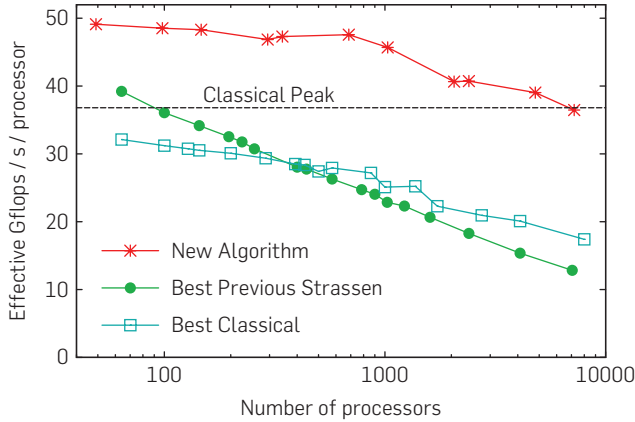
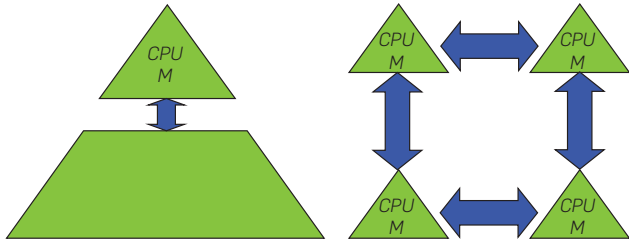


Figure 2. Sequential two-level (left) and parallel distributed-memory (right) models.



to be the total number of words transferred between the slow and fast memories.

In the parallel case (see Figure 2), we consider p processors, each with a local memory of size M , connected over a network. In this case, the communication cost is the number of words transferred between processors, counted along the critical path of the algorithm. That is, two words that are communicated simultaneously between separate pairs of processors are counted only once.

1.2. Classical matrix multiplication

To illustrate the effects of arithmetic reordering on communication and running time of a sequential computation, consider the problem of computing matrix multiplication $C = A \cdot B$, where the $(i, j)^{\text{th}}$ output element is computed by the classical formula $C_{ij} = \sum_k A_{ik} \cdot B_{kj}$. One “naive” ordering of the computation of the classical algorithm can be specified simply by three nested loops (see Algorithm 1). For matrices that are too large to fit in fast memory, this ordering requires the communication of at least one operand for each scalar

multiplication, resulting in a total communication cost of $\Theta(n^3)$. A natural question to ask is: can we do better?

Algorithm 1 Naive Classical Matrix Multiplication

```

1: for  $i = 1$  to  $n$  do
2:   for  $j = 1$  to  $n$  do
3:     for  $k = 1$  to  $n$  do
4:        $C_{ij} = C_{ij} + A_{ik} \cdot B_{kj}$ 

```

The answer is yes. We can reduce communication by using a “blocked” algorithm (see Algorithm 2). The idea is to partition A , B , and C into square blocks of size $b \times b$ so that three blocks can simultaneously fit in the fast memory. We use the notation $C[I, J]$ to refer to the $(I, J)^{\text{th}}$ $b \times b$ block of the C matrix. When $C[I, J]$, $A[I, K]$, and $B[K, J]$ are all in fast memory, then the inner loop of the algorithm (corresponding to (b^3) arithmetic operations) can be performed with no more communication.

Algorithm 2 Blocked Classical Matrix Multiplication

```

1: for  $I = 0$  to  $n/b$  do
2:   for  $J = 0$  to  $n/b$  do
3:     for  $K = 0$  to  $n/b$  do
4:        $C[I, J] = C[I, J] + A[I, K] \cdot B[K, J]$ 

```

If we pick the maximum block size of $b = \sqrt{M}/3$, this results in a total of $\Theta((n/\sqrt{M})^3)$ block operations, each requiring (M) words to be communicated. Hence, the total communication cost is $\Theta(n^3/\sqrt{M})$, a factor of $\Theta(\sqrt{M})$ better than that of the naive algorithm.

The typical performance difference between the naive and blocked algorithms on a sequential machine is an order of magnitude. With the blocked algorithm, attained performance is close to the peak capabilities of the machine. Again, the question arises: can we do better? Can we further reorder these computations to communicate less?

If we insist on performing the (n^3) arithmetic operations given by the classical formulation, the answer is no. Hong and Kung¹⁸ proved a communication cost lowerbound of $\Omega(n^3/\sqrt{M})$ for any reordering, showing that the blocked algorithm is communication-optimal. But this is not the end of the story: this communication optimality of the blocked algorithm assumes (n^3) arithmetic operations.

1.3. Strassen’s matrix multiplication

While the classical algorithms for matrix multiplication have already been optimized for reducing communication cost to the minimum possible, a completely different algorithmic approach for this problem is possible. Let us recall Strassen’s algorithm²⁴ (see Algorithm 3).

Strassen’s key idea is to multiply 2×2 matrices using seven scalar multiplies instead of eight. Because $n \times n$ matrices can be divided into quadrants, Strassen’s idea applies recursively. Each of the seven quadrant multiplications is computed recursively, and the computational cost of additions and subtractions of quadrants is (n^2) . Thus, the recurrence for the flop count is $F(n) = 7F(n/2) + (n^2)$ with base case $F(1) = 1$, which

yields $F(n) = (n^{\log_2 7})$, which is asymptotically less computation than the classical algorithm.

The main results presented in the following section expose a wonderful fact: not only does Strassen's algorithm require less computation than the classical algorithm, but it also requires less communication!

Algorithm 3 Strassen's Matrix Multiplication Algorithm

Input: $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ and $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \in \mathbb{R}^{n \times n}$

```

1: if  $n = 1$  then
2:    $C = A \cdot B$ 
3: else
4:    $M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$ 
5:    $M_2 = (A_{21} + A_{22}) \cdot B_{11}$ 
6:    $M_3 = A_{11} \cdot (B_{12} - B_{22})$ 
7:    $M_4 = A_{22} \cdot (B_{21} - B_{11})$ 
8:    $M_5 = (A_{11} + A_{12}) \cdot B_{22}$ 
9:    $M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$ 
10:   $M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$ 
11:   $C_{11} = M_1 + M_4 - M_5 + M_7$ 
12:   $C_{12} = M_3 + M_5$ 
13:   $C_{21} = M_2 + M_4$ 
14:   $C_{22} = M_1 - M_2 + M_3 + M_6$ 

```

Output: $A \cdot B = C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \in \mathbb{R}^{n \times n}$

2. COMMUNICATION LOWER BOUNDS

In this section, we state our main results: communication lower bounds for Strassen's matrix multiplication. The proof technique described in Section 3 allows us to state bounds in both sequential and parallel cases. As mentioned in the Section 1, the lower bounds are lower than the bounds for the classical algorithm.^{18, 20} In both sequential and parallel cases, there now exist communication-optimal algorithms that achieve the lower bounds.

2.1. Sequential case

We obtain the following lower bound:

THEOREM 1.¹⁰ *Consider Strassen's algorithm implemented on a sequential machine with fast memory of size M . Then for $M \leq n^2$, the communication cost of Strassen's algorithm is*

$$IO(n, M) = \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot M\right).$$

It holds for any implementation and any known variant of Strassen's algorithm that is based on performing 2×2 matrix multiplication with seven scalar multiplications. This includes Winograd's $O(n^{\log_2 7})$ variant that uses 15 additions instead of 18, which is the most commonly used fast matrix multiplication algorithm in practice.

This lower bound is tight, in that it is attained by the standard recursive sequential implementation of Strassen's algorithm. The recursive algorithm's communication cost

Table 1. Asymptotic communication cost lower bounds for sequential matrix multiplication, where n is the matrix dimension and M is the fast memory size. Note that although the expressions for classical and Strassen are similar, the proof techniques are quite different

	Classical	Strassen
Sequential lower bound ^{18, 10}	$\left(\frac{n}{\sqrt{M}}\right)^3 M$	$\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M$

is given by the recurrence $IO(n, M) \leq 7 \cdot IO\left(\frac{n}{2}, M\right) + O(n^2)$. The base case occurs when the input and output sub-matrices fit in the fast memory and the matrix multiplication can be performed with no further communication. This yields

$$IO(n, M) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot M\right)$$

for $M \leq n^2$, matching the lower bound stated in Theorem 1.

2.2. Parallel case

The proof technique of Theorem 1 extends to parallel machines, yielding

COROLLARY 2.¹⁰ *Consider Strassen's algorithm implemented on a parallel machine with p processors, each with a local memory of size M . Then for $M = O\left(\frac{n^2}{p^{2/\log_2 7}}\right)$, the communication cost of Strassen's algorithm is*

$$IO(n, p, M) = \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot \frac{M}{p}\right)$$

While Corollary 2 does not hold for all sizes of local memory (relative to the problem size and number of processors), the following memory-independent lower bound can be proved using similar techniques⁵ and holds for all local memory sizes, though it requires separate assumptions.

THEOREM 3.⁵ *Suppose a parallel algorithm performing Strassen's matrix multiplication load balances the computation. Then, the communication cost is*

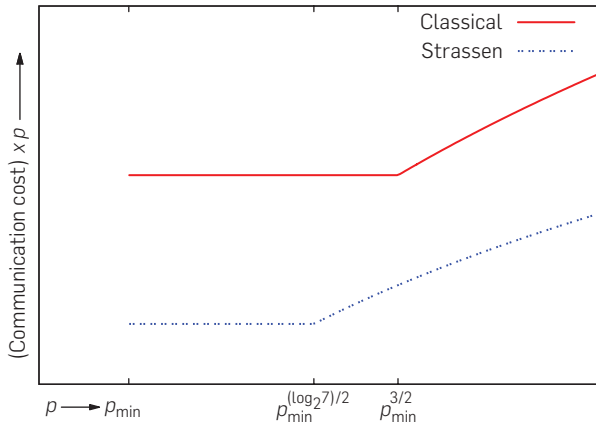
$$IO(n, p) = \Omega\left(\frac{n^2}{p^{2/\log_2 7}}\right).$$

Note that the bound in Corollary 2 dominates the one in Theorem 3 for $M = O(n^2/p^{2/\log_2 7})$. Thus, the tightest lower bound for parallel implementations of Strassen is the maximum of these two bounds. Table 2 and Figure 3, both adapted from Ballard et al.,⁵ illustrate the relationship between the two functions. Figure 3 in particular shows bounds on strong scaling: for a fixed dimension n , increasing the number of processors (each with local memory size M) within a limited range does not increase the total volume of communication. Thus, the communication cost along the critical path decreases linearly with p . This is because in this "perfect strong scaling range,"

Table 2. Asymptotic communication cost lower bounds for parallel matrix multiplication, where n is matrix dimension, M is local memory size, and p is the number of processors

	Classical	Strassen
Memory-dependent lower bound ^{20,10}	$\left(\frac{n}{\sqrt{M}}\right)^3 \frac{M}{p}$	$\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{p}$
Memory-independent lower bound ⁵	$\frac{n^2}{p^{2/3}}$	$\frac{n^2}{p^{2/\log_2 7}}$

Figure 3. Communication costs and strong scaling of matrix multiplication: classical vs. Strassen.⁵ The vertical axis corresponds to p times the communication cost, so horizontal lines correspond to perfect strong scaling. The quantity $p_{\min}$ is the minimum number of processors required to store the input and output matrices (i.e., $p_{\min} = 3n^2/M$ where n is the matrix dimension and M is the local memory size).



the dominant lower bound includes a p in the denominator; however, when the second bound begins to dominate, the denominator includes a $p^{2/3}$ rather than p , and increasing p leads to more communication volume. As shown in the figure, a similar phenomenon occurs for the classical algorithm, though with slightly different parameters.^{5,23}

The recent parallel algorithm for Strassen's matrix multiplication⁶ has communication cost

$$IO(n, p, M) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \cdot \frac{M}{p} + \frac{n^2}{p^{2/\log_2 7}}\right)$$

where p is the number of processors and M is the size of the local memory. Note that this matches the lower bounds of Corollary 2 and Theorem 3 above. A similar algorithm for Strassen's matrix multiplication in the BSP model is presented in McColl and Tiskin.²²

3. PROOF HIGHLIGHTS

The crux of the proof of Theorem 1 is based on estimating the edge expansion of the computation graph of Strassen's algorithm. We describe below how communication cost is closely related to the edge expansion properties of this graph. The graph has a recursive structure, and we use a combinatorial analysis of the expansion. The high-level argument is based on partitioning the computation in

segments, which we explain in Section 3.3. Let us first define two key concepts: computation graphs and edge expansion. See Ballard et al.¹⁰ for the full proof.

3.1. Computation graphs

The computation performed by an algorithm on a given input can be modeled as a computation directed acyclic graph (CDAG): we have a vertex for each input, intermediate, and output argument, and edges according to direct dependencies (e.g., for the binary arithmetic operation $x := y + z$, we have directed edges from vertices corresponding to operands y and z to the vertex corresponding to x).

In the sequential case, an implementation (or scheduling) determines the order of execution of the arithmetic operations, which respects the partial ordering of the CDAG. In the parallel case, an implementation determines which arithmetic operations are performed by which of the p processors as well as the ordering of local operations. This corresponds to partitioning the CDAG into p parts. Edges crossing between the various parts correspond to arguments that are in the possession of one processor but are needed by another processor and therefore relate to communication.

3.2. Edge expansion

Expansion is a graph-theoretic concept¹⁹ that relates a given subset of a graph to its boundary. If a graph has large expansion, then subsets of vertices will have relatively large boundaries. For example, a 2D grid where each vertex has north, south, east, and west neighbors has small expansion, whereas a complete graph has large expansion. While there are several variants of expansion metrics, we are interested in edge expansion of regular graphs, defined as follows: the edge expansion $h(G)$ of a d -regular undirected graph $G = (V, E)$ is

$$h(G) \equiv \min_{U \subseteq V, |U| \leq |V|/2} \frac{|E_G(U, V \setminus U)|}{d \cdot |U|} \quad (1)$$

where $E_G(A, B)$ is the set of edges connecting the disjoint vertex sets A and B .

Note that CDAGs are typically not regular. If a graph $G = (V, E)$ is not regular but has a bounded maximal degree d , then we can add ($< d$) loops to vertices of degree $< d$, obtaining a regular graph G . We use the convention that a loop adds 1 to the degree of a vertex. Note that for any $S \subseteq V$, we have $|E_G(S, V \setminus S)| = |E_G(S, V \setminus S)|$, as none of the added loops contributes to the edge expansion of G .

For many graphs, small sets have larger expansion than larger sets. Let $h_s(G)$ denote the edge expansion of G for sets of size at most s :

$$h_s(G) \equiv \min_{U \subseteq V, |U| \leq s} \frac{|E_G(U, V \setminus U)|}{d \cdot |U|}. \quad (2)$$

For many interesting graph families (including Strassen's CDAG), $h_s(G)$ does not depend on $|V(G)|$ when s is fixed, although it may decrease when s increases.

3.3. The partition argument

The high-level lower bound argument is based on partitioning the execution of an algorithm's implementation into segments. Let O be any total ordering of the vertices that

respects the partial ordering of the CDAG G , that is, all the edges are directed upwards in the total order. This total ordering can be thought of as the actual order in which the computations are performed. Let P be any partition of V into segments $S_1, S_2, \dots$, so that a segment S_i is a subset of the vertices that are contiguous in the total ordering O .

Let S be some segment, and define R_S and W_S to be the set of read and write operands, respectively (see Figure 4), namely, R_S is the set of vertices outside S that have an edge going into S , and W_S is the set of vertices in S that have an edge going outside of S . Recall that M is the size of the fast memory. Then, the total communication cost due to reads of operands in S is at least $|R_S| - M$, as at most M of the needed $|R_S|$ operands are already in fast memory when the segment starts. Similarly, S causes at least $|W_S| - M$ actual write operations, as at most M of the operands needed by other segments are left in the fast memory when the segment ends. The total communication cost is therefore bounded below by

$$IO \geq \max_P \sum_{S \in P} (|R_S| + |W_S| - 2M). \quad (3)$$

3.4. Edge expansion and communication

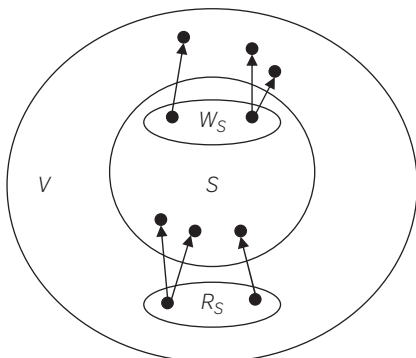
Consider a segment S and its read and write operands R_S and W_S (see Figure 4). If the graph G containing S has $h(G)$ edge expansion, maximum degree d and at least $2|S|$ vertices, then (using the definition of $h(G)$), we have

$$\text{CLAIM 4. } |R_S| + |W_S| \geq h(G) \cdot |S|.$$

Combining this with (3) and choosing to partition V into $|V|/s$ segments of equal size s , we obtain $IO \geq \max_s (|V|/s) \cdot (h(G) \cdot s - 2M) = (|V| \cdot h(G) - 2M) \cdot (|V|/s)$. In many cases, $h(G)$ is too small to attain the desired communication cost lower bound. Typically, $h(G)$ is a decreasing function of $|V(G)|$; that is, the edge expansion deteriorates with the increase of the input size and number of arithmetic operations of the corresponding algorithm (this is the case with Strassen's algorithm). In such cases, it is better to consider the expansion of G on small sets only: $IO \geq \max_s (|V|/s) \cdot (h_s(G) \cdot s - 2M)$. Choosing the minimal s so that

$$h_s(G) \cdot s \geq 3M \quad (4)$$

Figure 4. A subset (segment) S and its corresponding read operands R_S and write operands W_S .



we obtain

$$IO \geq \frac{|V|}{s} \cdot M. \quad (5)$$

The existence of a value $s \leq |V|/2$ that satisfies condition (4) is not always guaranteed. In Ballard et al.,¹⁰ we confirm the existence of such s for Strassen's CDAG for sufficiently large $|V|$.

4. STRASSEN'S CDAG

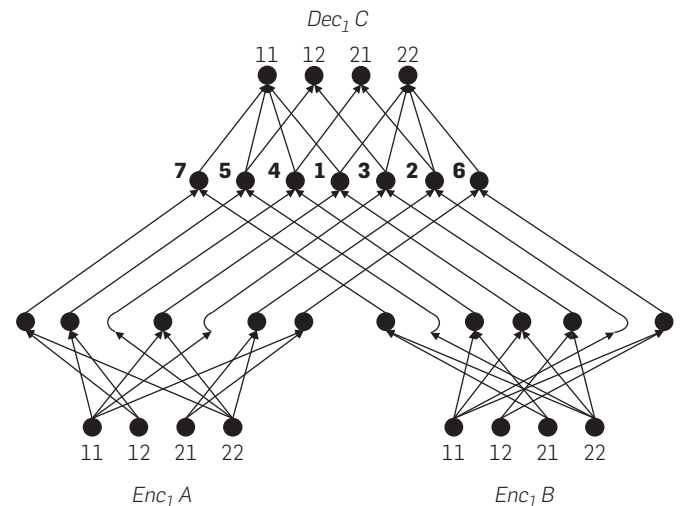
Recall Strassen's algorithm for matrix multiplication and consider its computation graph. If we let H_i be the computation graph of Strassen's algorithm for recursion of depth i , then $H_{\log_2 n}$ corresponds to the computation for input matrices of size $n \times n$. Let us first consider H_1 as shown in Figure 5, which corresponds to multiplying 2×2 matrices. Each of A and B is "encoded" into seven pairs of multiplication inputs, and vertices corresponding to the outputs of the multiplications are then "decoded" to compute the output matrix C .

The general computation graph $H_{\log_2 n}$ has similar structure:

- Encode A : generate weighted sums of elements of A
- Encode B : generate weighted sums of elements of B
- Multiply the encodings of A and B element-wise
- Decode C : take weighted sums of the products

Denote by $Enc_{\log_2 n} A$ the part of $H_{\log_2 n}$ that corresponds to the encoding of matrix A . Similarly, $Enc_{\log_2 n} B$, and $Dec_{\log_2 n} C$ correspond to the parts of $H_{\log_2 n}$ that compute the encoding of B and the decoding of C , respectively. Figure 6 shows a high level picture of $H_{\log_2 n}$. In the next section, we provide a more detailed description of the CDAG.

Figure 5. Computation graph of Strassen's algorithm for multiplying 2×2 matrices (H_1). The encodings of A and B correspond to the additions and subtractions in lines 4–10 of Algorithm 3, and the decoding of the seven multiplications to compute C corresponds to lines 11–14. A vertex labeled with two indices ij corresponds to the (i, j) th entry of a matrix and a vertex labeled with one index k corresponds to the k th intermediate multiplication.



4.1. Recursive construction

We construct the computation graph H_{i+1} by constructing $Dec_{i+1}C$ from Dec_iC and Dec_1C , similarly constructing $Enc_{i+1}A$ and $Enc_{i+1}B$, and then composing the three parts together. Here is the main idea for recursively constructing $Dec_{i+1}C$, which is illustrated in Figure 7.

- Replicate Dec_1C 7^i times.
- Replicate Dec_1C 4 times.
- Identify the $4 \cdot 7^i$ output vertices of the copies of Dec_1C with the $4 \cdot 7^i$ input vertices of the copies of Dec_1C :
 - Recall that each Dec_1C has four output vertices.
 - The set of each first output vertex of the 7^i Dec_1C graphs is identified with the set of 7^i input vertices of the first copy of Dec_1C .
 - The set of each second output vertex of the 7^i Dec_1C graphs is identified with the set of 7^i input vertices of the second copy of Dec_1C , and so on.
 - We make sure that the j^{th} input vertex of a copy of Dec_1C is identified with an output vertex of the j^{th} copy of Dec_1C .

After constructing $Enc_{i+1}A$ and $Enc_{i+1}B$ in a similar manner, we obtain H_{i+1} by connecting edges from the k^{th} output vertices of $Enc_{i+1}A$ and $Enc_{i+1}B$ to the k^{th} input vertex of $Dec_{i+1}C$, which corresponds to the element-wise scalar multiplications.

Figure 6. High-level view of Strassen's CDAG for $n \times n$ matrices. The graph is composed of two encoding subgraphs and one decoding subgraph; connections between the subgraphs are not shown.

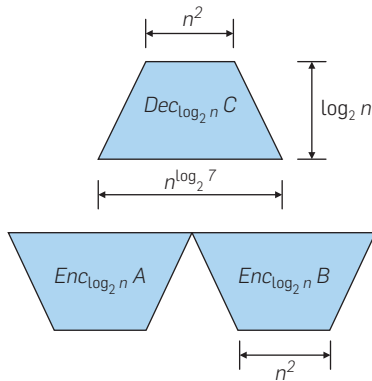
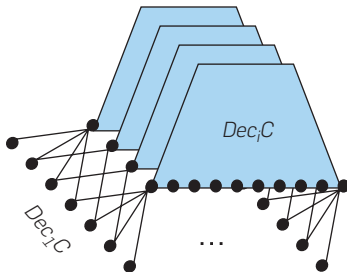


Figure 7. Illustration of the recursive construction of the decoding subgraph. To construct $Dec_{i+1}C$, Dec_iC is replicated 4 times and Dec_1C is replicated 7^i times, and appropriate vertices are identified.



4.2. Strassen's edge expansion

Given the construction of the CDAG for Strassen's algorithm, we now state our main lemma on the edge expansion of the decoding graph. The proof technique resembles the expander analysis in Alon et al.² For the complete proof, see Ballard et al.¹⁰

LEMMA 5. (MAIN LEMMA) *The edge expansion of Dec_kC is*

$$h(Dec_kC) = \Omega\left(\left(\frac{4}{7}\right)^k\right).$$

By another argument (proof in Ballard et al.¹⁰), we obtain that

$$h_s(Dec_{\log_2 n}C) \geq h(Dec_kC),$$

where $s = (7^k)$. Choosing $s = (M^{(\log_2 7)/2})$, we satisfy Inequality 4 and obtain Inequality 5 (for sufficiently large $|V|$). This gives Theorem 1.

5. EXTENSIONS

In this paper, we focus on lower bounds for Strassen's matrix multiplication algorithm on two machine models. However, the design space of improving fundamental algorithms via communication minimization is much larger. It includes proving lower bounds and developing optimal algorithms; using classical methods as well as fast algorithms like Strassen's; performing matrix multiplication, other matrix algorithms, and more general computations; minimizing time and/or energy; using minimal memory or trading off extra memory for less communication; and using hierarchical, homogeneous, or heterogeneous sequential and parallel models. In this section, we discuss a subset of these extensions; see Ballard et al.^{9,10} and the references therein for more details.

5.1. Lower bounds

The proof technique described in Section 3 is not specific to Strassen's algorithm and can be applied more widely. The partition argument is used for classical algorithms in numerical linear algebra^{8, 20} where a geometric inequality specifies the per-segment communication cost rather than edge expansion. Further, the edge expansion technique applies to *Strassen-like* algorithms that also multiply square matrices with $o(n^3)$ arithmetic operations, to other fast algorithms for rectangular matrix multiplication, and to other matrix computations.

Strassen-like algorithms. Strassen-like algorithms are recursive matrix multiplication algorithms based on a scheme for multiplying $k \times k$ matrices using q scalar multiplications for some k and $q < k^3$ (so that the algorithm performs $O(n^{\omega_0})$ flops where $\omega_0 = \log_k q$.) For the latest bounds on the arithmetic complexity of matrix multiplication and references to previous bounds, see Williams.²⁵ For our lower bound proof to apply, we require another technical criterion for Strassen-like algorithms: the decoding graph must be connected. This class of algorithms includes many (but not all) fast matrix multiplications. For details and examples, see Ballard et al.^{7,10}

For Strassen-like algorithms, the statements of the communication lower bounds have the same form as Theorem 1, Corollary 2, and Theorem 3: replace $\log_2 7$ with ω_0 everywhere it appears! The proof technique follows that for Strassen's

algorithm. While the bounds for the classical algorithm have the same form, replacing $\log_2 7$ with 3, the proof techniques are quite different.^{18,20}

Fast rectangular matrix multiplication. Many fast algorithms have been devised for multiplication of rectangular matrices (see Ballard et al.⁷ for a detailed list). A fast algorithm for multiplying $m \times k$ and $k \times r$ matrices in $q < mkr$ scalar multiplications can be applied recursively to multiply $m^t \times k^t$ and $k^t \times r^t$ matrices in $O(q^t)$ flops. For such algorithms, the CDAG has very similar structure to Strassen and Strassen-like algorithms for square multiplication in that it is composed of two encoding graphs and one decoding graph. Assuming that the decoding graph is connected, the proofs of Theorem 1 and Lemma 5 apply where we plug in mr and q for 4 and 7. In this case, we obtain a result analogous to Theorem 1 which states that the communication cost of such an algorithm is given by $\Omega(q^t/M^{\log_{mr} q-1})$. If the output matrix is the largest of the three matrices (i.e., $k < m$ and $k < r$), then this lower bound is attained by the natural recursive algorithm and is therefore tight. The lower bound extends to the parallel case as well, analogous to Corollary 2, and can be attained using the algorithmic technique of Ballard et al.⁶

The rest of numerical linear algebra. Fast matrix multiplication, algorithms are basic building blocks in many fast algorithms in linear algebra, such as algorithms for LU, QR, and eigenvalue and singular value decompositions.¹³ Therefore, communication cost lower bounds for these algorithms can be derived from our lower bounds for fast matrix multiplication algorithms. For example, a lower bound on LU (or QR, etc.) follows when the fast matrix multiplication algorithm is called by the LU algorithm on sufficiently large sub-matrices. This is the case in the algorithms of Demmel et al.,¹³ and we can then deduce matching lower and upper bounds.¹⁰

Nested loops computation. Nearly all of the arguments for proving communication lower bounds are based on establishing a relationship between a given set of data and the amount of useful computation that can be done with that data, a so-called “surface-to-volume” ratio. For example, Hong and Kung¹⁸ use an analysis of dominator sets and minimal sets of CDAGs to establish such ratios. The Loomis–Whitney geometric inequality is applied for this purpose to matrix computations specified by three nested loops in Ballard et al.⁸ and Irony et al.²⁰ Recently, Christ et al.¹² have extended this analysis using a generalization of the Loomis–Whitney inequality, known as the Hölder–Brascamp–Lieb inequality, to prove lower bounds for computations that are specified by an arbitrary set of nested loops that linearly access arrays and meet certain other criteria.

5.2. Algorithms

The main motivation for pursuing communication lower bounds is to provide targets for algorithmic performance. Indeed, the conjecture and proof of Theorem 1 and Corollary 2, as well as the existence of an optimal algorithm in the sequential case, were the main motivations for improving the parallel implementations of Strassen’s algorithm. Not only were we able to devise an optimal algorithm, but we were also able to show with an

implementation for distributed-memory machines that it performs much faster in practice.^{6,21}

Communication avoiding parallel Strassen. In Section 2.2, we stated the communication cost of a new parallel algorithm for Strassen’s matrix multiplication, matching the asymptotic lower bound. The details of the algorithm appear in Ballard et al.,⁶ and more extensive implementation details and performance data are given in Lipshitz et al.²¹ We show that the new algorithm is more efficient than any other parallel matrix multiplication algorithm of which we are aware, including those that are based on the classical algorithm and those that are based on previous parallelizations of Strassen’s algorithm.

Figure 1 shows performance on a Cray XT4. For results on other machines, see Lipshitz et al.²¹ For example, running on a Cray XE6 with up to 10,000 cores, for a problem of dimension $n = 131712$, our new algorithm attains performance as high as 30% above the peak for classical matrix multiplication, 83% above the best classical implementation, and 75% above the best previous implementation of Strassen’s algorithm. Even for a small problem of dimension $n = 4704$, it attains performance 66% higher than the best classical implementation.

Further applications. The key algorithmic idea in our parallel implementation of Strassen’s algorithm is a careful parallel traversal of the recursion tree. This idea works for many other recursive algorithms where the subproblems do not have interdependencies (and it also works in some cases where dependencies exist). For example, classical rectangular matrix multiplication¹⁴ and sparse matrix–matrix multiplication⁴ can be parallelized in this way to obtain communication optimality.

The same techniques can be utilized to save energy at the algorithmic level (since communication consumes more energy than computation) as well as to obtain lower bounds on energy requirements.¹⁵

In summary, we believe this work flow of theoretical lower bounds to algorithmic development to efficient implementations is very effective: by considering fundamental computations at an algorithmic level, significant improvements in many applications are possible.

Acknowledgments

We would like to thank Benjamin Lipshitz for his work on many of these ideas and for useful discussions during the writing of this paper.

This work is supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227); additional support from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung is acknowledged. This research is supported by U.S. Department of Energy grants under Grant Numbers DE-SC0003959, DE-SC0004938, DE-SC0005136, DE-SC0008700, AC02-05CH11231, and DE-FC02-06-ER25786, and DARPA grant HR0011-12-2-0016. The research is also supported by the Sofja Kovalevskaja programme of Alexander von Humboldt Foundation and by the National Science Foundation under agreement DMS-0635607, and by ERC Starting Grant Number 239985. 

References

- Agarwal, R.C., Balle, S.M., Gustavson, F.G., Joshi, M., Palkar, P. A three-dimensional approach to parallel matrix multiplication. *IBM J. Res. Dev.* 39, 5 (1995), 575–582.
- Alon, N., Schwartz, O., Shapira, A. An elementary construction of constant-degree expanders. *Combinator. Probab. Comput.* 17, 3 (2008), 319–327.
- Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D. LAPACK's User's Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992. Also available from <http://www.netlib.org/lapack/>.
- Ballard, G., Buluç, A., Demmel, J., Grigori, L., Lipshitz, B., Schwartz, O., Toledo, S. Communication Optimal Parallel Multiplication of Sparse Random Matrices. In *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures*, (2013), ACM, New York, NY, USA.
- Ballard, G., Demmel, J., Holtz, O., Lipshitz, B., Schwartz, O. Brief announcement: Strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, (2012), ACM, New York, NY, USA, 77–79.
- Ballard, G., Demmel, J., Holtz, O., Lipshitz, B., Schwartz, O. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12 (2012), ACM, New York, NY, USA, 193–204.
- Ballard, G., Demmel, J., Holtz, O., Lipshitz, B., Schwartz, O. Graph expansion analysis for communication costs of fast rectangular matrix multiplication. In *Design and Analysis of Algorithms*. G. Even and D. Rawitz, eds., Volume 7659 of *Lecture Notes in Computer Science* (2012), Springer, Berlin-Heidelberg, 13–36.
- Ballard, G., Demmel, J., Holtz, O., Schwartz, O. Graph expansion and communication costs of fast matrix multiplication. In *Proceedings of the 23rd Annual ACM Symposium on Parallel Algorithms and Architectures* (2011), ACM, New York, NY, USA, 1–12.
- Ballard, G., Demmel, J., Holtz, O., Schwartz, O. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.* 32, 3 (2011), 866–901.
- Ballard, G., Demmel, J., Holtz, O., Schwartz, O. Graph expansion and communication costs of fast matrix multiplication. *J. ACM* (Dec. 2012) 59, 6, 32:1–32:23.
- Cannon, L. A cellular computer to implement the Kalman filter algorithm. PhD thesis, Montana State University, Bozeman, MN (1969).
- Christ, M., Demmel, J., Knight, N., Scanlon, T., Yelick, K. Communication lower bounds and optimal algorithms for programs that reference arrays – Part I. Manuscript, 2013.
- Demmel, J., Dumitriu, I., Holtz, O. Fast linear algebra is stable. *Numer. Math.* 108, 1 (2007), 59–91.
- Demmel, J., Elichu, D., Fox, A., Kamil, S., Lipshitz, B., Schwartz, O., Spillinger, O. Communication-optimal parallel recursive rectangular matrix multiplication. In *Proceedings of the 27th IEEE International Parallel & Distributed Processing Symposium (IPDPS)* (2013), IEEE.
- Demmel, J., Gearhart, A., Lipshitz, B., Schwartz, O. Perfect strong scaling using no additional energy. In *Proceedings of the 27th IEEE International Parallel & Distributed Processing Symposium, IPDPS '13* (2013), IEEE.
- Fuller, S.H., Millett, L.I., eds. *The Future of Computing Performance: Game Over or Next Level?* The National Academies Press, Washington, D.C., 2011, 200 pages, <http://www.nap.edu>.
- Graham, S.L., Snir, M., Patterson, C.A., eds. *Getting up to Speed: The Future of Supercomputing*. Report of National Research Council of the National Academies Sciences. The National Academies Press, Washington, D.C., 2004, 289 pages, <http://www.nap.edu>.
- Hong, J.W., Kung, H.T. I/O complexity: The red-blue pebble game. In *STOC '81: Proceedings of the 13th annual ACM Symposium on Theory of Computing* (1981), ACM, New York, NY, USA, 326–333.
- Hoory, S., Linial, N., Wigderson, A. Expander graphs and their applications. *Bull. AMS* 43(4), (2006), 439–561.
- Irony, D., Toledo, S., Tiskin, A. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.* 64, 9, (2004), 1017–1026.
- Lipshitz, B., Ballard, G., Demmel, J., Schwartz, O. Communication-avoiding parallel Strassen: Implementation and performance. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, (2012), IEEE Computer Society Press, Los Alamitos, CA, USA, 101:1–101:11.
- McColl, W.F., Tiskin, A. Memory-efficient matrix multiplication in the BSP model. *Algorithmica* 24 (1999), 287–297.
- Solomonik, E., Demmel, J. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *Proceedings of the 17th International European Conference on Parallel and Distributed Computing* (2011), Springer.
- Strassen, V. Gaussian elimination is not optimal. *Numer. Math.* 13 (1969), 354–356.
- Williams, V.V. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing, STOC '12* (2012), ACM, New York, NY, USA, 887–898.

Grey Ballard (ballard@eecs.berkeley.edu), Electrical Engineering and Computer Science Department, University of California, Berkeley, CA.

James Demmel (demmel@cs.berkeley.edu), Department of Mathematics and Computer Science Division, University of California, Berkeley, CA.

Olga Holtz (holtz@math.berkeley.edu), Department of Mathematics, University of California, Berkeley, CA, and Institut für Mathematik, Technische Universität Berlin, Germany.

Oded Schwartz (odedsc@eecs.berkeley.edu), Electrical Engineering and Computer Science Department, University of California, Berkeley, CA.

© 2014 ACM 0001-0782/14/02 \$15.00

World-Renowned Journals from ACM

ACM publishes over 50 magazines and journals that cover an array of established as well as emerging areas of the computing field. IT professionals worldwide depend on ACM's publications to keep them abreast of the latest technological developments and industry news in a timely, comprehensive manner of the highest quality and integrity. For a complete listing of ACM's leading magazines & journals, including our renowned Transaction Series, please visit the ACM publications homepage: www.acm.org/pubs.

ACM Transactions on Interactive Intelligent Systems



ACM Transactions on Interactive Intelligent Systems (TIIS). This quarterly journal publishes papers on research encompassing the design, realization, or evaluation of interactive systems incorporating some form of machine intelligence.

ACM Transactions on Computation Theory



ACM Transactions on Computation Theory (ToCT). This quarterly peer-reviewed journal has an emphasis on computational complexity, foundations of cryptography and other computation-based topics in theoretical computer science.

PLEASE CONTACT ACM MEMBER SERVICES TO PLACE AN ORDER

Phone: 1.800.342.6626 (U.S. and Canada)
+1.212.626.0500 (Global)
Fax: +1.212.944.1318
(Hours: 8:30am–4:30pm, Eastern Time)
Email: acmhlp@acm.org
Mail: ACM Member Services
General Post Office
PO Box 30777
New York, NY 10087-0777 USA



Association for
Computing Machinery

Advancing Computing as a Science & Profession

www.acm.org/pubs

Boston College

Assistant Professor, Computer Science

The Computer Science Department of Boston College invites applications for a tenure-track Assistant Professorship beginning September, 2014. Applications from all areas of Computer Science will be considered. Applicants should have a Ph.D. in Computer Science or related discipline, a strong research record, and a commitment to undergraduate teaching.

We will begin reviewing applications on December 1, 2013, and will continue considering applications until the position is filled. Additional information about the department and the position is available at www.cs.bc.edu. **Submit applications online at apply.interfolio.com/22805.**

Dartmouth College

Department of Computer Science

Assistant Professor of Computer Science: Computer Graphics/Digital Arts

The Dartmouth College Department of Computer Science invites applications for a tenure-track faculty position at the level of assistant professor. We seek candidates who will be excellent researchers and teachers in the areas of computer graphics and/or digital arts, although outstanding candidates in any area will be considered. We particularly seek candidates who will be integral members of the Digital Arts program and help lead, initiate, and participate in collaborative research projects both within Computer Science and involving other Dartmouth researchers, including those in other Arts & Sciences departments, Dartmouth's Geisel School of Medicine, and Thayer School of Engineering.

The department is home to 17 tenured and tenure-track faculty members and two research faculty members. Research areas of the department encompass the areas of systems, security, vision, digital arts, algorithms, theory, robotics, and computational biology. The Computer Science department is in the School of Arts & Sciences, and it has strong Ph.D. and M.S. programs and outstanding undergraduate majors. Digital Arts at Dartmouth is an interdisciplinary program housed in the Computer Science department, working with several other departments, including Studio Art, Theater, and Film and Media Studies. The department is affiliated with Dartmouth's M.D.-Ph.D. program and has strong collaborations with Dartmouth's other schools.

Dartmouth College, a member of the Ivy League, is located in Hanover, New Hampshire (on the Vermont border). Dartmouth has a beautiful, historic campus, located in a scenic area on the Connecticut River. Recreational opportunities abound in all four seasons.

With an even distribution of male and female students and over one third of the undergraduate student population members of minority groups, Dartmouth is committed to diversity and encourages applications from women and minorities.

To create an atmosphere supportive of research, Dartmouth offers new faculty members grants for research-related expenses, a quarter of sabbatical leave for each three academic years in residence, and flexible scheduling of teaching responsibilities.

Applicants are invited to submit application materials via Interfolio at <http://apply.interfolio.com/23489>. Upload a CV, research statement, and teaching statement, and request at least four references to upload letters of recommendation, at least one of which should comment on teaching. Email facsearch14@cs.dartmouth.edu with any questions.

Application review will begin November 1, 2013, and continue until the position is filled.

Dartmouth College

Department of Computer Science

Assistant Professor of Computer Science: Machine Learning

The Dartmouth College Department of Computer Science invites applications for a tenure-track faculty position at the level of assistant professor. We seek candidates who will be excellent researchers and teachers in the area of machine learning, although outstanding candidates in any area will be considered. We particularly seek candidates who will help lead, initiate, and participate in collaborative research projects both within Computer Science and involving other Dartmouth researchers, including those in other Arts & Sciences departments, Dartmouth's Geisel School of Medicine, Thayer School of Engineering, and Tuck School of Business.

The department is home to 17 tenured and tenure-track faculty members and two research faculty members. Research areas of the department encompass the areas of systems, security, vision, digital arts, algorithms, theory, robotics, and computational biology. The Computer Science department is in the School of Arts & Sciences, and it has strong Ph.D. and M.S. programs and outstanding undergraduate majors. The department is affiliated with Dartmouth's M.D.-Ph.D. program and has strong collaborations with Dartmouth's other schools.

Dartmouth College, a member of the Ivy League, is located in Hanover, New Hampshire (on the Vermont border). Dartmouth has a beautiful, historic campus, located in a scenic area on the Connecticut River. Recreational opportunities abound in all four seasons.

With an even distribution of male and female students and over one third of the undergraduate student population members of

minority groups, Dartmouth is committed to diversity and encourages applications from women and minorities.

To create an atmosphere supportive of research, Dartmouth offers new faculty members grants for research-related expenses, a quarter of sabbatical leave for each three academic years in residence, and flexible scheduling of teaching responsibilities.

Applicants are invited to submit application materials via Interfolio at <http://apply.interfolio.com/23502>. Upload a CV, research statement, and teaching statement, and request at least four references to upload letters of recommendation, at least one of which should comment on teaching. Email facsearch14@cs.dartmouth.edu with any questions.

Application review will begin December 15, 2013, and continue until the position is filled.

Dartmouth College

Department of Computer Science

Assistant Professor of Computer Science: Theory/Algorithms

The Dartmouth College Department of Computer Science invites applications for a tenure-track faculty position at the level of assistant professor. We seek candidates who will be excellent researchers and teachers in the area of theoretical computer science, including algorithms, although outstanding candidates in any area will be considered. We particularly seek candidates who will help lead, initiate, and participate in collaborative research projects both within Computer Science and involving other Dartmouth researchers, including those in other Arts & Sciences departments, Dartmouth's Geisel School of Medicine, Thayer School of Engineering, and Tuck School of Business.

The department is home to 17 tenured and tenure-track faculty members and two research faculty members. Research areas of the department encompass the areas of systems, security, vision, digital arts, algorithms, theory, robotics, and computational biology. The Computer Science department is in the School of Arts & Sciences, and it has strong Ph.D. and M.S. programs and outstanding undergraduate majors. The department is affiliated with Dartmouth's M.D.-Ph.D. program and has strong collaborations with Dartmouth's other schools.

Dartmouth College, a member of the Ivy League, is located in Hanover, New Hampshire (on the Vermont border). Dartmouth has a beautiful, historic campus, located in a scenic area on the Connecticut River. Recreational opportunities abound in all four seasons.

With an even distribution of male and female students and over one third of the undergraduate student population members of minority groups, Dartmouth is committed to diversity and encourages applications from women and minorities.

To create an atmosphere supportive of research, Dartmouth offers new faculty members grants for research-related expenses, a quarter of sabbatical leave for each three academic years in residence, and flexible scheduling of teaching responsibilities.

Applicants are invited to submit application materials via Interfolio at <http://apply.interfolio.com/23503>. Upload a CV, research statement, and teaching statement, and request at least four references to upload letters of recommendation, at least one of which should comment on teaching. Email facsearch14@cs.dartmouth.edu with any questions.

Application review will begin December 15 2013, and continue until the position is filled.

Harvard School of Engineering and Applied Sciences Tenure-Track Positions in Computer Science

The Harvard School of Engineering and Applied Sciences (SEAS) seeks applicants for positions at the tenure-track level in Computer Science, with an expected start date of July 1, 2014.

This is a broad faculty search and we welcome outstanding applicants in all areas of computer science, including applicants whose research and interests connect to such areas as engineering, health and medicine, or the social sciences. Of particular interest are candidates with a focus on data science, with research at the intersection of computer science, applied mathematics, statistics, and computational science.

The Computer Science program at Harvard University benefits from outstanding undergraduate and graduate students, an excellent location, significant industrial collaboration, and substantial support from the School of Engineering and Applied Sciences. Information about Harvard's current faculty, research, and educational programs in computer science is available at <http://www.seas.harvard.edu/computer-science>. The associated Institute for Applied Computational Science (<http://iacs.seas.harvard.edu>) fosters connections among computer science, applied math, data science, and various domain sciences at Harvard through its graduate program and events.

Candidates are required to have a doctorate or terminal degree by the expected start date. We seek candidates who have an outstanding research record and a strong commitment to undergraduate teaching and graduate training.

Required application documents include a cover letter, cv, a statement of research interests, a teaching statement, and up to three representative papers. Candidates are also required to submit the names and contact information for at least three to five references (three letters of recommendation are required), and the application is complete only when three letters have been submitted. We encourage candidates to apply by January 1, 2014, but will continue to review applications until the positions are filled. Applicants will apply on-line at <http://academicpositions.harvard.edu/postings/5199>.

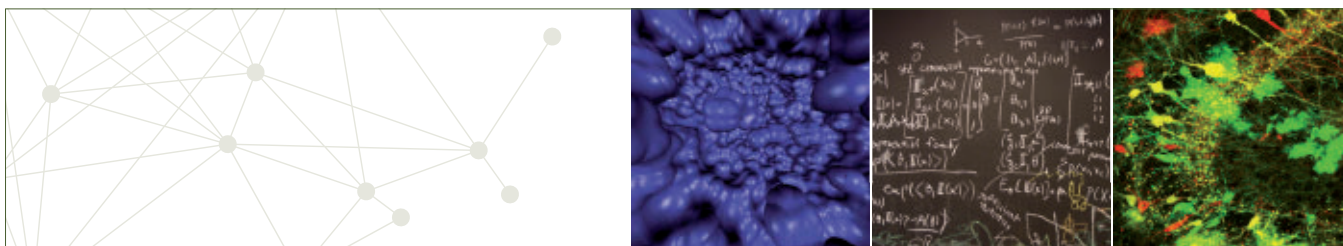
Harvard is an Equal Opportunity/Affirmative Action employer. Applications from women and minority candidates are strongly encouraged.

Max Planck Institute for Software Systems Senior Faculty Position in Software Systems

Applications are invited for a senior faculty position in the Max Planck Institute for Software Systems (MPI-SWS). The position is that of a director and scientific member of the Max Planck Society, and is comparable to an endowed chair position at a leading university. Directors lead their individual research groups, and also provide strategic direction for the institute, mentor junior faculty, and take turn in chairing the faculty. A successful candidate is an internationally recognized leader in the research community, and pursues a compelling and far-reaching research vision.

All areas related to the study, design, and engineering of software systems are considered. These areas include, but are not limited to, security and privacy, embedded and mobile systems, social computing, large-scale data management, programming languages and systems, software verification and analysis, parallel and distributed systems, storage systems, and networking. Preference will be given to candidates whose research complements existing strengths.

MPI-SWS, founded in 2005, is part of a network of 82 Max Planck Institutes, Germany's premier basic research facilities. MPIs have an established record of world-class, foundational research in the fields of medicine, biology, chemistry, physics, technology and humanities. Since 1948, MPI researchers have won 17 Nobel prizes. MPI-SWS aspires to meet the



ISTFELLOW: Call for Postdoctoral Fellows

Are you a talented, dynamic, and motivated scientist looking for an opportunity to conduct research in the fields of **BIOLOGY, COMPUTER SCIENCE, MATHEMATICS, PHYSICS, or NEUROSCIENCE** at a young, thriving institution that fosters scientific excellence and interdisciplinary collaboration?

Apply to the ISTFellow program. Deadlines March 15 and September 15

www.ist.ac.at/istfellow



ISTFELLOW is partially funded
by the European Union



highest standards of excellence and international recognition with its research in software systems.

To this end, the institute offers a unique environment that combines the best aspects of a university department and a research laboratory:

a) Faculty independently lead a team of graduate students and post-docs. They have full academic freedom and publish their research results freely. Substantial base funding complements third-party funds.

b) Faculty supervise doctoral theses, and have the opportunity to teach graduate and undergraduate courses.

c) Faculty are provided with outstanding technical and administrative support facilities as well as internationally competitive compensation packages.

MPI-SWS currently has 10 tenured and tenure-track faculty and 50 doctoral and post-doctoral researchers. The institute is funded to support 17 faculty and up to 100 doctoral and post-doctoral positions. Additional growth through outside funding is expected. We maintain an open, international and diverse work environment and seek applications from outstanding researchers regardless of national origin or citizenship. The working language is English.

The institute is located in Kaiserslautern and Saarbrücken, in the tri-border area of Germany, France and Luxembourg. The area offers a high standard of living, beautiful surroundings and easy access to major metropolitan areas in the center of Europe, as well as a stimulating, competitive and collaborative work environment. In immediate proximity are the MPI for Informatics, Saarland University, the Technical University of Kaiserslautern, the German Center for Artificial Intelligence (DFKI), and the Fraunhofer Institutes for Experimental Software Engineering and for Industrial Mathematics.

Qualified candidates are invited to send a CV and cover letter to rupak@mpi-sws.org. The review of applications will begin on Feb 1, 2014; applications will continue to be accepted until the position is filled.

The Max Planck Society is committed to increasing the representation of women and individuals with physical disabilities in Computer Science. We particularly encourage such individuals to apply.

National Taiwan University Professor-Associate Professor- Assistant Professor

The Department of Computer Science at National Taiwan Univ. has faculty openings at all ranks beginning in August 2014. Highly qualified candidates in all areas of computer science are invited to apply. A Ph.D. or its equivalent is required. Applicants are expected to conduct outstanding research and be committed to teaching. Candidates should send curriculum vitae, statements of research/teaching, three letters of reference, and supporting materials before February 28, 2014, to Prof. Chih-Jen Lin, Department of Computer Science, National Taiwan Univ., No 1, Sec 4, Roosevelt Rd., Taipei 106, Taiwan. Reference letters can be sent to faculty_search@csie.ntu.edu.tw but use regular mails for other materials. An early submission is strongly encouraged.

Purdue University Tenure-Track/Tenured Faculty Positions

The Department of Computer Science at Purdue University is entering a phase of sustained expansion. Applications for tenure-track and tenured positions at the Assistant, Associate and Full Professor levels beginning August 2014 are being solicited. Outstanding candidates in all areas will be considered.

The Department of Computer Science offers a stimulating and nurturing academic environment with active research programs in all areas of our discipline. Information about the department and a description of open positions are available at <http://www.cs.purdue.edu>.

Applicants should hold a PhD in Computer Science, or related discipline, be committed to excellence in teaching, and have demonstrated excellence in research. Successful candidates will be expected to teach courses in computer science, conduct research in their field of expertise, and participate in other department and university activities. Salary and benefits are competitive. Applicants are strongly encouraged to apply online at <https://hiring.science.purdue.edu>. Alternatively, hardcopy applications can be sent to: Faculty Search Chair, Department of Computer Science, 305 N. University Street, Purdue University, West Lafayette, IN 47907. Review of applications will begin in fall 2013, and will continue until positions are filled. A background check will be required for employment. Purdue University is an Equal Opportunity/Equal Access/Affirmative Action employer committed to achieving a diverse workforce.

Southern Illinois University Assistant/Associate Professor

The Department of Computer Science at Southern Illinois University Edwardsville invites applications for one tenure-track position at the Assistant or Associate Professor level beginning August 2014. Visit cs.siu.edu for more information.

State University of New York at Binghamton Department of Computer Science Four Tenure-Track Assistant Professor Positions

Applications are invited for **four tenure-track Assistant Professor** positions beginning Fall 2014 with specializations in: (a) cybersecurity (three positions) and, (b) embedded systems programming/design with an emphasis on energy optimization (one position). The Department has established graduate and undergraduate programs, including 60 full-time PhD students. Junior faculty have a significantly reduced teaching load for at least the first three years. Please indicate your teaching and research areas of interest in a single sentence on your cover letter.

Further details and application information are available at:

<http://www.binghamton.edu/cs>

Applications will be reviewed until positions are filled. First consideration will be given to applications received by **February 17, 2014**.

We are an EE/AA employer.

JOIN THE INNOVATION.

Qatar Computing Research Institute seeks talented scientists and software engineers to join our team and conduct world-class applied research focused on tackling large-scale computing challenges.

We offer unique opportunities for a strong career spanning academic and applied research in the areas of Arabic language technologies including natural language processing, information retrieval and machine translation, distributed systems, data analytics, cyber security, social computing and computational science and engineering.

Scientist applicants must hold (or will hold at the time of hiring) a PhD degree, and should have a compelling track record of accomplishments and publications, strong academic excellence, effective communication and collaboration skills.

Software engineer applicants must hold a degree in computer science, computer engineering or related field; MSc or PhD degree is a plus.

We also welcome applications for post doctoral researcher positions.

As a **national research institute** and proud member of Qatar Foundation, our research program offers a collaborative, multidisciplinary team environment endowed with a comprehensive support infrastructure.

Successful candidates will be offered a highly competitive compensation package including an attractive tax-free salary and additional benefits such as furnished accommodation, excellent medical insurance, generous annual paid leave, and more.

For full details about our vacancies and how to apply online please visit <http://www.qcri.qa/join-us/> For queries, please email QFJobs@qf.org.qa



معهد قطر لبحوث الحوسبة
Qatar Computing Research Institute

عضو في مؤسسة قطر
Member of Qatar Foundation



/QCRI.QA



@QatarComputing



QatarComputing



QatarComputing

www.qcri.qa

The Cooper Union for the Advancement of Science and Art Open-Rank Faculty Position in Computer Science

The *Albert Nerken School of Engineering* at The Cooper Union for the Advancement of Science and Art seeks outstanding candidates for a tenured or tenure-track faculty position in Computer Science. The Nerken School is distinguished by a curriculum that is rigorous, analytical, and project-oriented and a student body that is highly gifted. The School's new initiatives emphasize computing, entrepreneurship, design, undergraduate research, global partnerships, graduate programs, and teaching innovation. Faculty will develop computing and entrepreneurship programs, leveraging MOOC content and the New York City tech community. Preferred candidates will have established records of funded research in information-based sciences, with ability to involve undergraduates in applied research and tech transfer.

In the heart of Manhattan, The Cooper Union offers an unparalleled education in engineering, art, and architecture. The College typically admits 8 percent of applicants and ranks number one among Baccalaureate Colleges in the Northeast in *U.S. News & World Report*.

A PhD is required. Please submit a C.V., statements of teaching and research, and contact information of at least three references to hr@cooper.edu. Direct questions to NerkenSchool@cooper.edu.

gmail.com. Benefits and work terms are negotiated through the Cooper Union Federation of College Teachers bargaining unit. The Cooper Union is an AA/EOE by choice. Women and individuals from underrepresented groups are encouraged to apply.

University of Central Florida Ten Provost Professorships in Engineering & Computer Science

The University of Central Florida (UCF) announces multiple Provost Professorships to be filled by the College of Engineering and Computer Science (CECS). The Provost Professorship is accompanied with a yearly allocation of discretionary funds to facilitate the candidate's expected extraordinary research productivity.

We are seeking outstanding candidates in all disciplines associated with the College's research mission and are especially interested in candidates who work across academic fields both within and outside of the College's domains. With this targeted hiring initiative, the University seeks to build on its existing strengths in the engineering and computing disciplines by adding senior faculty members who will have an immediate impact on the College's research funding and scholarly productivity.

Specific requirements for appointment to one of these positions are:

- An earned doctoral degree in engineering, computer science or other appropriate disciplines
- A record of accomplishments that justifies appointment as a tenured full professor
- A professional history of successfully mentoring those in junior positions

Highly desirable qualifications include:

- A record of securing competitive research funding
- A record of sustained scholarly achievements, including publications in highly respected venues and international peer recognition
- A history of working in teams, especially those that span multiple disciplines
- A record of directing PhD students to completion

Located in Orlando, FL, UCF is one of the nation's most dynamic metropolitan research universities, having been recognized as a "very high research activity" institution by the Carnegie Foundation, and has been ranked consistently in the top 10 in the country in the impact of its patents. UCF is also a very academically diverse institution, offering 91 undergraduate, 86 masters and 31 doctoral programs along with the M.D. degree in its College of Medicine.

Candidates must submit all documents on-line to <http://www.jobswithucf.com/postings/37068>. Applicants must submit all required documents at the time of application. Required documents include a signed cover letter; complete curriculum vitae; maximum two-page statement each outlining research vision and teaching interests; a list of at least three references with contact information. Review of applications will begin immediately and continue until the positions are filled.

Interested persons with questions about the positions may contact the Search Committee Chair, Dr. MJ Soileau, Vice President of Research, at mj@ucf.edu.

UCF is an equal opportunity, affirmative action employer and encourages the candidacies of women, members of racial and ethnic minorities, and persons with disabilities. All searches and documents are subject to the Sunshine and public records laws of the State of Florida.

University of Maryland, Baltimore County Computer Science and Electrical Engineering Department Two Tenure Track Assistant Professor Positions, Computer Science

We invite applications for two tenure track positions in Computer Science at the rank of Assistant Professor to begin in August 2014. All areas will be considered, but we are especially interested in candidates in systems, security, or data analytics. Unusually strong candidates at the Associate Professor level will be considered. Submit a cover letter, brief statement of teaching and research experience and interests, CV, and three letters of recommendation. See <http://csee.umbc.edu/about/jobs/> for more information about this search and concurrent searches for a tenure track position in Electrical and Computer Engineering and a Professor of the Practice position in Computer Science. UMBC is an AA/EOE.

Faculty Positions - Hamad bin Khalifa University

Hamad bin Khalifa University (HBKU), a member of Qatar Foundation for Education, Science and Community Development, is an emerging research university that offers graduate research and educational programs in a unique research education model for world-class higher education. It is innovative, collaborative, and international, breaking with tradition by establishing interdisciplinary graduate colleges that will offer an array of research education Master's and Doctoral programs, and conduct original, cutting-edge research.

The College of Science, Engineering, and Technology (CSET), a newly established College within HBKU, is seeking applicants to be founding faculty members at all levels (Assistant, Associate, and Full Professor) in all areas of Engineering, Life Sciences, and Computer Sciences. Successful candidates must have an earned PhD degree from an accredited university, a demonstrated ability for research, and a commitment to graduate teaching and program development.

This is a great and unique time to consider joining HBKU and CSET as founding members. The University and College offer unprecedented opportunities for professional enrichment and personal fulfillment, with ample research funding opportunities.

HBKU offers an attractive compensation package that includes a tax-free salary and additional benefits such as furnished accommodation, annual paid leave, medical insurance, etc.

Candidates should apply via email by sending a cover letter, résumé (including the names and contact information of at least three references), a research statement, and a teaching statement (all in PDF format) to Professor Mounir Hamdi, Dean of the College of Science, Engineering, and Technology, at dean.cset@qf.org.qa.

For more information on Hamad bin Khalifa University, visit www.hbku.edu.qa. For more information on Qatar Foundation, visit www.qf.org.qa.

hbku.edu.qa



جامعة حمد بن خليفة
HAMAD BIN KHALIFA UNIVERSITY

University of Rochester
Department of Computer Science
Faculty Positions in Computer Science:
Experimental Systems and Data Science

The University of Rochester Department of Computer Science seeks applicants for multiple tenure track positions in the broad areas of experimental systems and data science research (including but not exclusively focused on very large data-driven systems, machine learning and/or optimization, networks and distributed systems, operating systems, sustainable systems, security, and cloud computing). Candidates must have a PhD in computer science or a related discipline.

Apply online at
<https://www.rochester.edu/fort/csc>

Consideration of applications at any rank will begin immediately and continue until all interview slots are filled. Candidates should apply no later than January 1, 2014 for full consideration. Applications that arrive after this date incur a probability of being overlooked or arriving after the interview schedule is filled up.

The Department of Electrical and Computer Engineering (<http://www.ece.rochester.edu/about/jobs.html>) is also searching for a candidate broadly oriented toward data science. While the two searches are concurrent and plan to coordinate, candidates should apply to the department/s that best match their academic background and interests.

The Department of Computer Science is a research-oriented department with a distinguished history of contributions in systems, theory, artificial intelligence, and HCI. We have a collaborative culture and strong ties to electrical and computer engineering, cognitive science, linguistics, and several departments in the medical center. Over the past decade, a third of the department's PhD graduates have won tenure-track faculty positions, and its alumni include leaders at major research laboratories such as Google, Microsoft, and IBM.

The University of Rochester is a private, Tier I research institution located in western New York State. It consistently ranks among the top 30 institutions, both public and private, in federal funding for research and development. The university has made substantial investments in computing infrastructure through the Center for Integrated Research Computing (CIRC) and the Health Sciences Center for Computational Innovation (HSCCI). Teaching loads are light and classes are small. Half of all undergraduates go on to post-graduate or professional education. The university includes the Eastman School of Music, a premiere music conservatory, and the University of Rochester Medical Center, a major medical school, research center, and hospital system. The greater Rochester area is home to over a million people, including 80,000 students who attend its 8 colleges and universities.

The University of Rochester has a strong commitment to diversity and actively encourages applications from candidates from groups under-represented in higher education. The University is an Equal Opportunity Employer.



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 Search

ShanghaiTech University

The newly launched *ShanghaiTech University* invites *highly qualified* candidates to fill multiple tenure-track/tenured faculty positions as its core team in the School of Information Science and Technology (SIST). Candidates should have exceptional academic records or demonstrate strong potential in cutting-edge research areas of information science and technology. They must be fluent in English. Overseas academic connection or background is highly desired.

ShanghaiTech is built as a world-class research university for training future generations of scientists, entrepreneurs, and technological leaders. Located in Zhangjiang High-Tech Park in the cosmopolitan Shanghai, ShanghaiTech is ready to trail-blaze a new education system in China. Besides establishing and maintaining a world-class research profile, faculty candidates are also expected to contribute substantially to graduate and undergraduate education within the school.

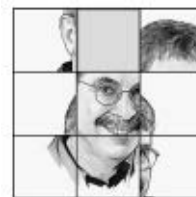
Academic Disciplines: We seek candidates in all cutting edge areas of information science and technology. Our recruitment focus includes, but is not limited to: computer architecture and technologies, nano-scale electronics, high speed and RF circuits, intelligent and integrated signal processing systems, computational foundations, big data, data mining, visualization, computer vision, bio-computing, smart energy/power devices and systems, next-generation networking, as well as inter-disciplinary areas involving information science and technology.

Compensation and Benefits: Salary and startup funds are highly competitive, commensurate with experience and academic accomplishment. We also offer a comprehensive benefit package to employees and eligible dependents, including housing benefits. All regular ShanghaiTech faculty members will be within its new tenure-track system commensurate with international practice for performance evaluation and promotion.

Qualifications:

- A detailed research plan and demonstrated record/potentials;
- Ph.D. (Electrical Engineering, Computer Engineering, Computer Science, or related field);
- A minimum relevant research experience of 4 years.

Applications: Submit (in English) a cover letter, a 2-page research plan, a CV plus copies of 3 most significant publications, and names of three referees to: sist@shanghaitech.edu.cn by March 31st, 2014 (until positions are filled). For more information, visit <http://www.shanghaitech.edu.cn>.



DOI:10.1145/2559597

Peter Winkler

Puzzled

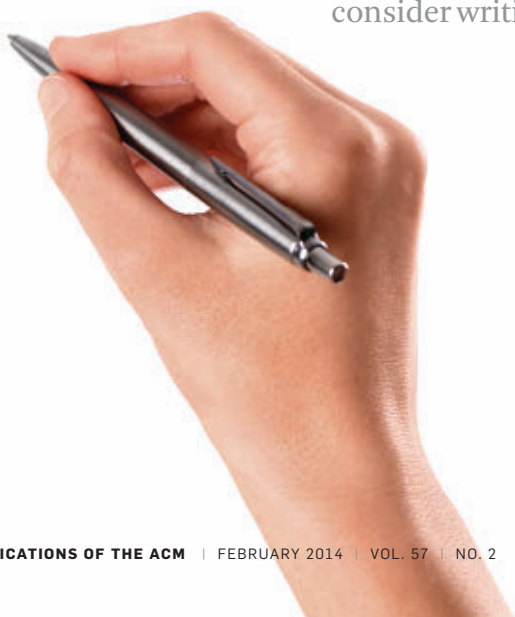
Lowest Number Wins

Each of these puzzles involves a symmetric game. You will be asked about your best strategy, but what does “best strategy” mean? Here, we want a strategy that is a “Nash equilibrium” for all players; that is, one with the property that if it is followed by all other players, you can do no better than follow it yourself. Often, such a strategy requires that players do some randomization; for example, in the familiar game “Rock, Paper, Scissors,” the Nash equilibrium strategy requires each player to choose rock, paper, or scissors with equal probability. As in “Rock, Paper, Scissors,” plays in the games here are done simultaneously, with no collaboration allowed, so every man/woman for him/herself. For solutions and sources, see next month’s column.

1. Having found a dollar bill on the street, Alice and Bob each write down a positive integer. Lowest integer wins the dollar. If they each write the same number, the dollar is torn up. What is the highest integer you, as Alice, should consider writing down?

2. No dollar is found this time. Alice and Bob instead play a “zero-sum game” with their own money. Each again writes down a positive integer. Lowest integer wins \$1 from the other player, unless it is lower by exactly 1; in that case, the player with the higher number wins \$2 from the other player. If the players happen to choose the same number, no money changes hands. What is the highest integer you, as Alice, should consider writing down?

3. Three players this time, with a \$10 prize to be given to the player who writes down the lowest number not written down by any other player. For example, if Alice and Bob each write “1” and Charlie writes “2,” Charlie wins the \$10. If Alice writes “2,” Bob “3,” and Charlie “5,” Alice wins. If all three write the same number, the prize goes unclaimed. What is the highest integer you, as Alice, should consider writing down?

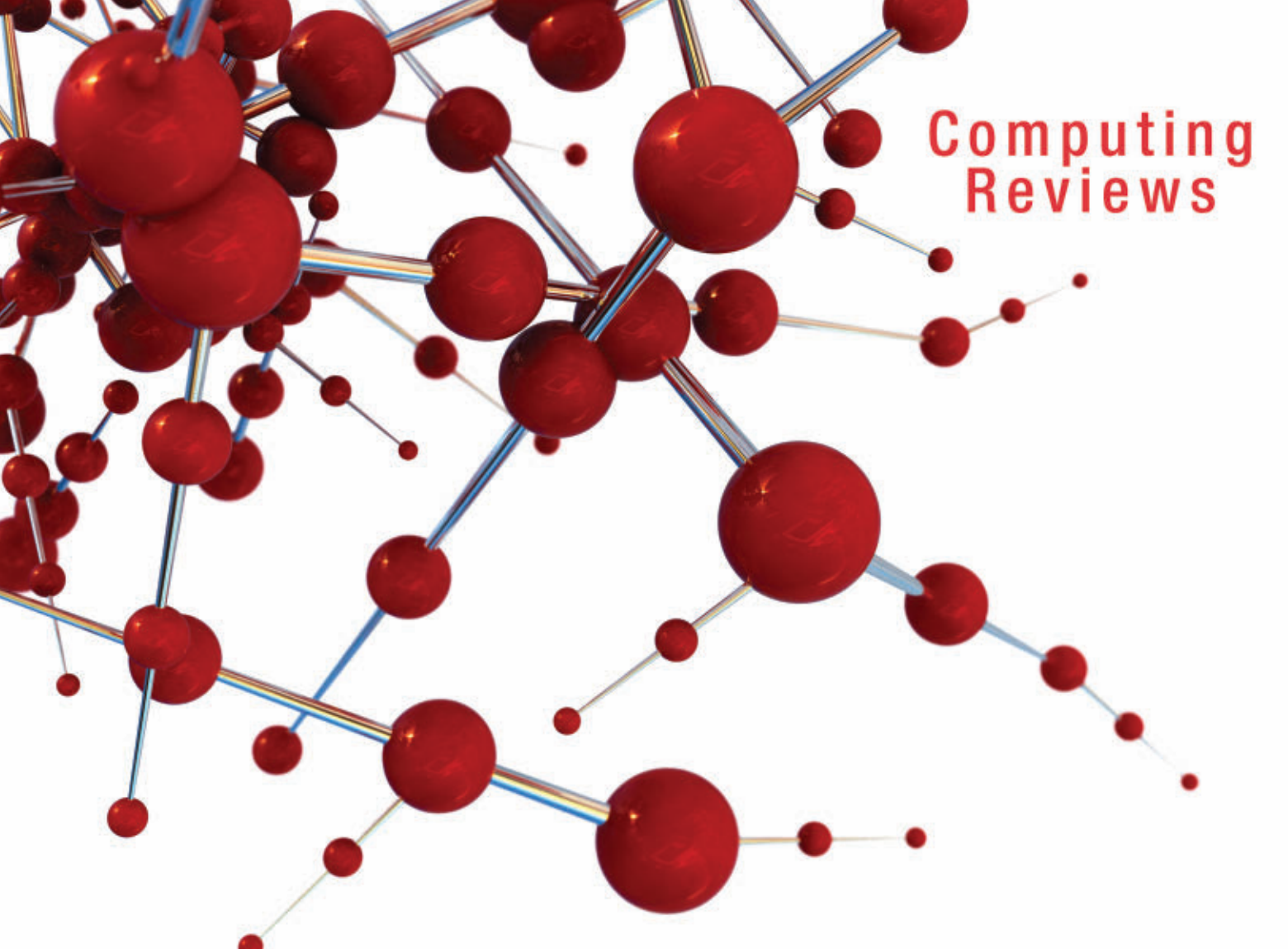


Corrections to “Solutions and Sources” (Dec. 2013) to “Coin Flipping” (Nov. 2013). Three careful readers pointed out there are 12, not 10 (as we said in the solution to Problem 1) head-tail sequences of length 5 that take on average only 32 flips to occur. In addition, Joseph Skudlarek found a sequence (HTTHH or its complement THHTT) that gets the first player a better than 1/3 chance to win in Problem 2, 9/26, to be exact.

Readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is William Morrill Professor of Mathematics and Computer Science at Dartmouth College, Hanover, NH.

Copyright held by Author/Owner(s).



Computing Reviews

**CONNECT WITH OUR
COMMUNITY OF REVIEWERS.**

www.computingreviews.com



Association for
Computing Machinery

ThinkLoud

They'll help you find the best new books
and articles in computing.

Computing Reviews is a collaboration between the ACM and ThinkLoud.

32nd ACM Conference on Human Factors in Computing Systems

April 26th – May 1st

Toronto, Canada



CHI 2014
One of a CHInd

Inspiring keynotes: Margaret Atwood, Scott Jenson

World-leading research findings

Courses from HCI legends including Bill Buxton & Don Norman

Industry focused HCI in practice & case study sessions

Attend. Be changed. Change the world.

chi2014.acm.org @sig_chi

Conference Chairs: Matt Jones, Philippe Palanque

Technical Program Chairs: Tovi Grossman, Albrecht Schmidt