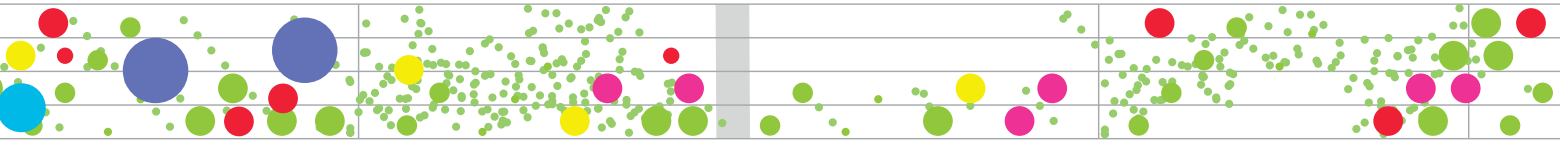
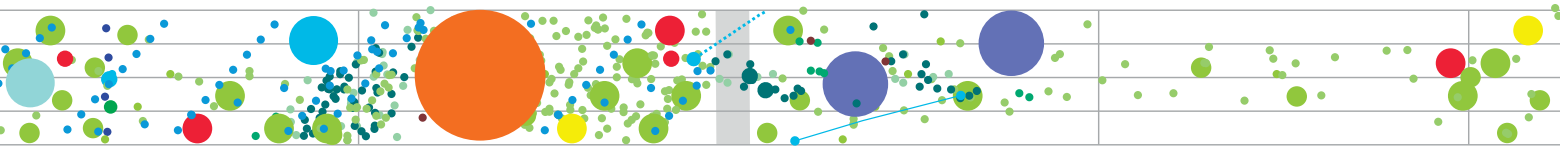
COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

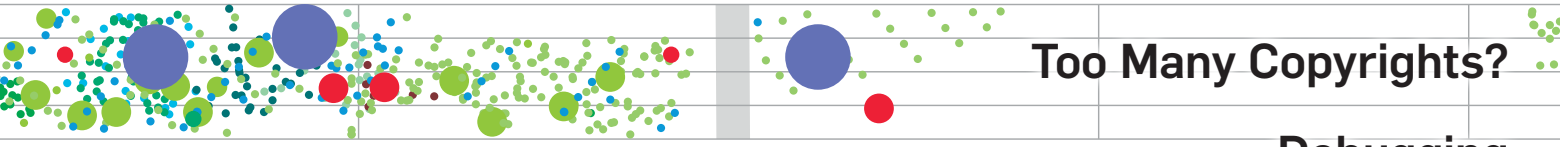
07/2011 VOL.54 NO.7



Computational Thinking In Music

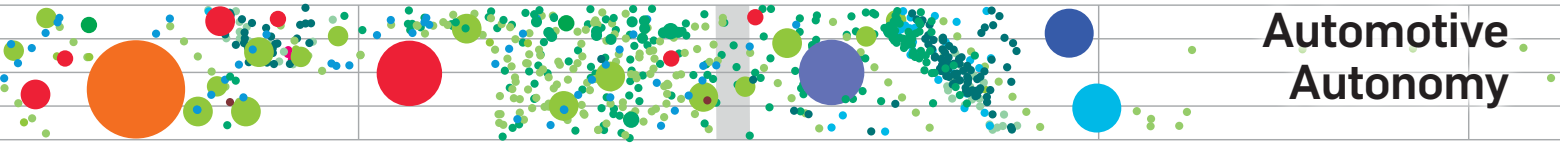


Cellular Telephony And The Question Of Privacy



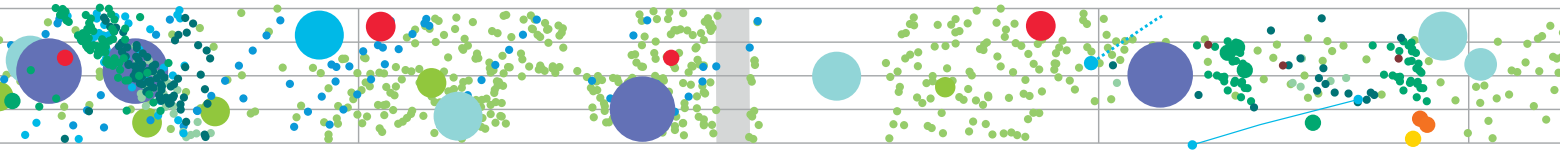
Too Many Copyrights?

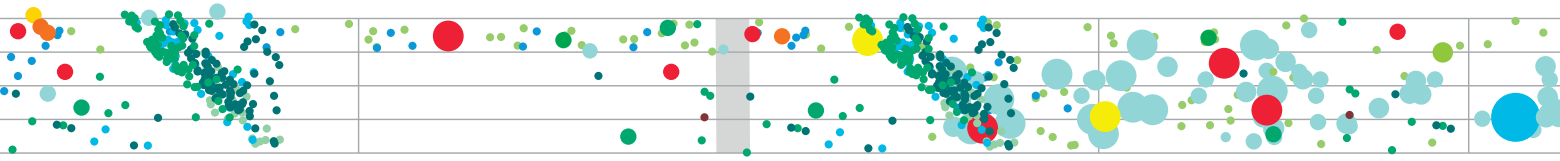
Debugging In The (Very) Large



Automotive Autonomy

DSL For The Uninitiated



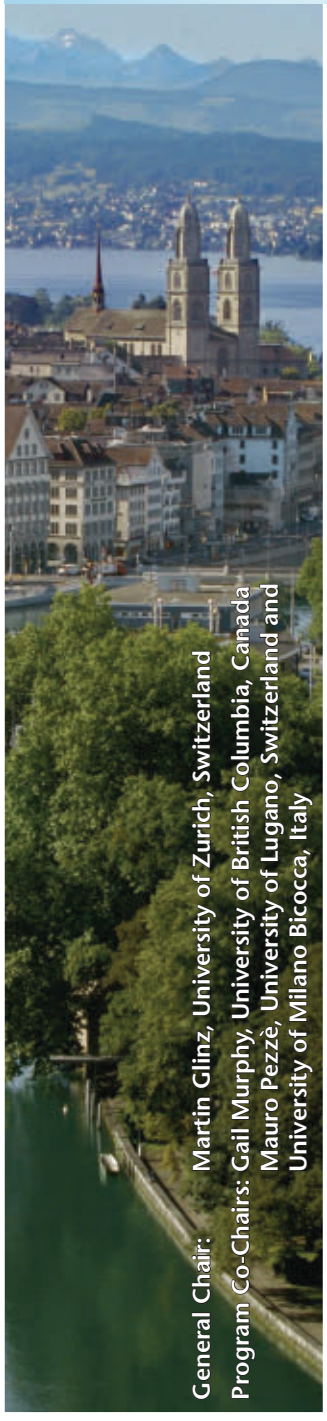


Association for Computing Machinery

34th International Conference on Software Engineering

ICSE 2012

June 2-9, 2012
Zurich • Switzerland



Sustainable Software for a Sustainable World

Submit to ICSE 2012!

Mark your agenda

Sep 29, 2011	Technical research papers
Oct 27, 2011	Software engineering in practice papers · Software engineering education papers · Formal research demonstrations Workshop proposals · Tutorial and technical briefing proposals
Dec 1, 2011	New ideas and emerging results · Doctoral symposium submissions
Feb 17, 2012	Workshop papers · Posters · Informal demonstrations
Jun 2-9, 2012	Conference

IEEE computer society

tcse Technical Council on Software Engineering

University of Zurich[UZH]
Department of Informatics

acm

SIGSOFT Special Interest Group on Software Engineering

SI-SE

http://www.icse2012.org

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.

Eligibility

Each nominated dissertation must have been accepted (successfully defended) by the department between October 2010 and September 2011. Exceptional dissertations completed in September 2010, but too late for submission last year will be considered. 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.

Deadline

Submissions must be received by **October 31, 2011** to qualify for consideration.

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 2012.

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

COMMUNICATIONS OF THE ACM

Departments

5 Editor's Letter
Solving the Unsolvable
By Moshe Y. Vardi

6 Letters To The Editor
Practical Research Yields Fundamental Insight, Too

9 In the Virtual Extension

10 BLOG@CACM
Reviewing Peer Review
Jeannette M. Wing discusses peer review and its importance in terms of public trust. Ed H. Chi writes about alternatives, such as open peer commentary.

12 CACM Online
ACM Aggregates Publication Statistics in the ACM Digital Library
By Scott E. Delman

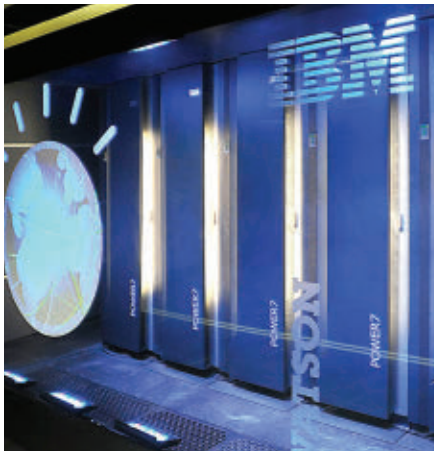
31 Calendar

117 Careers

Last Byte

120 Future Tense
My Office Mate
I became a biocomputational zombie for science...and for love.
By Rudy Rucker

News



13 **Weighing Watson's Impact**
Does IBM's Watson represent a distinct breakthrough in machine learning and natural language processing or is the 2,880-core wunderkind merely a solid feat of engineering?
By Kirk L. Kroeker

16 **Automotive Autonomy**
Self-driving cars are inching closer to the assembly line, thanks to promising new projects from Google and the European Union.
By Alex Wright

19 **Brave, New Social World**
How three different individuals in three different countries—Brazil, Egypt, and Japan—use Facebook, Twitter, and other social-media tools.
By Dennis McCafferty

22 **ACM Award Recipients**
Craig Gentry, Kurt Mehlhorn, and other computer scientists are honored for their research and service.

Viewpoints

23 Technology Strategy and Management
Driving Power in Global Supply Chains
How global and local influences affect product manufacturers.
By Mari Sako

26 Computing Ethics
Values in Design
Focusing on socio-technical design with values as a critical component in the design process.
By Cory Knobel and Geoffrey C. Bowker

29 Legally Speaking
Too Many Copyrights?
Reinstituting formalities—notice of copyright claims and registration requirements—could help address problems related to too many copyrights that last for too many years.
By Pamela Samuelson

32 Broadening Participation
The Status of Women of Color in Computer Science
Addressing the challenges of increasing the number of women of color in computing and ensuring their success.
By Maria (Mia) Ong

35 Viewpoint
Non-Myths About Programming
Viewing computer science in a broader context to dispel common misperceptions about studying computer science.
By Mordechai (Moti) Ben-Ari

Association for Computing Machinery
Advancing Computing as a Science & Profession

PHOTOGRAPH COURTESY OF IBM

Practice



38 **Passing a Language Through the Eye of a Needle**
How the embeddability of Lua impacted its design.
By Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes

44 **DSL for the Uninitiated**
Domain-specific languages bridge the semantic gap in programming.
By Debasish Ghosh

51 **Microsoft's Protocol Documentation Program: Interoperability Testing at Scale**
A discussion with Nico Kicillof, Wolfgang Grieskamp, and Bob Binder.
ACM Case Study

Articles' development led by acmqueue
queue.acm.org

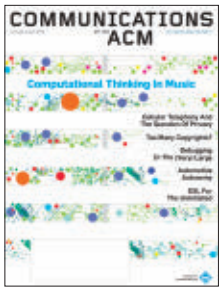
Contributed Articles

58 **Algorithmic Composition: Computational Thinking in Music**
The composer still composes but also gets to take a programming-enabled journey of musical discovery.
By Michael Edwards

68 **A Decade of Software Model Checking with SLAM**
SLAM is a program-analysis engine used to check if clients of an API follow the API's stateful usage rules.
By Thomas Ball, Vladimir Levin, and Sriram K. Rajamani

77 **Searching for Jim Gray: A Technical Overview**
The volunteer search for Jim Gray, lost at sea in 2007, highlights the challenges of computer-aided emergency response.
By Joseph M. Hellerstein and David L. Tennenhouse (on behalf of a large team of volunteers)

VE **The Case for RAMCloud**
With scalable high-performance storage entirely in DRAM, RAMCloud will enable a new breed of data-intensive applications.
By John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Diego Ongaro, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman



About the Cover:
An algorithmic approach to music composition has been in evidence in Western classical music for at least 1,000 years, says Michael Edwards, who chronicles the history of algorithmic composition before and after the dawn of the digital computer beginning on p. 58. Illustration by Studio Tonne.

Review Articles

88 **Cellular Telephony and the Question of Privacy**
A private overlay may ease concerns over surveillance tools supported by cellular networks.
By Stephen B. Wicker

VE **Workload Management for Power Efficiency in Virtualized Data Centers**
Power-aware dynamic application placement can address underutilization of servers as well as the rising energy costs in a data center.
By Gargi Dasgupta, Amit Sharma, Akshat Verma, Anindya Neogi, and Ravi Kothari

Research Highlights

100 Technical Perspective
FAWN: A Fast Array of Wimpy Nodes
By Luiz André Barroso

101 **FAWN: A Fast Array of Wimpy Nodes**
By David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan

110 Technical Perspective
Is Scale Your Enemy, Or Is Scale Your Friend?
By John Ousterhout

111 **Debugging in the (Very) Large: Ten Years of Implementation and Experience**
By Kinshuman Kinshumann, Kirk Glerum, Steve Greenberg, Gabriel Aul, Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt

ILLUSTRATION BY HANK OSUNA

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 Marketing and Membership
David M. Smith
Director, Office of SIG Services
Donna Cappo
Director, Office of Publications
Bernard Rous
Director, Office of Group Publishing
Scott E. Delman

ACM COUNCIL

President
Alain Chesnais
Vice-President
Barbara G. Ryder
Secretary/Treasurer
Alexander L. Wolf
Past President
Wendy Hall
Chair, SGB Board
Vicki Hanson
Co-Chairs, Publications Board
Ronald Boisvert and Jack Davidson
Members-at-Large
Vinton G. Cerf;
Carlo Ghezzi;
Anthony Joseph;
Mathai Joseph;
Kelly Lyons;
Mary Lou Soffa;
Salil Vadhan
SGB Council Representatives
Joseph A. Konstan;
G. Scott Owens;
Douglas Terry

PUBLICATIONS BOARD

Co-Chairs
Ronald F. Boisvert; Jack Davidson
Board Members
Nikil Dutt; Carol Hutchins; Joseph A. Konstan; Ee-Peng Lim; Catherine McGeoch; M. Tamer Ozsu; Holly Rushmeier; Vincent Shen; Mary Lou Soffa

ACM U.S. Public Policy Office
Cameron Wilson, Director
1828 L Street, N.W., Suite 800
Washington, DC 20036 USA
T (202) 659-9711; F (202) 667-1066

Computer Science Teachers Association
Chris Stephenson
Executive Director
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (800) 401-1799; F (541) 687-1840

Association for Computing Machinery (ACM)
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (212) 869-7440; F (212) 869-0481

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 Directors
Mia Angelica Balaquiot
Brian Greenberg
Production Manager
Lynn D'Addesio
Director of Media Sales
Jennifer Ruzicka
Public Relations Coordinator
Virgina Gold
Publications Assistant
Emily Williams

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
acmhelp@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

EDITORIAL BOARD

EDITOR-IN-CHIEF
Moshe Y. Vardi
eic@cacm.acm.org

NEWS
Co-chairs
Marc Najork and Prabhakar Raghavan
Board Members
Hsiao-Wuen Hon; Mei Kobayashi; William Pulleyblank; Rajeev Rastogi; Jeannette Wing

VIEWPOINTS
Co-chairs
Susanne E. Hambrusch; John Leslie King; J Strother Moore
Board Members
P. Anandan; William Aspray; Stefan Bechtold; Judith Bishop; Stuart I. Feldman; Peter Freeman; Seymour Goodman; Shane Greenstein; Mark Guzdial; Richard Heeks; Rachelle Hollander; Richard Ladner; Susan Landau; Carlos Jose Pereira de Lucena; Beng Chin Ooi; Loren Terveen

PRACTICE
Chair
Stephen Bourne
Board Members
Eric Allman; Charles Beeler; David J. Brown; Bryan Cantrill; Terry Coatta; Stuart Feldman; Benjamin Fried; Pat Hanrahan; Marshall Kirk McKusick; Erik Meijer; George Neville-Neil; Theo Schlossnagle; Jim Waldo

The Practice section of the CACM Editorial Board also serves as the Editorial Board of acmqueue.

CONTRIBUTED ARTICLES
Co-chairs
Al Aho and Georg Gottlob
Board Members
Robert Austin; Yannis Bakos; Elisa Bertino; Gilles Brassard; Kim Bruce; Alan Bundy; Peter Buneman; Andrew Chien; Peter Druschel; Blake Ives; James Larus; Igor Markov; Gail C. Murphy; Shree Nayar; Bernhard Nebel; Lionel M. Ni; Sriram Rajamani; Marie-Christine Rousset; Avi Rubin; Krishan Sabnani; Fred B. Schneider; Abigail Sellen; Ron Shamir; Marc Snir; Larry Snyder; Veda Storey; Manuela Veloso; Michael Vitale; Wolfgang Wahlster; Andy Chi-Chih Yao

RESEARCH HIGHLIGHTS
Co-chairs
David A. Patterson and Stuart J. Russell
Board Members
Martin Abadi; Stuart K. Card; Jon Crowcroft; Shafi Goldwasser; Monika Henzinger; Maurice Herlihy; Dan Huttenlocher; Norm Jouppi; Andrew B. Kahng; Gregory Morrisett; Michael Reiter; Mendel Rosenblum; Ronitt Rubinfeld; David Salesin; Lawrence K. Saul; Guy Steele, Jr.; Madhu Sudan; Gerhard Weikum; Alexander L. Wolf; Margaret H. Wright

WEB
Co-chairs
James Landay and Greg Linden
Board Members
Gene Golovchinsky; Marti Hearst; Jason I. Hong; Jeff Johnson; Wendy E. MacKay

ACM Copyright Notice
Copyright © 2011 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

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

Subscriptions
An annual subscription cost is included in ACM member dues of $99 ($40 of which is allocated to a subscription to *Communications*); for students, cost is included in $42 dues ($20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is $100.

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

Single Copies
Single copies of *Communications of the ACM* are available for purchase. Please contact 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:10.1145/1965724.1965725

Moshe Y. Vardi

Solving the Unsolvable

On June 16, 1902, British philosopher Bertrand Russell sent a letter to Gottlob Frege, a German logician, in which he argued, by using what became known as "Russell's Paradox," that Frege's logical system was inconsistent. The letter launched a "Foundational Crisis" in mathematics, triggering an almost anguished search for proper foundations for mathematics. In 1921, David Hilbert, the preeminent German mathematician, launched a research program aimed at disposing "the foundational questions once and for all." Hilbert's Program failed; in 1931, Austrian logician Kurt Goedel proved two incompleteness theorems that proved the futility of Hilbert's Program.

One element in Hilbert's Program was the mechanization of mathematics: "Once a logical formalism is established one can expect that a systematic, so-to-say computational, treatment of logic formulas is possible, which would somewhat correspond to the theory of equations in algebra." In 1928, Hilbert and Ackermann posed the "Entscheidungsproblem" (Decision Problem), which asked if there is an algorithm for checking whether a given formula in (first-order) logic is valid; that is, necessarily true. In 1936–1937, Alonzo Church, an American logician, and Alan Turing, a British logician, proved independently that the Decision Problem for first-order logic is *unsolvable*; there is no algorithm that checks the validity of logical formulas. The Church-Turing Theorem can be viewed as the birth of theoretical computer science. To prove the theorem, Church and Turing introduced computational models, recursive functions, and Turing machines, respectively, and proved that the *Halting Problem*—checking whether a given recursive function or Turing machine yields an output on a given input—is unsolvable.

The unsolvability of the Halting Problem, proved just as Konrad Zuse in Germany and John Atanasoff and Clifford Berry in the U.S. were embarking on the construction of their digital computers—the Z3 and the Atanasoff-Berry Computer—meant that computer science was born with a knowledge of the inherent limitation of mechanical computation. While Hilbert believed that "every mathematical problem is necessarily capable of strict resolution," we know that *the unsolvable* is a barrier that cannot be breached. When I encountered unsolvability as a fresh graduate student, it seemed to me an insurmountable wall. Much of my research over the years was dedicated to delineating the boundary between the solvable and the unsolvable.

It is quite remarkable, therefore, that the May 2011 issue of *Communications* included an article by Byron Cook, Andreas Podelski, and Andrey Rybalchenko, titled "Proving Program Termination" (p. 88), in which they argued that "in contrast to popular belief, proving termination is not always impossible." Surely they got it wrong! The Halting Problem (termination is the same as halting) is unsolvable! Of course, Cook et al. do not really claim to have solved the Halting Problem. What they describe in the article is a new method for proving termination of programs. The method itself is not guaranteed to terminate—if it did, this would contradict the Church-Turing Theorem. What Cook et al. illustrate is that the method is remarkably effective in practice and can handle a large number of real-life programs. In fact, a software tool called Terminator, used to implement their method, has been able to find some very subtle termination errors in Microsoft software.

I believe this noteworthy progress in proving program termination ought to force us to reconsider the meaning of unsolvability. In my November 2010 editorial, "On P, NP, and Computational Complexity," I pointed out that NP-complete problems, such as Boolean Satisfiability, do not seem as intractable today as they seemed in the early 1970s, with industrial SAT solvers performing impressively in practice. "Proving Program Termination" shows that unsolvable problems may not be as unsolvable as we once thought. In theory, unsolvabilty does impose a rigid barrier on computability, but it is less clear how significant this barrier is in practice. Unlike Collatz's Problem, described in the article by Cook et al., most real-life programs, if they terminate, do so for rather simple reasons, because programmers almost never conceive of very deep and sophisticated reasons for termination. Therefore, it should not be shocking that a tool such as Terminator can prove termination for such programs.

Ultimately, software development is an engineering activity, not a mathematical activity. Engineering design and analysis techniques do not provide mathematical guarantee, they provide confidence. We do not need to solve the Halting Problem, we just need to be able to reason successfully about termination of real-life programs. It is time to give up our "unsolvability phobia." It is time to solve the unsolvable.

Moshe Y. Vardi, EDITOR-IN-CHIEF

DOI:10.1145/1965724.1965726

Practical Research Yields Fundamental Insight, Too

Tim Wu's viewpoint "Bell Labs and Centralized Innovation" (May 2011) was inaccurate regarding a specific example of research at Bell Labs.

Wu wrote, "Bell's scientists did cutting-edge work in fields as diverse as quantum physics and data theory. It was a Bell Labs employee named Clinton Davisson who would win a Nobel Prize for demonstrating the wave nature of matter, an insight more typically credited to Einstein than to a telephone company employee." However, Albert Einstein actually discovered that some perplexing data regarding the photoelectric effect could be explained through a hypothesis proposing that light, previously described purely as waves, could behave as particles, now called photons. Others, in particular Louis de Broglie, proposed that matter, previously viewed as particles, could be described by waves. While the Davisson-Germer experiment confirmed de Broglie, neither Davisson nor Lester Germer at the time knew about de Broglie's research; see http://courses.science.fau.edu/voss/modphys/pdf/Ch05_2.pdf.

Germer (a casual acquaintance) told me he and Davisson did not realize the data showed the wave nature of matter initially due to the wave nature of matter being a rather esoteric idea at the time. That is, they discovered something very important but somewhat by accident. It took time before these two researchers realized what they had actually measured.

There were practical reasons (of interest to a telephone company) for Davisson's and Germer's research, including vacuum tubes, which were then used in amplifiers. Electrons arrive at a vacuum tube's anode with enough energy to cause secondary emission of electrons at the anode, in some cases degrading a vacuum tube's performance.

Understanding how electrons interact with an anode was obviously useful in any attempt to improve the anode's design.

William Zaumen, Palo Alto, CA

Author's Response:

Zaumen is correct. Davisson demonstrated that all particles, not light, have wave-like properties; for example, electrons, and even people, have a wave-like nature. Zaumen is also correct in saying that Einstein worked in a field that assumed light was wave-like, showing its particle-like properties.

Tim Wu, New York

No Reconciling Irreconcilable Models

Erik Meijer's and Gavin Bierman's article "A Co-Relational Model of Data for Large Shared Data Banks" (Apr. 2011) overreached by claiming equivalence between the Relational Model and NoSQL "key-value pairs" without regard to the definition of a data model by E.F. Codd more than 30 years ago. Finding similarity in NoSQL systems to some parts of the Relational Model, Meijer and Bierman mistakenly concluded the two are equivalent.

Codd, in his paper "Data Models in Database Management" in *Proceedings of the 1980 Workshop on Data Abstraction, Databases and Conceptual Modeling* (http://portal.acm.org/citation.cfm?id=806891) defined a data model as comprising three components: data structures to represent well-formed expressions in first-order logic; operators closed over these structures, permitting inferencing; and integrity constraints to enforce internal consistency.

NoSQL systems have no data model so defined. All else is commentary.

Meijer and Bierman ignored logic and inferencing and did not explain how key-value systems recognize, let alone enforce, integrity constraints. They cited referential integrity—a form of integrity constraint—as an exogenous cost relational databases bear to correct for a deficiency. The truth is actually the opposite; consistency is a central obligation of any database-management system. The lack of constraint-checking in key-value systems imposes the constraint-checking burden on the application, a situation the Relational Model was invented specifically to correct.

Codd encountered a similar lack of understanding in his day. In the same proceedings paper, he wrote, "In comparing data models people often ignore the operators and integrity rules altogether. When this occurs, the resulting comparisons run the risk of being meaningless."

Codd's landmark article "A Relational Model of Data for Large Shared Data Banks" (*Communications*, June 1970) addressed other points raised by Meijer and Bierman, including path independence. An interested reader would learn much in an evening spent with that one article alone.

Object Relational Mapping libraries and NoSQL systems attempt to solve (through technical means) a nontechnical problem: reluctance of talented people to master the Relational Model, and thus benefit from its data consistency and logical inferencing capabilities. Rather than exploit it and demand more relational functionality from DBMS vendors, they seek to avoid and replace it, unwittingly advocating a return to the fragile, unreliable, illogical systems of the 1960s, minus the green-bar fanfold paper.

James K. Lowden, New York

Authors' Response:

Lowden's comment contains a number of errors. Our article was, in fact, explicitly critical of the lack of an agreed data model for NoSQL. We didn't ignore "inferencing," proposing instead a query language based on monad comprehensions—interestingly, the same query language we prefer for the relational model. We did not assert that

the relational and key-value models are equivalent, but rather dual. The issue of weakening consistency checking goes to the heart of the interest in NoSQL systems and is beyond the scope of our article.

Erik Meijer, Redmond, WA
Gavin Bierman, Cambridge, U.K.

Financial Incentives vs. Algorithms in Social Networks

I thank John C. Tang et al. for their analysis of the crowdsourcing strategies of three successful teams in their article "Reflecting on the DARPA Red Balloon Challenge" (Apr. 2011). Though the iSchools team might have had better data-mining algorithms, it was the MIT team that recognized and exploited financial incentives as the most effective way to be first to identify the 10 red balloons DARPA scattered across the U.S. last year.

In retrospect, the recursive incentive strategy adopted by the MIT team is used in many network-marketing situations worldwide. I first came across it almost 20 years ago when trying to sell a database management system to one of India's oldest non-banking finance companies, which happened to employ a motivated network of insurance agents throughout India. These agents were required to recruit other agents, with the initial premium for the first few months from each new account they signed up distributed hierarchically, though not in the precise geometric progression the MIT team used in the DARPA Challenge. This way, the company's senior agents, having recruited a large network, could virtually sit back and watch as the money poured in. I suppose this, too, is how most Ponzi schemes work, though, in this case, nothing illegal was involved, as is generally implied by the term.

The important takeaway from the Tang et al. analysis is that motivating people is the key to success and that money is often the most effective motivation in any given social network. Whether that is good or bad is a question that needs a totally different kind of analysis.

Prithwis Mukerjee, Kharagpur, India

Let Leap Seconds Sync

Poul-Henning Kamp's article "The One-Second War" (May 2011) was enlightening and, from the perspective of an old-time (ex)hardware engineer, entertaining. The reason solder jockeys (hardware engineers) don't see leap seconds as a problem is they presume computers know only what they've been told; if the system clock slows by 1/86,400th of a second per second, the system's software won't have the slightest idea it happened, nor will it care.

By extension, astronomers using terrestrial time are (by definition) off by some indeterminate amount until leap time, then off in another direction after the leap. Garden-variety system clocks (not directly atomically controlled) are constantly in need of adjustment and aren't very accurate over days at a time. Diddling a fraction of a millisecond out of a second only disappears in the noise. Since atomic clocks are the reference standard, they can skip however many beats are needed to ensure the seconds counter always reads 86,400 when the solar year ends.

Why not make the (invisible to code) system clock adjustable so it always counts to 86,400 seconds until the moment the year counter ticks over? To the code, a second is whatever a register says it is. Hardware, not software, counts electrical oscillations, and if it includes an "add x seconds in y years" pair of adjustment thumbwheels, the result is that 86,400 will have gone by exactly when the (real) year turns over.

Adjusting to leap seconds can be simple, unless programmers try turning a timing-gate issue into a planetary software project. Let astronomers use whatever time-sync definition they want, but if system clocks are adjusted in tiny amounts to keep "better" time, telescopes will be more accurate than if they were abruptly forced to catch up by a full second each year.

Just tell the electrical engineers the numbers and let them provide them to astronomers, system administrators, home users, and everyone else.

David Byrd, Arlington, VA

Author's Response:

Byrd proposes a number of additional ways we might paper over the fact that the planet is itself an unpredictable and unstable clock. There is no shortage of such ideas, and all are bad hacks. If computers were still huge boxes with a few attached terminals and printer, all these ideas would work, as indeed a number of them did, from the invention of the computer to the mid-1980s. Like today's deployed bad hack—leap seconds—all the schemes Byrd proposes rely on somebody measuring what the planet does and everybody else reacting to it on short notice. His ideas do not improve the current situation in any way but do reintroduce at least one bad idea already discarded—variable-length seconds.

Poul-Henning Kamp,
Slagelse, Denmark

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

© 2011 ACM 0001-0782/11/07 $10.00



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
http://www.acm.org/join

Phone
+1-800-342-6626 (US & Canada)
+1-212-626-0500 (Global)

Fax
+1-212-944-1318

Or, complete this application and return with payment via postal mail

Special rates for residents of developing countries:
http://www.acm.org/membership/L2-3/

Special rates for members of sister societies:
http://www.acm.org/membership/dues.html

Please print clearly

Name

Address

City | State/Province | Postal code/Zip

Country | E-mail address

Area code & Daytime phone | Fax | Member number, if applicable

Purposes of ACM

ACM is dedicated to:
1) advancing the art, science, engineering, and application of information technology
2) fostering the open interchange of information to serve both professionals and the public
3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature

ACM Code of Ethics:
http://www.acm.org/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*. Student membership dues include $15 toward a subscription to *XRDS*. 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:10.1145/1965724.1965727

In the Virtual Extension

To ensure the timely publication of articles, Communications *created the Virtual Extension (VE) to expand the page limitations of the print edition by bringing readers the same high-quality articles in an online-only format. VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on merit. The following synopses are from articles now available in their entirety to ACM members via the Digital Library.*

contributed article

DOI: 10.1145/1965724.1965751

The Case for RAMCloud

John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Diego Ongaro, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman

For the past four decades magnetic disks have been the primary storage location for online information in computer systems. Over that period, disk technology has undergone dramatic improvements while being harnessed by higher-level storage systems (such as file systems and relational databases). However, disk performance has not improved as quickly as disk capacity, and developers find it increasingly difficult to scale disk-based systems to meet the needs of large-scale Web applications. Many computer scientists have proposed new approaches to disk-based storage as a solution, and others have suggested replacing disks with flash memory devices. In contrast, we say the solution is to shift the primary locus of online data from disk to DRAM, with disk relegated to a backup/archival role.

A new class of storage called RAMCloud will provide the storage substrate for many future applications. RAMCloud stores all of its information in the main memories of commodity servers and uses hundreds or thousands of these servers to create a large-scale storage system. Because all data is in DRAM at all times, RAMCloud promises 100x–1,000x lower latency than disk-based systems and 100x–1,000x greater throughput. Though individual memories are volatile, RAMCloud can use replication and backup techniques to provide data durability and availability equivalent to disk-based systems.

The combination of latency and scale offered by RAMCloud will change the storage landscape in three ways: simplify development of large-scale Web applications by eliminating many of the scalability issues that sap developer productivity today; enable a new class of applications that manipulate data 100x–1,000x more intensively than is possible today; and provide the scalable storage substrate needed for cloud computing and other data-center applications.

review article

DOI: 10.1145/1965724.1965752

Workload Management for Power Efficiency in Virtualized Data Centers

Gargi Dasgupta, Amit Sharma, Akshat Verma, Anindya Neogi, and Ravi Kothari

By most estimates, energy-related costs will become the single largest contributor to the overall cost of operating a data center. Ironically, several studies have shown that a typical server in a data center is seriously underutilized. For example, Bohrer et al. find the average server utilization to vary between 11% and 50% for workloads from sports, e-commerce, financial, and Internet proxy clusters. This underutilization is the consequence of provisioning a server for the infrequent though inevitable peaks in the workload. Power-aware dynamic application placement can simultaneously address underutilization of servers as well as the rising energy costs in a data center by migrating applications to better utilize servers and switching freed-up servers to a lower power state.

Though the concept of dynamic application placement is not new, the two recent trends of virtualization and energy management technologies in modern servers have made it possible for it to be widely used in a data center. While virtualization has been the key enabler, power minimization has been the key driver for energy-aware dynamic application placement.

Server virtualization technologies first appeared in the 1960s to enable timesharing of expensive hardware between multiple users. As hardware became less expensive, virtualization gradually lost its charm. However, since the late 1990s there has been renewed interest in server virtualization and is now regarded as a disruptive business model to drive significant cost reductions. Advances in system management allow the benefits of virtualization to be now realized without any appreciable increase in the system management costs.

The benefits of virtualization include more efficient utilization of hardware (especially when each virtual machine, or VM, on a physical server reaches peak utilization at different points in time or when the applications in the individual VMs have complementary resource usage), as well as reduced floor space and facilities management costs. Additionally, virtualization software tends to hide the heterogeneity in server hardware and make applications more portable or resilient to hardware changes. Virtualization Planning entails sizing and placing existing or fresh workloads as VMs on physical servers.

In this article, the authors simplify resource utilization of a workload to be captured only by CPU utilization. However in practice, multiple parameters, such as memory, disk, and network I/O bandwidth consumption, among others, must be considered.

Coming Next Month in COMMUNICATIONS

Cognitive Computing

Reputation Systems for Open Collaboration

An Overview of Business Intelligence Technology

Gender and Computing Conference Papers

Rethinking the Role of Journals in Computer Science

Skinput: Appropriating the Skin as an Interactive Canvas

Storage Strife

As Simple As Possible—But Not More So

And the latest news on supercomputers, Monte Carlo tree search, and improvements in language translation.

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:10.1145/1965724.1965728 http://cacm.acm.org/blogs/blog-cacm

Reviewing Peer Review

Jeannette M. Wing discusses peer review and its importance in terms of public trust. Ed H. Chi writes about alternatives, such as open peer commentary.



Jeannette M. Wing
"Why Peer Review Matters"
http://cacm.acm.org/blogs/blog-cacm/98560

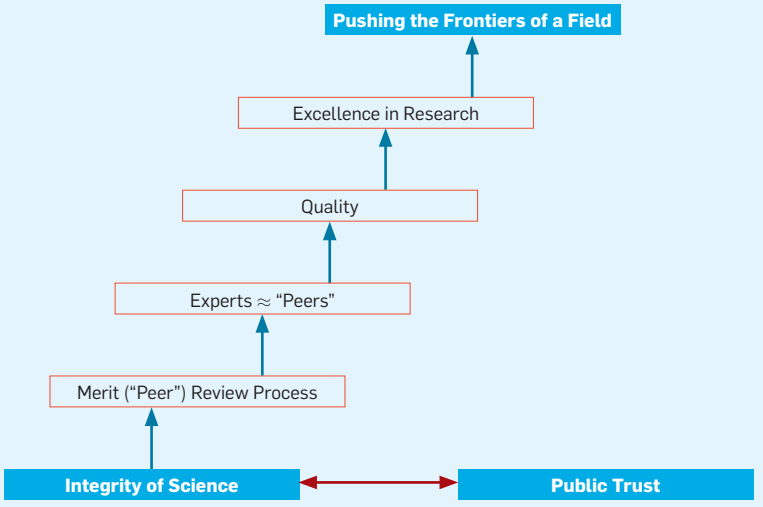
At the most recent Snowbird conference, where the chairs of computer science departments in the U.S. meet every two years, there was a plenary session during which the panelists and audience discussed the peer review processes in computing research, especially as they pertain to a related debate on conferences versus journals. It's good to go back to first principles to see why peer review matters, to inform how we then would think about process.

In research we are interested in discovering new knowledge. With new knowledge we push the frontiers of the field. It is through excellence in research that we advance our field, keeping it vibrant, exciting, and relevant. How is excellence determined? We rely on experts to distinguish new results from previously known, correct results from incorrect, relevant problems from irrelevant, significant results from insignificant, interesting results from dull, the proper use of scientific methods from being sloppy, and so on. We call these experts our peers. Their/our judgment assesses the quality and value of the research we produce. It is important for advancing our field to ensure we do high-quality work. That's why peer review matters.

In science, peer review matters not just for scientific truth, but, in the broader context, for society's perception of science. Peer review matters for the integrity of science. Scientific integrity is the basis for public trust in us, in our results, in science. Most people don't understand the technical details of a scientific result, let alone how it was obtained, what assumptions were made, in what contexts the result is applicable, or what practical implications it has. When they read in the news that "Scientists state X," there is an immediate trust that "X" is true. They know that science uses peer review to vet results before they are published. They trust this process to work. It is important for us, as scientists, not to lose the public trust in science. That's why peer review matters.

"Public" includes policymakers. Most government executives and congressional members are not scientists. They do not understand science, so

Why peer review matters.



they need to rely on the judgment of experts to determine scientific truth and how to interpret scientific results. We want policymakers in the administration and Congress to base policy decisions on facts, on evidence, and on data. So it is important for policymakers that, to the best of our ability, we, as scientists, publish results that are correct. That's why peer review matters.

While I argue peer review matters, it's a whole other question of what the best process is for carrying out peer review. In this day and age of collective intelligence through social networks, we should think creatively about how to harness our own technology to supplement or supplant the traditional means used by journals, conferences, and funding agencies. Peer review matters, and now is the time to revisit our processes—not just procedures and mechanisms, but what it is we review (papers, data, software, and tools), our evaluation criteria, and our incentives for active participation.

Comments

It is important for us, as scientists, not to lose the public trust in science. That's why peer review matters.

I think we must continue to educate our students and the public about truth. Even if a research paper is published in the most respectable venue possible, it could still be wrong. Conventional peer review is essentially an insider game: It does nothing against systematic biases.

In physics, almost everyone posts his papers on arXiv. It is not peer review in the conventional sense. Yet, our trust in physics has not gone down. In fact, Perelman proved the Poincaré conjecture and posted his solution on arXiv, bypassing conventional peer review entirely. Yet, his work was peer reviewed, and very carefully.

We must urgently acknowledge that our traditional peer review is an honor-based system. When people try to game the system, they may get away with it. Thus, it is not the gold standard we make it out to be.

Moreover, conventional peer review puts a high value in getting papers published. It is the very source of the paper-counting routine we go through. If it was as easy to publish a research paper as it is to publish a blog post, nobody would be counting research papers. Thus, we must realize that conventional peer review also has some unintended consequences.

Yes, we need to filter research papers. But the Web, open source software, and Wikipedia have shown us that filtering after publication, rather than before, can work too. And filtering is not so hard.

Filtering after publication is clearly the future. It is more demanding from an IT point of view. It could not work in a paper-based culture. But there is no reason why it can't work in the near future. And the Perelman example shows that it already works.

—Daniel Lemire



Ed H. Chi
"How Should Peer Review Evolve?"
http://cacm.acm.org/blogs/blog-cacm/100284

Peer review publications have been around scientific academic scholarship since 1665, when the Royal Society's funding editor Henry Oldenburg created the first scientific journal. As Jeannette Wing nicely argued in her "Why Peer Review Matters" post, it is the public, formal, and final archival nature of the process of the Oldenburg model that established the importance of publications to scientific authors, as well as their academic standings and careers.

Recently, as the communication of research results reaches breakneck speeds, some have argued that it is time to fundamentally examine the peer review model, and perhaps to modify it somewhat to suit the modern times. One such proposal recently posed to me via email is open peer review, a model not entirely unlike the Wikipedia editing model in many ways. Astute readers will realize the irony of how the Wikipedia editing model makes academics squirm in their seats.

The proposal for open peer review suggests that the incumbent peer review process has problems in bias, suppression, and control by elites against competing non-mainstream theories, models, and methodologies. By opening up the peer review system, we might increase accountability and transparency of the process, and mitigate other flaws. Unfortunately, while we have anecdotal evidence of these issues, there remains significant problems in quantifying these flaws with hard numbers and data, since reviews often remain confidential.

Perhaps more distressing is that several experiments in open peer review (such as done by *Nature* in 2006, *British Medical Journal* in 1999, and *Journal of Interactive Media in Education* in 1996) have had mixed results in terms of the quality and tone of the reviews. Interestingly, and perhaps unsurprisingly, many of those who are invited to review under the new model decline to do so, potentially reducing the pool of reviewers. This is particularly worrisome for academic conferences and journals, at a time when we desperately need more reviewers due to the growth of the number of submissions.

A competing proposal might be open peer commentary, which elicits and publishes commentary on peer-reviewed articles. This can be done prior to publication or after the date of publication. In fact, recent SIGCHI conferences have already started experimenting with this idea, with several popular paper panels in which papers are first presented, and opinions from a panel is openly discussed with an audience. The primary focus here is to increase participation, while also improve transparency. The idea of an open debate, with improved transparency, is of course the cornerstone of the Wikipedia editing model (and the PARC research project WikiDashboard).

Finally, it is worth pointing out the context under which these proposals might be evaluated. We live in a different time than Oldenburg. In the mean time, communication technology has already experienced several revolutions of gigantic proportions. Now, real-time research results are often distributed, blogged, tweeted, Facebooked, Googled, and discussed in virtual meetings. As researchers, we can ill-afford to stare at these changes and not respond.

Beyond fixing problems and issues of bias, suppression, and transparency, we also need to be vigilant of the speed of innovation and whether our publication processes can keep up. Web review-management systems like PrecisionConference have gone a long way in scaling up the peer-review process. What else can we do to respond to this speed of growth yet remain true to the openness and quality of research? C

Jeannette M. Wing is a professor at Carnegie Mellon University. **Ed H. Chi** is a research scientist at Google.

© 2011 ACM 0001-0782/11/07 $10.00



DOI:10.1145/1965724.1965729 Scott E. Delman

ACM Aggregates Publication Statistics in the ACM Digital Library

Many of you know the ACM Digital Library (http://dl.acm.org) consists of a massive full-text archive of all ACM publications (currently over 300,000 articles and growing at a rate of over 22,000 per year). But many of you may not know the DL also consists of the computing field's largest dedicated index of bibliographic records (currently over 1.6 million records and growing) called the *Guide to Computing Literature*, and that starting in 2010 ACM began aggregating these records along with key citation information and online usage data from the DL platform itself to provide a unique and incredibly valuable tool for the computing community at large.



It is now possible to click on any author's name inside the DL and view a complete record of that author's publication history, including a dynamically generated list of all of their ACM and non-ACM publications, affiliations, citations, ACM DL download statistics, and other relevant data related to their publications' history. Currently, over one million author pages exist in the DL, and this figure grows every day!

In addition, ACM aggregates all of this data at the publication level, article level, SIG level, conference level, and most recently the institutional level. All of this data is freely available for users of the ACM DL. For example, *Communications*' page in the DL (see the image here) currently shows the magazine has published 10,691 articles since 1958 with over 117,065 citations in other publications, over 9.2 million downloaded articles from the DL platform, resulting in an average of over 866 downloads per article published and 10.95 citations per article.

On many of these new "bibliometric pages," comparative data also exposes the top cited and downloaded articles, so that authors can view both the usage activity and impact of their work. If you haven't yet spent a few minutes drilling down into these pages, I suggest you do so. The data is fascinating and what you find may surprise you and your colleagues!

ACM Member News

EDWARD W. FELTEN, FTC CHIEF TECHNOLOGIST



To the delight of many in the computer science community, Edward W. Felten became the first chief technologist for the U.S. Federal Trade Commission (FTC) earlier this year. Taking a one-year leave from his position as professor of computer science and public affairs at Princeton University, Felten assumed the FTC position in January to work on technology policy issues related to consumer privacy and security.

As chief technologist, Felten advises the FTC chair and commissioners, and operates as a liaison to the technical community. "I have enjoyed the job so far, and I feel like I am having a positive impact," says Felten, who is a vice-chair of ACM's U.S. Public Policy Council.

While Felten is engaged in several aspects of the FTC's work related to consumer protection and antitrust issues, he is largely focusing on online privacy, which he says is currently receiving a lot of attention at the FTC, especially in terms of online tracking and behavioral marketing.

FTC officials, browser companies, and advertisers are discussing the creation of a do-not-track system, and Felten says there has been significant progress toward a workable system through voluntary steps taken by industry. A do-not-track system that adequately protects consumers might be built without the need for government rulemaking or legislation, according to Felten.

Of Washington's attitudes toward computer scientists, Felten says explaining technical details in a clear and useful way is an important part of his role. "Some see our expertise as valuable for policymaking. Others are still figuring out how to approach us," says Felten. "The more we can engage constructively in the policy process, the more people will learn to listen to us."

—Kirk L. Kroeker

PHOTOGRAPH COURTESY OFFICE OF COMMUNICATIONS, PRINCETON UNIVERSITY

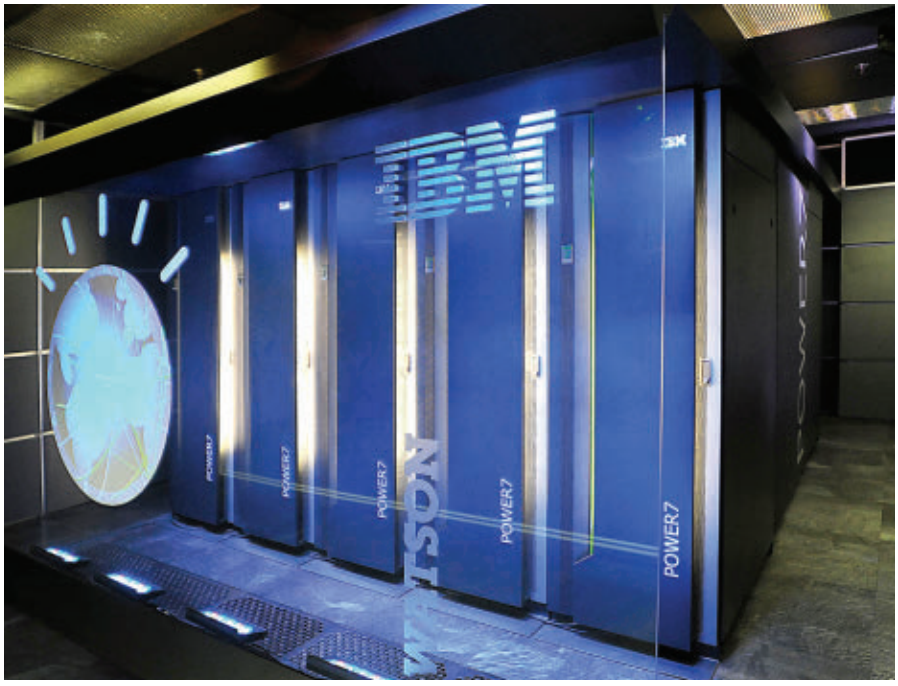
Science | DOI:10.1145/1965724.1965730 Kirk L. Kroeker

Weighing Watson's Impact

Does IBM's Watson represent a distinct breakthrough in machine learning and natural language processing or is the 2,880-core wunderkind merely a solid feat of engineering?

In the history of speculative fiction, from the golden age of science fiction to the present, there are many examples of artificial intelligences engaging their interlocutors in dialogue that exhibits self-awareness, personality, and even empathy. Several fields in computer science, including machine learning and natural language processing, have been steadily approaching the point at which real-world systems will be able to approximate this kind of interaction. IBM's Watson computer, the latest example in a long series of efforts in this area, made a television appearance earlier this year in a widely promoted human-versus-machine "Jeopardy!" game show contest. To many observers, Watson's appearance on "Jeopardy!" marked a milestone on the path toward achieving the kind of sophisticated, knowledge-based interaction that has traditionally been relegated to the realm of fiction.

The "Jeopardy!" event, in which Watson competed against Ken Jennings and Brad Rutter, the two most successful contestants in the game show's history, created a wave of coverage across mainstream and social media. During the three-day contest in February, hints of what might be called Watson's quirky personality shone through, with the machine wagering oddly precise amounts, guessing at answers after wildly misinterpreting clues, but ultimately prevailing against its formidable human opponents.



IBM's Watson soundly defeated the two most successful contestants in the history of the game show "Jeopardy!," Ken Jennings and Brad Rutter, in a three-day competition in February.

Leading up to the million-dollar challenge, Watson played more than 50 practice matches against former "Jeopardy!" contestants, and was required to pass the same tests that humans must take to qualify for the show and compete against Jennings, who broke the "Jeopardy!" record for the most consecutive games played, resulting in winnings of more than $2.5 mil-

PHOTO COURTESY IBM

lion, and Rutter, whose total winnings amounted to $3.25 million, the most money ever won by a single "Jeopardy!" player. At the end of the three-day event, Watson finished with $77,147, beating Jennings, who had $24,000, and Rutter, who had $21,600. The million-dollar prize money awarded to Watson went to charity.

Named after IBM founder Thomas J. Watson, the Watson system was built by a team of IBM scientists whose goal was to create a standalone platform that could rival a human's ability to answer questions posed in natural language. During the "Jeopardy!" challenge, Watson was not connected to the Internet or any external data sources. Instead, Watson operated as an independent system contained in several large floor units housing 90 IBM Power 750 servers with a total of 2,880 processing cores and 15 terabytes of memory. Watson's technology, developed by IBM and several contributing universities, was guided by principles described in the Open Advancement of Question-Answering (OAQA) framework, which is still operating today and facilitating ongoing input from outside institutions.

Judging by the sizeable coverage of the event, Watson piqued the interest of technology enthusiasts and the general public alike, earning "Jeopardy!" the highest viewer numbers it had achieved in several years and leading to analysts and other industry observers speculating about whether Watson represents a fundamental new idea in computer science or merely a solid feat of engineering. Richard Doherty, the research director at Envisioneering Group, a technology consulting firm based in Seaford, NY, was quoted in an Associated Press story as saying that Watson is "the most significant breakthrough of this century."

Doherty was not alone in making such claims, although the researchers on the IBM team responsible for designing Watson have been far more modest in their assessment of the technology they created. "Watson is a novel approach and a powerful architecture," says David Ferrucci, director of the IBM DeepQA research team that created Watson. Ferrucci does characterize Watson as a breakthrough in artificial intelligence, but he is careful to qualify this assertion by saying that the breakthrough is in the development of artificial-intelligence systems.

"The breakthrough is how we pulled everything together, how we integrated natural language processing, information retrieval, knowledge representation, machine learning, and a general reasoning paradigm," says Ferrucci. "I think this represents a breakthrough. We would have failed had we not invested in a rigorous scientific method and systems engineering. Both were needed to succeed."

Contextual Evidence

The DeepQA team was inspired by several overarching design principles, with the core idea being that no single algorithm or formula would accurately understand or answer all questions, says Ferrucci. Rather, the idea was to build Watson's intelligence from a broad collection of algorithms that would probabilistically and imperfectly interpret language and score evidence from different perspectives. Watson's candidate answers, those answers in which Watson has the most confidence, are produced from hundreds of parallel hypotheses collected and scored from contextual evidence.

Ferrucci says this approach required innovation at the systems level so individual algorithms could be developed independently, then evaluated for their contribution to the system's overall performance. The approach allowed for loosely coupled interaction between algorithm components, which Ferrucci says ultimately reduced the need for team-wide agreement. "If every algorithm developer had to agree with every other or reach some sort of consensus, progress would have been slowed," he says. "The key was to let different members of the team develop diverse algorithms independently, but regularly perform rigorous integration testing to evaluate relative impact in the context of the whole system."

Ferrucci and the DeepQA team are expected to release more details later this year in a series of papers that will outline how they dealt with specific aspects of the Watson design. For now, only bits and pieces of the complete picture are being disclosed. Ferrucci says that, looking ahead, his team's research agenda is to focus on how Watson can understand, learn, and interact more effectively. "Natural language understanding remains a tremendously difficult challenge, and while Watson demonstrated a powerful approach, we have only scratched the surface," he says. "The challenge continues to be about how you build systems to accurately connect language to some representation, so the system can automatically learn from text and then reason to discover evidence and answers."

Lillian Lee, a professor in the computer science department at Cornell University, says the reactions about Watson's victory echo the reactions following Deep Blue's 1997 victory over chess champion Garry Kasparov, but with several important differences. Lee, whose research focus is natural



Watson's on-stage persona simulates the system's processing activity and relative answer confidence through moving lines and colors. Watson is shown here in a practice match with Ken Jennings, left, and Brad Rutter at IBM's Watson Research Center in January.

PHOTO COURTESY IBM

language processing, points out that some observers were dismissive about Deep Blue's victory, suggesting that the system's capability was due largely to brute-force reasoning rather than machine learning. The same criticism, she says, cannot be leveled at Watson because the overall system needed to determine how to assess and integrate diverse responses.

"Watson incorporates machine learning in several crucial stages of its processing pipeline," Lee says. "For example, reinforcement learning was used to enable Watson to engage in strategic game play, and the key problem of determining how confident to be in an answer was approached using machine-learning techniques, too."

Lee says that while there has been substantial research on the particular problems the "Jeopardy!" challenge involved for Watson, that prior work should not diminish the team's accomplishment in advancing the state of the art to Watson's championship performance. "The contest really showcased real-time, broad-domain question-answering, and provided as comparison points two extremely formidable contestants," she says. "Watson represents an absolutely extraordinary achievement."

Lee suggests that with language-processing technologies now maturing, with the most recent example of such maturation being Watson, the field appears to have passed through an important early stage. It now faces an unprecedented opportunity in helping sift through the massive amounts of user-generated content online, such as opinion-oriented information in product reviews or political analysis, according to Lee.

While natural-language processing is already used, with varying degrees of success, in search engines and other applications, it might be some time before Watson's unique question-answering capabilities will help sift through online reviews and other user-generated content. Even so, that day might not be too far off, as IBM has already begun work with Nuance Communications to commercialize the technology for medical applications. The idea is for Watson to assist physicians and nurses in finding information buried in medical tomes, prior cases, and the latest science journals. The first commercial offerings from the collaboration are expected to be available within two years.

"Natural language understanding remains a tremendously difficult challenge, and while Watson demonstrated a powerful approach, we have only scratched the surface," says David Ferrucci.

Beyond medicine, likely application areas for Watson's technology would be in law, education, or the financial industry. Of course, as with any technology, glitches and inconsistencies will have to be worked out for each new domain. Glitches notwithstanding, technology analysts say that Watson-like technologies will have a significant impact on computing in particular and human life in general. Ferrucci, for his part, says these new technologies likely will mean a demand for higher-density hardware and for tools to help developers understand and debug machine-learning systems more effectively. Ferrucci also says it's likely that user expectations will be raised, leading to systems that do a better job at interacting in natural language and sifting through unstructured content.

To this end, explains Ferrucci, the DeepQA team is moving away from attempting to squeeze ever-diminishing performance improvements out of Watson in terms of parsers and local components. Instead, they are focusing on how to use context and information to evaluate competing interpretations more effectively. "What we learned is that, for this approach to extend beyond one domain, you need to implement a positive feedback loop of extracting basic syntax and local semantics from language, learning from context, and then interacting with users and a broader community to acquire knowledge that is otherwise difficult to extract," he says. "The system must be able to bootstrap and learn from its own failing with the help of this loop."

In an ideal future, says Ferrucci, Watson will operate much like the ship computer on "Star Trek," where the input can be expressed in human terms and the output is accurate and understandable. Of course, the "Star Trek" ship computer was largely humorless and devoid of personality, responding to queries and commands with a consistently even tone. If the "Jeopardy!" challenge serves as a small glimpse of things to come for Watson—in particular, Watson's precise wagers, which produced laughter in the audience, and Watson's visualization component, which appeared to express the state of a contemplative mind through moving lines and colors—the DeepQA team's focus on active learning might also include a personality loop so Watson can accommodate subtle emotional cues and engage in dialogue with the kind of good humor reminiscent of the most personable artificial intelligences in fiction. C

Further Reading

Baker, S.
Final Jeopardy: Man vs. Machine and the Quest to Know Everything. **Houghton Mifflin Harcourt, New York, NY, 2011.**

Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A.A., Lally, A., Murdock, J.W., Nyberg, E., Prager, J., Schlaefer, N., and Welty, C.
Building Watson: An overview of the DeepQA project, *AI Magazine 59*, Fall 2010.

Ferrucci, D., et al.
Towards the Open Advancement of Question Answering Systems. *IBM Research Report RC24789 (W0904-093)*, April 2009.

Simmons, R.F.
Natural language question-answering systems, *Communications of the ACM 13*, 1, Jan. 1970.

Strzalkowski, T., and Harabagiu, S. (Eds.)
Advances in Open Domain Question Answering. **Springer-Verlag, Secaucus, NJ, 2006.**

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

© 2011 ACM 0001-0782/11/07 $10.00

Technology | DOI:10.1145/1965724.1965731 Alex Wright

Automotive Autonomy

Self-driving cars are inching closer to the assembly line, thanks to promising new projects from Google and the European Union.

At the 1939 World's Fair, General Motors' fabled Futurama exhibit introduced the company's vision for a new breed of car "controlled by the push of a button." The self-driving automobile would travel along a network of "magic motorways" outfitted with electrical conductors, while its occupants would glide along in comfort without ever touching the steering wheel. "Your grandchildren will snap across the continent in 24 hours," promised Norman Bel Geddes, the project's chief architect.

Seventy years later, those grandchildren are still waiting for their self-driving cars to roll off the assembly lines. Most analysts agree that commercially viable self-driving cars remain at least a decade away, but the vision is finally coming closer to reality, thanks to the advent of advanced sensors and onboard computers equipped with increasingly sophisticated driving algorithms.

In theory, self-driving cars hold out enormous promise: lower accident rates, reduced traffic congestion, and improved fuel economy—not to mention the productivity gains in countless hours reclaimed by workers otherwise trapped in the purgatory of highway gridlock. Before self-driving cars make it to the showroom, however, car manufacturers will need to clear a series of formidable regulatory and manufacturing hurdles. In the meantime, engineers are making big strides toward proving the concept's technological viability.

The European Union-sponsored SARTRE project is developing technologies to allow cars to join organized platoons, with a lead car operated by a human driver.



One of Google's seven self-driving, robotic Toyota Priuses steers its way through a tight, closed circuit course.

For the past year, Bay Area residents have noticed a fleet of seven curious-looking Toyota Priuses outfitted with an array of sensors, sometimes spotted driving the highways and city streets of San Francisco, occasionally even swerving their way down the notoriously serpentine Lombard Street.

Designed by Sebastian Thrun, director of Stanford University's AI Laboratory currently on leave to work at Google, the curious-looking Priuses could easily be mistaken for one of Google's more familiar Street View cars. The Googlized Prius contains far more advanced technology, however, including a high-powered Velodyne laser rangefinder and an array of additional radar sensors.

The Google car traces its ancestry to Thrun's previous project, the Stanley robot car, which won the U.S. Defense Advanced Research Project Agency's (DARPA's) $2 million grand challenge prize after driving without human assistance for more than 125 miles in desert conditions. That project caught the attention of executives at Google, who have opened the company's deep pockets to help Thrun pursue his research agenda.

At Google, Thrun has picked up where the Stanley car left off, refining the sensor technology and driving algorithms to accommodate a wider range of potential real-world driving

PHOTOGRAPH BY STEVE JURVETSON

conditions. The Google project has made important advances over its predecessor, consolidating down to one laser rangefinder from five and incorporating data from a broader range of sources to help the car make more informed decisions about how to respond to its external environment.

"The threshold for error is minuscule," says Thrun, who points out that regulators will likely set a much higher bar for safety with a self-driving car than for one driven by notoriously error-prone humans. "Making a car drive is fundamentally a computer science issue, because you're taking in vast amounts of data and you need to make decisions on that data," he says. "You need to worry about noise, uncertainty, what the data entails." For example, stray data might flow in from other cars, pedestrians, and bicyclists—each behaving differently and therefore requiring different handling.

Google also has a powerful tool to help Thrun improve the accuracy of his driving algorithms: Google Maps. By supplementing the company's publicly available mapping data with details about traffic signage, lane markers, and other information, the car's software can develop a working model of the environment in advance. "We changed the paradigm a bit toward map-based driving, whereby we don't drive a completely unknown, unrehearsed road," Thrun explains. Comparing real-time sensor inputs with previously captured data stored at Google enables the car's algorithms to make more informed decisions and greatly reduce its margin of error.

Although the trial runs are promising, Thrun acknowledges that the cars must be put through many more paces before the project comes anywhere close to market readiness. He freely admits the Google car is a long way from rolling off an assembly line. "We are still in a research stage," says Thrun, "but we believe that we can make these cars safer and make driving more fun."

At press time, Google had hired a lobbyist to promote two robotic car-related bills to the Nevada legislature. One bill, an amendment to an existing electric vehicle law, would permit the licensing and testing of self-driving cars. The second is an exemption to allow texting during driving.

"Making a car drive is fundamentally a computer science issue," says Sebastian Thrun, "because you're taking in vast amounts of data and you need to make decisions on that data."

Europe's Car Platoons

If the Google project ultimately comes to fruition, it may do more than just improve the lives of individual car owners; it could also open up new possibilities for car sharing and advanced "highway trains" in which cars follow each other on long-distance trips, improving fuel efficiency and reducing the cognitive burden on individual drivers.

Researchers in Europe are pursuing just such an approach, developing a less sophisticated but more cost-efficient strategy in hopes of bringing a solution to market more quickly. The European Union-sponsored SARTRE project is developing technologies to allow cars to join organized platoons, with a lead car operated by a human driver. Ultimately, the team envisions a Web-based booking service that would allow drivers of properly equipped vehicles to search for nearby platoons matching their travel itineraries.

Two earlier European projects successfully demonstrated the viability of this approach using self-driving trucks. SARTRE now hopes to build on that momentum to prove the viability of the concept for both consumer and commercial vehicles.

By limiting the project's scope to vehicles traveling in formation on a highway, the project team hopes to realize greater gains in fuel economy and congestion reduction than would be possible with individual autonomous cars. "We wanted to drive these vehicles very close together because that's where we get the aerodynamic gains," says project lead Eric Chan, a chief engineer at Ricardo, the SARTRE project's primary contractor.

By grouping cars into platoons, the SARTRE team projects a 20% increase in collective fuel efficiency for each platoon. If the project ultimately attracts European drivers in significant numbers, it could also eventually begin to exert a smoothing effect on overall traffic flow, helping to reduce the "concertina effect," the dreaded speed-up and slow-down dynamic that often creates congestion on busy highways.

To realize those efficiency gains, the SARTRE team must develop a finely tuned algorithm capable of keeping a heterogeneous group of cars and trucks moving forward together in near-perfect lockstep. "The closer together, the less time you have to respond to various events," says Chan, "so cutting down latency and response times is critical." To achieve that goal, the system enables the vehicles to share data with each other on critical metrics like speed and acceleration.

Chan says the team's biggest technological hurdle has been developing a system capable of controlling a vehicle at differing speeds. "When you're controlling the steering system at low speed versus high speed, the dynamics of the vehicle behave differently," Chan says. "You have to use the controls in a slightly different way. At high speeds the vehicle dynamics become quite different and challenging."

In order to keep the platoon vehicles in sync at varying speeds, the team has developed a system that allows the vehicles to communicate directly with each other as well as with the lead vehicle. The systems within the lead vehicle act as a kind of central processor, responsible for managing the behavior of the whole platoon. The space between each vehicle is controlled by the system depending on weather or speed, but the lead driver can also exert additional influence through manual overrides.

In hopes of bringing the solution to market within the next few years, the SARTRE team is focused on developing with relatively low-cost systems and sensors that are production-level or

close to it, as opposed to the more expensive, laser-scanning sensors used in the Google and DARPA projects.

The larger challenge for the SARTRE project may have less to do with sensors and algorithms than with addressing the potential adoption barriers that might prevent consumers from embracing the platoon concept. After all, part of the appeal of driving a car lies in the freedom to go where you want, when you want. But will drivers be willing to adjust their driving behavior in exchange for the benefits of a kind of quasi-public transportation option?

"There's a big human factors aspect to this project," says Chan, who acknowledges that predicting market acceptance is a thorny issue. The team has been trying to understand the psychological impact of autonomous driving on the human occupants formerly known as drivers. The developers have been running trials with human subjects to see how people react to different gap sizes between cars, trying to identify potential psychological issues that could affect users' willingness to relinquish control of their vehicles. "How comfortable do people feel driving a short distance from another car?"

A human factors issue for the SARTRE project is whether consumers will embrace its car platoon concept.

asks Chan. "How much control should the operator really have?"

The team is also considering the potential impact on other drivers outside the platoon, since the presence of a long train of vehicles will inevitably affect other traffic on the freeway. For example, if the platoon is traveling in the slow lane on a multilane freeway, it will inevitably have to react to occasional interlopers.

Whether consumers will ultimately embrace self-driving cars will likely remain an open question for years to come, but in the meantime the underlying technologies will undoubtedly undergo further refinement. For the next few years, self-driving cars will continue to remain the province of researchers, while the rest of us can only dream of someday driving the magic motorway to Futurama.

Further Reading

Albus, J, et al.
4D/RCS: A Reference Model Architecture for Unmanned Vehicle Systems 2.0. NIST interagency/internal report, NISTIR 6910, Aug. 22, 2002.

O'Toole, R.
Gridlock! Why We're Stuck in Traffic and What to do About It. Cato Institute, Washington, D.C., 2010.

Robinson, R., Chan, E., and Coelingh, E.
Operating platoons on public motorways: An introduction to the SARTRE platooning program, 17th World Congress on Intelligent Transport Systems, Busan, Korea, Oct. 25–29, 2010.

Thrun, S. et al.
Stanley: The robot that won the DARPA grand challenge," *Journal of Field Robotics 23*, 9, Sept. 2006.

Thrun, S.
What we're driving at, The Official Google Blog, Oct. 9, 2010.

Alex Wright is a writer and information architect based in Brooklyn, NY.

© 2011 ACM 0001-0782/11/07 $10.00

Public Policy

U.S. Calls for Global Cybersecurity Cooperation

Whether it's thieves trading in stolen credit card information, spammers planting malicious code on computer networks, or hostile governments hacking into sensitive systems, cybersecurity is a growing issue in an increasingly networked world. In late May, for instance, the world's largest defense contractor, Lockheed Martin, announced it had been the target of a "significant and tenacious attack" on its Maryland-based servers. One result is the Obama administration is calling for an international effort to strengthen global cybersecurity. In a strategy report released in May, the White House called for governments to work together to develop standards that ensure privacy and the free flow of information while preventing theft of information or attacks on systems.

"We know that the Internet is changing, becoming less American-centric and maybe more dangerous. This lays out a path to make it more secure while preserving important values like openness and connectivity," says James Lewis, director of the Technology and Public Policy Program at the Center for International Strategic Studies. "Most importantly, it reverses our old policy of wanting unilateral 'domination' and replaces it with engagement with other countries, consistent with the Obama national security strategy."

During President Obama's visit to the United Kingdom on May 25, he and Prime Minister David Cameron issued a joint statement pledging cooperation on cybersecurity. They also announced that the U.K. had signed on to the Budapest Convention on Cybercrime, a treaty signed by the U.S. and 30 other countries. The U.S. strategy calls for expanding the convention's reach.

Fred Cate, a law professor and director of the Center for Applied Cyber Security Research at Indiana University Bloomington, says the administration deserves credit for taking a first step, but doesn't feel the proposal goes very far. "I think we'd like to have seen more, not just detail, but also a more aggressive strategy."

He says domestic law provides almost no incentive to take even the simplest steps toward better security, such as shipping cable modems with a firewall turned on by default. If there were a system of domestic legal liabilities, tax credits, and safe harbor provisions for companies to engage in good practices—the sort of mix of regulations and incentives that apply to health-care and financial institutions—that would give the country a good starting point for better international policies, Cate says. Even a requirement to report cyberattacks to a central clearinghouse, so companies and institutions could learn from others' experiences, would be useful.

"Right now we don't know how many cyber events there are," Cate says.

On the other hand, the U.S. Chamber of Commerce worries that the regulation could have a negative effect on business. "Layering new regulations on critical infrastructure will harm public-private partnerships, cost industry substantial sums, and not necessarily improve national security," the U.S. Chamber of Commerce said in a response to the domestic policy proposal.

—Neil Savage

Society | DOI:10.1145/1965724.1965732 Dennis McCafferty

Brave, New Social World

How three different individuals in three different countries—Brazil, Egypt, and Japan—use Facebook, Twitter, and other social-media tools.

TODAY, SOCIAL MEDIA IS emerging as a dominant form of instant global communication. Growing more addictively popular by the day—nearly two-thirds of Internet users worldwide use some type of social media, according to an industry estimate—Facebook, Twitter, and other easily accessible online tools deepen our interaction with societies near and far.

Consider these numbers: Facebook is poised to hit 700 million users and, as seven of 10 Facebook members reside outside the U.S., more than 70 global-language translations. Twitter's user numbers will reportedly hit 200 million later this year, and users can tweet in multiple languages. In terms of daily usage, Facebook generates the second-most traffic of any site in the world, according to Alexa.com, a Web information company, at press time. (Google is number one.) As for blogging, which now seems likes a relatively old-fashioned form of social media, the dominant site, blogger.com, ranks eighth. As for Twitter, it's now 11th—and climbing.

Nearly two-thirds of Internet users worldwide use some type of social media, according to an industry estimate.



A protestor's sign thanks the youth of Egypt and Facebook during the political unrest in Egypt in late January. The photo, by an NBC foreign correspondent, first appeared on Twitter.

The top five nations in terms of social media usage are the U.S., Poland, Great Britain, South Korea, and France, according to the Pew Research Center. But beyond international rankings and traffic numbers, there's much diversity in the manner in which the citizens of the world take advantage of these tools, according to *Blogging Around the Globe: Motivations, Privacy Concerns and Social Networking*, an IBM Tokyo research report. In Japan, blogs often serve as outlets for personal expression and diary-style postings. In the U.S., it's mostly about earning income or promoting an agenda. In the U.K., it's a combination of these needs, as well as professional advancement and acting as a citizen journalist.

Communications connected with three citizens in three different nations, each of whom are finding their own individual voice through these resources. In fact, we depended primarily upon social media to initially reach them. One is a Japanese female blogger who segues seamlessly from pop-culture observations to revealing reflections on the nation's recent earthquake, tsunami, and nuclear disaster. Another is a Brazilian businesswoman who uses multiple digital outlets to expand her marketing reach throughout the world. The third is an Egyptian newsman who is helping record history with his dispatches of daily life in a region undergoing dramatic political change. (In terms of social media usage, Brazil ranks eighth, Japan 12th, and Egypt 18th, according to Pew.) Here are their stories.

Me and Tokyo

The contrast is striking: Before March 11, Mari Kanazawa's blog, Watashi to

PHOTOGRAPH TWEETED BY RICHARD ENGEL NBC ON FRIDAY JAN 28, 2011 WITNESS.ORG

Tokyo (translation: Me and Tokyo), waxes whimsically about a recent tweet in Japanese by the band Radiohead, as well as consumer products such as Wasasco, a wasabi-flavored Tabasco.

After March 11, however, the conversation takes an abrupt turn. The day after the devastating Tōhoku earthquake and tsunami, Kanazawa writes this unsettling passage: "Earthquake, tsunami, fire and now we have a nuclear meltdown ... I was in the Midtown Tower when it happened. Japanese people are used to earthquakes, we can usually sense them because the building sways, but this time it was shaking up and down. Some people screamed and some hid under their desks."

Within a week, Kanazawa casts a sense of humor about the situation: "I really don't need to check Geiger counters and don't need a lot of toilet paper because earthquakes [don't] make me [go to the bathroom] more than usual."

A high-profile cyberpersonality in Japan, Kanazawa has always perceived her blog as equal parts diary and cultural commentary. She was one of the rare Japanese citizens who wrote a blog in English when she started in 2004, so her traffic numbers have spiked to a healthy 2,000 unique visitors a day. A Web site manager, Kanazawa prefers the free-form creativity of a blog, as opposed to the restrictive 140-character count of Twitter. "It doesn't fit me," she says of the latter. "My blog is an information hub for Japanese subculture. That's my style. I wanted to tell people that we have more interesting, good things than sushi, sumo, tempura, geishas, and ninjas."

Since the disaster, like many Japanese citizens posting blogs and Facebook status updates, Kanazawa has sought and published information about the nation's recovery efforts. "These tools are so effective in this disaster," she says. "People need to check for things such as the transportation situation and where the evacuation areas are. In Tōhoku, when someone tweeted 'We need 600 rice balls here,' they were delivered within an hour. Social media went from being a communication tool to a lifeline."

Brazil—and Beyond

In generations past, it would be difficult for a self-described life coach like Lygya Maya of Salvador, Brazil, to interact with a motivational-speaking giant like Tony Robbins, an American who has more than 200 books, audio CDs, and other products listed on Amazon.com. Perhaps she would have needed to take a trip to the U.S. in hopes of speaking with Robbins at one of his tour stops. Or write him a letter and hope he would answer with something beyond a polite thank you.

But this is the 21st century, and Maya takes full advantage of the digital age to engage with high-profile leaders such as Robbins and Mark Victor Hansen, co-author of the bestselling *Chicken Soup for the Soul* books. Robbins and Hansen are now Facebook friends with Maya, who they have advised and encouraged to push beyond perceived limitations in her work.

Such international collaborations have enabled Maya to create her own signature style to market herself, which she calls a "Brazilian Carnival Style" approach to guide clients to enjoying a happy, productive, and empowering life. Maya now sees up to 300 clients a year in private sessions, and hosts as many as 500 group sessions annually.

"I use blogs, Facebook, Twitter, and Plaxo [an online address book] to promote my business," Maya says. "I am about to start podcasting, as well as making YouTube videos on every channel that I can find on the Internet. Social media has opened up my business on many different levels. I am now able to promote it literally to the world, free of charge."

Maya has also established more than 2,500 personal connections via Facebook, LinkedIn, and other sites. She'll send tweets several times a day, offering reflections like "When truthfully expressed, words reflect our core value and spirit." All of this has helped Maya promote her budding empire of services and products, which will soon include a book, *Cheeka Cheeka BOOM Through Life!: The Luscious Story of a Daring Brazilian Woman*. It's gotten to the point where—like some of her counterparts in the U.S.—she must subcontract work just to keep up with it all.

"I'm about to hire a team to work with me on Twitter and all the social media out there that we can use to support campaigns," Maya says. "You must have a great team to share quality work. Otherwise, you will have stress. This allows me to promote my services and products 24/7—and that includes while I'm sleeping."

A Witness in Egypt

Amr Hassanein lists *Babel*, *Fantasia*, and *The Last Temptation of Christ* as his favorite movies on his Facebook page. And his organizations/activities

Blogs: Motivations for writing and readership levels by region.

Region	Motivation	Readership
Japan	Personal diary, self-expression	74% Internet users, average 4.54 times/week, 25% daily, highest in world
Korea	Personal diary, personal scrapbook, online journalism	43% Internet users , average 2.03 times/week, ages 8–24: 4 times/week ages 25–34: 3 times/week
China	96% personal blogs loaded with photos, audio, animations	Highest for ages 18–24 (less than 3 times/week), probably friends
U.S.	Make money, promote political or professional agenda	27% Internet users, average 0.9 times/week, lower than Asia, higher than Europe
Germany	For fun, like to write, personal diary	Bloggers are regular readers of other blogs on average 21.15 (std dev 39, med 10)
U.K.	Connect with others, express opinions/vent, make money, citizen journalist, validation, professional advancement	23% Internet users (average 0.68 times/week)
Poland	Self-expression, social interaction, entertainment	Not available

Source: Mei Kobayashi, *Blogging Around the Globe: Motivations, Privacy Concerns and Social Networking*, IBM Research-Tokyo, 2010.

"Social media makes me feel like an observer," says Amr Hassanein. "It gives me a sense of what's going on around me at all times."

of interest include Hands Along the Nile Development Services, a nonprofit organization that promotes intercultural understanding between the U.S. and his native Egypt. Now working as a freelance producer for ABC News, Hassanein is also using Facebook as a vehicle to showcase his own firsthand accounts of political unrest in the Middle East. Recently, for example, ABC sent him to Libya to assist with news coverage of the nation's conflict.

"My usage of social media tools is from a neutral side," says Hassanein, sounding very much like an objective news reporter. "Social media makes me feel like an observer. It gives me a sense of what's going on around me at all times. The impact events here in Egypt, like the demonstrations, were organized and known through Facebook."

Still, it's impossible to live through these times without getting caught up in the politics. His sympathies remain with We Are All Khaled Said, an anti-torture group that uses social media to allow voices of the Arab uprisings to be heard. (Sample Facebook post from the group: "Gaddafi has vowed it will be a 'long war' in Libya. Let's hope his [sic] wrong & Gaddafi's massacre of his people will end very soon.")

Hassanein recognizes that social media provides an opportunity to deliver an unfiltered message to the world about local developments, as well as debunk stereotypes about people of the Middle East. Yet, aside from this bigger-picture purpose, these tools allow him to easily remain in close contact with loved ones and work associates.

Actions taken by the Egyptian government to block access to Facebook and Twitter significantly backfired during its recent conflict, further fueling the resolve of the freedom movement, he says. "The impact was clear: What were normal demonstrations became a revolution. It made me think about the consequences of blocking people from information."

That said, some of the "anything goes" aspects of social media make Hassanein feel uncomfortable. "When you watch a news channel that presents a direction you don't like," he says, "you have the ability not to watch. In social media, there is no uni-direction you can refuse or reject. People are the senders and the receivers. Inputs need to be self-filtering and self-censoring. For me, I will use my head."

Further Reading

Hilts, A., and Yu, E.
Modeling social media support for the elicitation of citizen opinion, *Proceedings of the International Workshop on Modeling Social Media*, Toronto, Canada, June 13–16, 2010.

Kärkkäinen, H., Jussila, J., and Väisänen, J.
Social media use and potential in business-to-business companies' innovation, *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments*, Tampere, Finland, Oct. 6–8, 2010.

Kobayashi, M.
Blogging around the globe: Motivations, privacy concerns and social networking, *Computational Social Networks*, Abraham, A., (Ed.), Springer-Verlag, London, England, forthcoming.

Leskovec, J.
Social media analytics: Tracking, modeling and predicting the flow of information through networks, *Proceedings of the 20th International Conference Companion on World Wide Web*, Hyderabad, India, March 28–April 1, 2011.

Mehlenbacher, B., McKone, S., Grant, C., Bowles, T., Peretti, S., and Martin, P.
Social media for sustainable engineering communication, *Proceedings of the 28th ACM International Conference on Design of Communication*, São Carlos-São Paulo, Brazil, Sept. 26–29, 2010.

Dennis McCafferty is a Washington, D.C.-based technology writer.

© 2011 ACM 0001-0782/11/07 $10.00

In Memoriam

Max Mathews, 1926–2011

Max Mathews, often referred to as the father of computer music, died on April 21 in San Francisco at the age of 84 from pneumonia. In 1957, as an engineer at Bell Laboratories, Mathews wrote the world's first program for playing synthesized music on a computer. The 17-second composition—played on an IBM 704 mainframe—served as a foundation for much of today's music.

"Mathews was above all a visionary and an innovator," says Michael Edwards, program director for the School of Arts, Culture and Environment at the University of Edinburgh (and author of this month's cover story; see p. 58). "His legacy is felt every day."

In the 1960s, Mathews' work at Bell Labs helped develop advanced music and voice synthesis systems. A decade later, he developed Groove, the first computer system designed for live performances. It spawned other commercial programs, including Csound, Cmix, and MAX (named after him), which remain in use today.

In the 1970s, he assisted with the development of the Institut de Recherche et Coordination Acoustique/Musique in Paris, a center devoted to research in the science of music and sound .

"Mathews established the 'unit generator' paradigm of computer music applications, and despite the incredible speed of development in technology, this is still with us," Edwards notes. "Although many are quick to dismiss computer music as something inhuman or arcane, most music, regardless of genre, has been created with the aid of computers since the 1980s."

Mathews also invented musical instruments, including the Radio Baton, a pair of handheld wands that control the tempo and balance of electronic music through hand and arm gestures, and several electric violins.

At the time of his death, Mathews was a music professor in the Center for Computer Research in Music and Acoustics at Stanford University.

—Samuel Greengard

Milestones | DOI:10.1145/1965724.1965733

ACM Award Recipients

Craig Gentry, Kurt Mehlhorn, and other computer scientists are honored for their research and service.

ACM RECENTLY ANNOUNCED the winners of six prestigious awards for innovations in computing technology that have led to practical solutions to a wide range of challenges facing commerce, education, and society.

Craig Gentry, a researcher at IBM, was awarded the Grace Murray Hopper Award for his breakthrough construction of a fully homomorphic encryption scheme, which enables computations to be performed on encrypted data without unscrambling it. This long-unsolved mathematical puzzle requires immense computational effort, but Gentry's innovative approach broke the theoretical barrier to this puzzle by double encrypting the data in such a way that unavoidable errors could be removed without detection.

Kurt Mehlhorn, founding director of the Max Planck Institute for Informatics and a professor at Saarland University, was awarded the Paris Kanellakis Theory and Practice Award for contributions to algorithm engineering that led to creation of the Library of Efficient Data Types and Algorithms (LEDA). This software collection of data structures and algorithms, which Mehlhorn developed with Stefan Näher, provides practical solutions for problems that had previously impeded progress in computer graphics, computer-aided geometric design, scientific computation, and computational biology.

GroupLens Collaborative Filtering Recommender Systems received the ACM Software System Award. These systems show how a distributed set of users could receive personalized recommendations by sharing ratings, leading to both commercial products and extensive research. Based on automated collaborative filtering, these recommender systems were introduced, refined, and commercialized by a team at GroupLens. The team then brought automation to the process, enabling wide-ranging research and commercial applications. The GroupLens team includes **John Riedl**, University of Minnesota; **Paul Resnick**, University of Michigan; **Joseph A. Konstan**, University of Minnesota; **Neophytos Iacovou**, COVOU Technologists; **Peter Bergstrom**, Fluke Thermography; **Mitesh Suchak**, Massachusetts Institute of Technology; **David Maltz**, Microsoft; **Brad Miller**, Luther College; **Jon Herlocker**, VMware, Inc.; **Lee Gordon**, Gordon Consulting, LLC; **Sean McNee**, FTI Consulting, Inc.; and **Shyong (Tony) K. Lam**, University of Minnesota.



IBM researcher Craig Gentry, recipient of the Grace Murray Hopper Award.

Takeo Kanade, the U.A. and Helen Whitaker University Professor of Computer Science and Robotics at Carnegie Mellon University, is the recipient of the ACM/AAAI Allen Newell Award for contributions to research in computer vision and robotics. His approach balanced fundamental theoretical insights with practical, real-world applications in areas like face and motion detection and analysis, direct drive manipulators, three-dimensional shape recovery from both stereo vision and motional analysis, and video surveillance and monitoring.

Barbara Ericson, who directs the Institute for Computing Education at Georgia Tech, and **Mark Guzdial**, director of the Contextualized Support for Learning at Georgia Tech, received the Karl V. Karlstom Outstanding Educator Award for their contributions to broadening participation in computing. They created the Media Computation (MediaComp) approach, which motivates students to write programs that manipulate and create digital media, such as pictures, sounds, and videos. Now in use in almost 200 schools around the world, MediaComp's contextualized approach to introductory computer science attracts students not motivated by classical algorithmic problems addressed in traditional CS education.

Reinhard Wilhelm and **Joseph S. DeBlasi** were named recipients of the Distinguished Service Award. Wilhelm, scientific director of the Schloss Dagstuhl–Leibniz Center for Informatics, was honored for two decades of exceptional service at the center, creating a stimulating environment for advancing research in informatics. Wilhelm brought together researchers from complementary computing areas for intensive workshops that promoted new research collaborations and directions. DeBlasi, former executive director of ACM, was honored for his executive leadership from 1989–1999 that transformed ACM into a financially sound, globally respected institution, and for his foresight in implementing programs and expanding international initiatives that continue to sustain ACM today.

© 2011 ACM 0001-0782/11/07 $10.00

PHOTOGRAPH BY STEVE MOORS FOR TECHNOLOGY REVIEW

DOI:10.1145/1965724.1965734

Mari Sako

Technology Strategy and Management

Driving Power in Global Supply Chains

How global and local influences affect product manufacturers.

Supply chains are increasingly global. Consequently, we pour energy into managing existing global supply chains efficiently, with their risks (for example, risks arising from geographic dispersion) and rewards (such as the benefits derived from cost arbitrage). Yet we do not know enough about how profits are divided and distributed along a global supply chain that changes over time. This is a question worth posing at a time when new locations have become available not only for production but also for consumption, especially in rapidly growing emerging markets. For example, if the end market for electronic goods shifts from the U.S. to China or India, would the supply chain become driven by global or local corporate entities?

Any supplier to a famous brand, be it Apple or Nike, knows all too well that the corporate client does not need ownership to exert power over the supplier. In this world of corporate control without ownership, what opportunities exist for creating and capturing profit in global supply chains? By comparing the evolution of major players across different industries and service sectors, this column addresses the question: under what circumstances do value-adding activities migrate from the final product manufacturer to a component manufacturer? What strategies are available to the final product manufacturer to circumvent this migration of power in global supply chains?



ILLUSTRATION BY ANDRIJ BORYS ASSOCIATES

What We Already Know

Many readers of this column are likely familiar with the fate of IBM. In its initial era of dominance, IBM was a classic vertically integrated company. But faced with competition in the personal computer market, IBM decided it could not keep up on all fronts and outsourced its operating system to Microsoft and its microprocessors to Intel in the 1980s. This was the beginning of the end of IBM as a computer hardware company. With IBM's outsourcing decisions, new players came to occupy horizontal industry segments—Microsoft in operating systems and applications software, Intel in microprocessors, and Compaq and HP in IBM-compatible final assembly. Technological advances in subsystems made it more profitable to make microprocessors and software

than hardware. The "Intel Inside" platform strategy to extract high profits extended from desktop computers to notebook PCs with the launch of integrated chipsets.[3]

Was this horizontally disintegrated structure stable? No. Companies sought opportunities to capture greater profits, not only by specializing in focused technologies but also by bundling products and services. In particular, Microsoft strengthened its market power by bundling its operating system with applications software, Web browser, and networked services. In this competitive landscape, IBM withdrew from hardware by selling its PC division to Lenovo, and struck out for new territory in business services.

A similar cycle of moving from vertical integration to horizontal disintegration and back again to reintegration is evident in the evolution of Apple to become the world's most valuable technology company in terms of stock market value in May 2010.[1] In the 1980s, Apple Computers was a vertically integrated firm with its own in-house design and factories. The troubles in the 1990s culminated in Apple's decision to outsource final assembly to SCI Systems in 1996, laying the groundwork for modular thinking. The iPod is a prototypical modular product, enabling Apple to mix and match preexisting components. By leading in product innovation and design, but without doing any manufacturing, Apple pocketed $80 in gross profit for each 30GB iPod sold at $299.[2] The ongoing transformation of Apple Inc., bundling the iPod, iTunes, iPhone, and iPad, is a dramatic example of a company that has been able to reinvent itself by taking advantage of global supply chains. Innovative companies such as Apple have the power to reshape the boundaries of the industries in which they operate.

Thus, we know that value migrates from the final product manufacturer to component suppliers as a result of the former's outsourcing decisions and the pursuit of platforms by the latter. However, this could be reversed or circumvented if the product manufacturer regains control of its supply chain by reshaping its industry and developing an ecosystem of providers engaged in complementary innovation.

Important though this story is, there is a less well-known story behind this one, focused around the no-brand supply companies that actually make these products.

A Bit of History: The Rise and Rise of Large Factories

In the 19th century, improvements in transportation (especially railroads) and communication (such as telegraphs) led to the development of mass markets. By the early 20th century, such markets demanded large volumes of standardized products, exemplified by Ford's Model T, produced in large vertically integrated factories. Fast-forward into the early 21st century, and we see the current wave of improvements in transportation (this time in container shipping) and communication (this time with digital technology) have had a similar impact on the size of factory operations.[4] We see the rise of large horizontally integrated production factories in low-cost locations supplying products and services to the world.

Consider the case of athletic shoemaking. Several powerful brand owners exist in an oligopolistic market. But today, the largest footwear manufacturer in the world is not one of the brand owners such as Nike or Adidas, but Pou Chen Group. Its shoemaking subsidiary, Yue Yuen Industrial Ltd., has a sales turnover of $5.8 billion, employs around 300,000 workers, and churns out 186 million pairs of shoes per annum. That is, this company makes one in every six pairs of athletic shoes sold in the world.

Another good example is in laptop computers. In this market, Quanta Computer is the world's largest manufacturer. One in every three laptops is made by Quanta. Its factories make laptop computers for brand owners ranging from Apple, Compaq, Dell, Fujitsu, HP, Lenovo, Sharp, Sony, and Toshiba. One thing it does not do is produce its own brand of computers. Quanta Computer is the largest of the Taiwanese personal computer manufacturers, whose combined output accounts for over 90% of worldwide market share.

Under what circumstances do value-adding activities migrate from the final product manufacturer to a component manufacturer?

Similarly, Hon Hai Precision Industry Co. (Foxconn) heads the league table of electronic manufacturing service (EMS) providers, which include such firms as Flextronics, Jabil Circuit, Celestica, and Sanmina SCI. Having achieved a very rapid growth, FoxConn employs nearly one million workers mostly in China to assemble Apple's iPod, iPhone and iPad, cellphones for Nokia and Motorola, Nintendo's video game consoles, and Sony's PlayStation, among other things.

"Behind-the-Scenes Champions" Profit from Size and Diversification

These companies—Pou Chen, Quanta, Foxconn—are no-brand manufacturing firms that supply retailers or brand-owning firms, some with no factories. They are called CM (contract manufacturers) or ODM (original design manufacturers) if they undertake design as well as the manufacture of products for sale under the client's brand. The brand owners may command and drive power in global supply chains, but the behind-the-scene supply firms have not been totally powerless. The most obvious source of bargaining power for these no-brand suppliers is the sheer size of the operation. For example, Quanta Computer supplies nine out of the world's top 10 notebook PC brands. As such, it exercises power by being discriminating among these clients, setting up dedicated business units with product development and mass production capacity for some of the best (but not all) clients.

A small number of ODMs, such as Acer and Lenovo, transitioned to selling products with their own brand. However, turning your corporate client into a competitor is a risky move, as Lenovo initially found out with IBM when it terminated its contract with Lenovo. As an alternative strategy,

therefore, no-brand contract manufacturers turn to various modes of diversifying into related areas. Pou Chen Group went into the manufacturing of LCDs and later into retailing; Flextronics went into electronic repair.

A similar logic applies not only to manufacturing but also to services. In professional services, in particular, intangibles such as brand and reputation count for a lot in driving power in global supply chains. In management consulting, for example, the likes of McKinsey and Bain have outsourced business research, while in financial services, investment banks outsource and offshore financial research and analytics. With the disintegration in global supply chains, so-called knowledge process outsourcing (KPO) providers, such as Genpact and Evalueserve, have been pursuing strategies in three steps. They consist of climbing up, scaling up, and broadening out.

First, just as CM evolved into ODM, KPO suppliers have "climbed up the value chain" by providing higher value-adding services. This may involve writing an entire research report on the basis of business research for a consulting client or on the basis of the analysis of a valuation model for an investment-banking client; the clients then put their own brand onto the report. Second, KPO suppliers have also scaled up their operations, investing heavily not only in IT infrastructure but also in process and quality improvements for their "information processing" factories. Third, some KPO suppliers have pursued a diversification strategy by bundling different professional services, for example by pulling together business, financial, and legal research under one roof.

Shifting the End Market

Competing head-to-head with brand owners in established developed economy markets seems incredibly difficult in many cases. However, when the end market shifts from old to new emerging markets, this dynamic may change. For example, when cellphones are intended for purchase in China rather than in the U.S. or Europe brands matter less for the mass low-end market. This creates certain advantages for indigenous firms within global supply chains.

Companies pursue similar strategies in their attempt to drive power in global supply chains.

A decade of growth has made China by far the largest mobile phone handset market in the world, with over 800 million users in early 2011. Moreover, China has emerged as the largest exporter of mobile handsets. Initially, in the 2G market, foreign brands such as Nokia worked closely with chipset manufacturers (for example, Texas Instruments) to design handsets, which were in turn assembled by contract manufacturers such as Flextronics. In China, indigenous local firms' initial point of entry was not in assembly/manufacturing, but in sales and marketing for the local market. By being closer to the ultimate market than foreign brands, these firms evolved into independent design houses (IDH), with better knowledge of Chinese consumers' preferences in styling and the agility to respond quickly to the market. IDHs undertake the development of handsets from highly modularized components. Modularization was further enhanced in the transition to 3G multimedia phones for low-end markets, with MediaTek, a Taiwan-based chip design firm, providing an integrated chipset module that incorporated multimedia functions such as music and video players.[5]

Thus, when the end market shifts to emerging markets, we observe a "reverse pattern" in the way foreign firms and local firms interact to occupy different parts of the global supply chain. Traditionally, consumers for products made with global supply chains were in high-income locations, and low-income locations were for manufacturing. Also, local firms positioned themselves in global supply chains by doing assembly, leaving marketing to brand-owning foreign firms. But when emerging economies serve not only as manufacturing locations but also as huge consumer markets, local firms' competitive advantage lies in sales and marketing, tailoring products to local markets using modular components. Foreign firms may of course respond by investing in sales and marketing to meet the ultimate demand for "good enough" products.

Conclusion

What the economist Joseph Schumpeter wrote a century ago is still relevant today: discontinuous change happens as a result of five things: the introduction of a new product or process, the opening of a new market or source of supply of intermediate goods, and a new organization design.[6] Economic globalization, as typified by the rise of global supply chains, involves all the Schumpeterian forces. Although differences remain across sectors, companies pursue similar strategies in their attempt to drive power in global supply chains.

In particular, the final product manufacturer drives power typically by owning a brand, initiating innovation, and controlling the supply chain. However, value may migrate from the final product system manufacturer to component suppliers, if suppliers create significant value in their components and find horizontal markets to sell them. Beyond this, this column highlighted the role of two other significant entities that have come out to play the power game: the sizeable no-brand suppliers who climb up, scale up, and diversify and the indigenous emerging market operators that focus on local sales and marketing. C

References

1. Cusumano, M. Platforms and services: Understanding the resurgence of Apple. *Commun. ACM 53*, 10 (Oct, 2010), 22–24.
2. Dedrick, J. et al. Who profits from innovation in global value chains? A study of the iPod and notebook PCs. *Industrial and Corporate Change 19*, 1 (Jan. 2001), 81–116.
3. Gawer, A., Ed. *Platforms, Markets, and Innovation*, Edward Elgar, 2009.
4. Helper, S. and Sako, M. Managing innovation in supply chain: Appreciating Chandler in the twenty-first century. *Industrial and Corporate Change 19*, 2 (Feb. 2010), 399–429.
5. Kawakami, M. and Sturgeon, T. *The Dynamics of Local Learning in Global Value Chains: The Experiences from East Asia*. Palgrave Macmillan, 2011.
6. Schumpeter, J.A. *The Theory of Economic Development: An Inquiry into Profits, Capital, Credit, Interest, and the Business Cycle*. Galaxy Books, 1912/1934.

Mari Sako (mari.sako@sbs.ox.ac.uk) is Professor of Management Studies at Saïd Business School, University of Oxford, U.K.

Copyright held by author.

DOI:10.1145/1965724.1965735

Cory Knobel and Geoffrey C. Bowker

Computing Ethics
Values in Design

Focusing on socio-technical design with values as a critical component in the design process.

Values often play out in information technologies as disasters needing management. When Facebook started sharing data about what people were buying or viewing, it ended up with digital egg all over its face. Focusing the initial design process on complicated values of privacy might have helped Facebook avoid this uproar. To use another example, the "terms and conditions" that most users simply "accept" without reading could be made easier to read and understand if the values inherent in fair contracting were incorporated in the design of such agreements in the first place. But conversations and analyses of the values found in technologies are generally engaged after design and launch, and most users are faced with a daunting set of decisions already made on their behalf (and often not to their benefit) and impossible choices if they would like to do things differently. Sensible responses to this problem have been developed over the past 10 years, and a community of researchers has formed around the role of human values in technology design.[a] A new book on Values in Design from the MIT Press *Infrastructures* series illustrates the issues.

Helen Nissenbaum has created a Values in Design Council, working with the National Science Foundation on the Futures of Internet Architecture (FIA) research program (see http://www.nyu.edu/projects/nissenbaum/vid_council.html). This suite of projects is aimed at redesigning Internet architecture to handle ever-expanding modes of usage with fewer problems due to design mistakes about values. An initial meeting of people from these projects revealed three values that need immediate attention. One involves the trade-off between security and privacy: for example, can we design computing "clouds" so that search queries cannot be traced to an individual user or IP except in carefully controlled circumstances subject to appropriate prior review. Not surprisingly, the U.S. National Security Agency wants to maintain loopholes that allow it to pursue the important value of national security. Can these values be reconciled through a compromise design? Another involves hardwire design for Digital Rights Management (DRM) that protects digital rights while permitting flexibility as information policy evolves. A third concern, "cultural valence," means systems designed by one group (for example, Americans) should not impose American values about structure, protocol, use, and policy on non-Americans as Internet architectures go global. The point is not that designers have the *wrong* values, but that one of the key features of values is that different people hold different values, and often hold to those values very strongly.

a Examples of existing work in along this theme include Batya Friedman's *values-sensitive design*, Mary Flanagan and Helen Nissenbaum's *Values at Play*, Phoebe Sengers' *reflective design*, T.L. Taylor's *values in design* in ludic systems, and Ann Cavoukian's *privacy by design*.

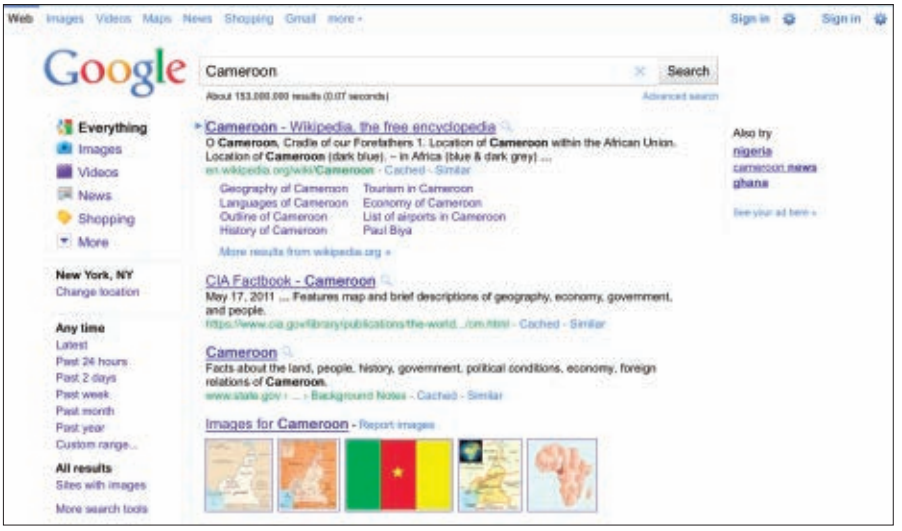


Figure 1. Results of a Google search on "Cameroon."

Infrastructures and Values

Successful infrastructures serve people with different values. A good example of this is mobile technologies.



Figure 2. Designer Mary Flanagan's reconceptualized classic Atari video games with giant joystick; http://www.maryflanagan.com/giant-joystick.

Inclusion of GPS capability creates new opportunities regarding information tied to geography. Mobile applications coupled to social networks allow users to know when they are near friends. Loopt and FourSquare[b] show where friends have "checked in" and their distances from a user's current location to facilitate social gathering and serendipitous meeting. However, such technologies can cause tension in social values as the benefit of potential meetings with friends causes problems of attention and interrogation, as when a paramour says, "You said you were going to the store, then the library, and then home, but you never checked in. *Where were you?*" a GPS-based network applications may increase locational accountability because, unlike a phone call that might originate anywhere, GPS-enabled applications carry information about specific geographic location. In principle, a user can work around "stalking" and other problematic situations with some mobile apps such as Tall Tales and Google Latitude[c] that allow a user to lie about location, but equating privacy with lying creates its own values-centric problems. An "open hand" of location-based transparency can easily become a "backhand" when geographic privacy and autonomy are compromised.

Another good example of value clashes concerns search engines. Google might be the greatest information retrieval tool in world history, but it falls prey to the "Matthew effect" named for a line in the Gospel of Matthew (25:29): "For to all those who have, more will be given, and they will have an abundance; but from those who have nothing, even what they have will be taken away." The results of a simple Google search on the word "Cameroon" shown in Figure 1 indicate Wikipedia, the CIA, the U.S. State Department, and the BBC seem to know more about Cameroon than any of its inhabitants. The highest-ranked site from the country does not appear until page 4, a link to the country's main newspaper. Given that most users never go beyond the first few links,[d] few will get to information about Cameroon *from* Cameroon. The country is officially French-speaking, so sophisticated searchers might find better results searching for "Cameroun," but few English-speaking users would do this. The algorithm that provides nearly universal access to knowledge also unwittingly suppresses knowledge of African countries. Or is this always unwitting? A search on "Obamacare" produces a taxpayer-paid-for link to http://www.healthcare.gov as a top hit.[1]

Interdisciplinary Scholars

A community of scholars has formed around VID, or Values in Design (or more formally, Values in the Design of Information Systems and Technology). It consists of researchers and practitioners in computer science, engineering, human-computer interaction, science and technology studies, anthropology, communications, law, philosophy, information science, and art and design. They find common ground through the interdisciplinarity implied by the broad spectrum of interests. Decades of research in the sociology of science and technology have shown that technical infrastructures reveal human values most often through counterproductivity, tension, or failure. Workshops conducted over the past six years by Helen Nissenbaum, Geoffrey Bowker, and Susan Leigh Star have sparked conversations among people in these fields, producing a cohort of interdisciplinary scholars of values in design. This group departs from a traditional view of critical theory that tackles technology once it is in place, and focuses instead on socio-technical design with values as a critical component in the design process. The objective of VID is to create infrastructures that produce less friction over values than those created in the past. This objective is timely given the rise of social computing and networks, games that address social problems and change (see http://www.gamesforchange.org/) and the interconnection of corporate, government, and academic institutions' interests ranging from the individual to the transglobal.

b With over 4 million and 6.5 million registered users as of February 2011, respectively; see http://about.loopt.com/tag/loopt/ and http://foursquare.com/about.

c http://itunes.apple.com/us/app/tall-tales-geolocation-spoofing/; http://mashable.com/2009/02/04/google-latitude/; http://www.androidzoom.com/android_applications/fake%20locations

d http://seoblackhat.com/2006/08/11/tool-clicks-by-rank-in-google-yahoo-msn/

ACM Transactions on Accessible Computing



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

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

Association for Computing Machinery

VID aims to create a new field that understands values and technology in the early stages of design.

The VID program depends on having sensible definitions for the terms "values" and "design." The philosopher of science Jacob Metcalf provides a useful framing for "values" by comparing it with generally well-understood concept of *ethics*. Ethics are a set of prescriptions (nouns), while values are tied to action (verbs). VID is a call to action, an effort in "verbing" design work by practical exercises. Exercises include background readings in computer and information science and information policy literatures. Exercises also include use of Mary Flanagan's *Values at Play* cards to modify or create new values-driven computer games, or Batya Friedman's *Envisioning* cards to reveal social sensitivities during the design process.[e] Workshop groups are split into interdisciplinary teams that produce a values-driven design in about one week. Academics and industry experts judge the proposals, which have ranged from a system to support community gardening projects and green space development, to a geocaching system that reveals the geographic routes by which kidnapped women are trafficked into the sex trade.

Conclusion

The VID effort has been under way for 15 years since first articulated,[2–4] and for six years at building the cadre of scholars through workshops. Next steps are to open the design space through collaborative interdisciplinary work; in contrast to customary university training that teaches how to work individually. The world demands skills in collaboration, and future designers must work in highly connected and intellectually fertile environments. The VID community sees design as a process in which constraints impose new directions for innovation, and values are a source of constraints. The VID community is rethinking design to go beyond user studies, marketing, documentation, programming, and implementation to actual engagement where the rubber truly hits the road. Design must be integrated in ways that challenge assumptions about what can and cannot be changed. A newly forming Center for Values in Design at the University of Pittsburgh's School of Information Sciences will explore and apply these ideas as they emerge (see http://vid.pitt.edu/).

Design must be integrated in ways that challenge assumptions about what can and cannot be changed.

To close, consider the work of theorist, artist, and designer Mary Flanagan (see http://www.maryflanagan.com/giant-joystick). She has reconceptualized classic Atari video games by replacing the single-user, joystick-and-fire button control with a 10-foot high mechanism that requires collaboration and coordination among several people to operate the game (see Figure 2). She subverts design by taking a nontraditional perspective that produces radical reinterpretations of everyday practice. She shows that the social values of collaboration, cooperation, coordination, and play can transform a taken-for-granted utility, the simple joystick, into an opportunity for engagement and discourse about the design of information technologies. C

e http://www.valuesatplay.org/?page_id=6 and Friedman, B., Nathan, L. P., Kane, S., and Lin, J. *Envisioning Cards. Value Sensitive Design Research Lab*. The Information School, University of Washington. Seattle, WA, 2011; http://www.envisioningcards.com

References

1. Diaz, A. Through the Google Goggles: Sociopolitical bias in search engine design. In *Web Search: Multidisciplinary Perspectives*. A. Spink and M. Zimmer, Eds., Springer New York, 2008.
2. Flanagan, M., Howe, D., and Nissenbaum, H. *Values in Design: Theory and Practice*. Working Paper, 2005.
3. Friedman, B., and Nissenbaum, H. Bias in computer systems. *ACM Transactions on Information Systems 14*, 3 (1996), 330–347.
4. Sengers, P., Boehner, K., David, S., and Kay, J. Reflective design. In *Proceedings of the 4th Decennial ACM Conference on Critical Computing*. (2005).

Cory Knobel (cknobel@sis.pitt.edu) is an assistant professor at the University of Pittsburgh.

Geoffrey C. Bowker (gbowker@sis.pitt.edu) is a professor and senior scholar in Cyberscholarship at the University of Pittsburgh.

Copyright held by author.

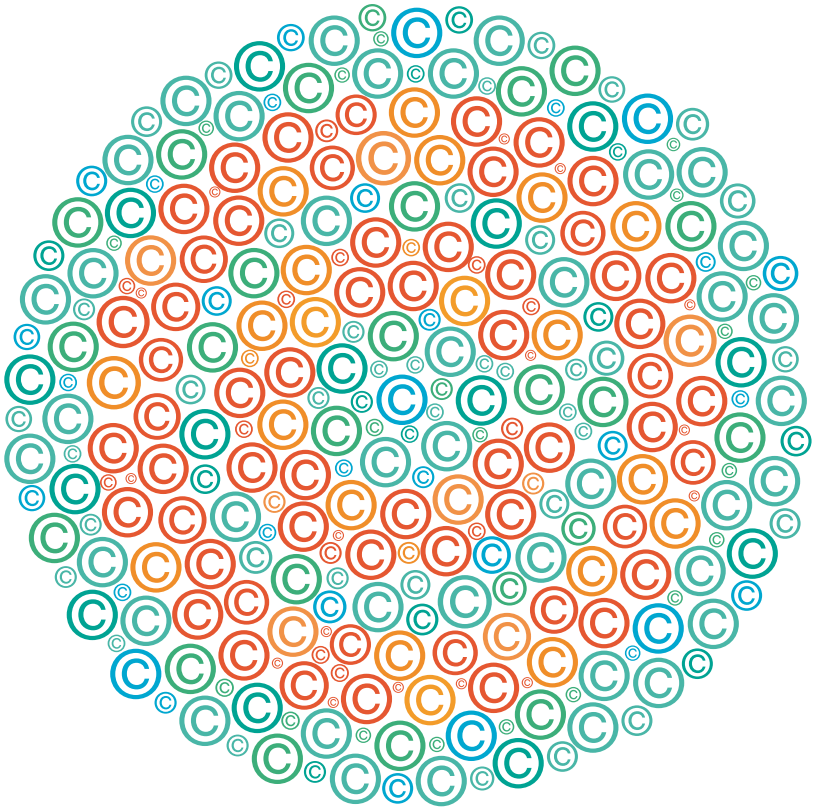


DOI:10.1145/1965724.1965736

Pamela Samuelson

Legally Speaking
Too Many Copyrights?

Reinstituting formalities—notice of copyright claims and registration requirements—could help address problems related to too many copyrights that last for too many years.



Virtually all of the photographs on flickr, videos on YouTube, and postings in the blogosphere, as well as routine business memos and email messages, are original works of authorship that qualify for copyright protection automatically by operation of law, even though their authors really do not need copyright incentives to bring these works into being. Yet, copyrights in these works, like those owned by best-selling authors, will nonetheless last for 70 years after the deaths of their authors in the U.S. and EU (and 50 years post-mortem in most other countries).

Are there too many copyrights in the world, and if so, what should be done to weed out unnecessary copyrights? Some copyright scholars and practitioners who think there are too many copyrights are exploring ways of limiting the availability of copyright to works that actually need the exclusive rights that copyright law confers.[1,3,4]

Copyright Formalities as an Opt-In Mechanism

One obvious way to eliminate unnecessary copyrights is to require authors who care about copyright to register their claims, put copyright notices on copies of their works, and/or periodically renew copyrights after a period of years instead of granting rights that attach automatically and last far beyond the commercial life of the overwhelming majority of works.

Copyright lawyers speak of such requirements as "formalities," for they make the enjoyment or exercise of copyright depend on taking some steps to signal that copyright protection is important to their creators.[4]

Conditioning the availability of copyright on formalities is not exactly a new idea. For most of the past 300 years, copyright was an opt-in system. That is, copyright protection did not commence when a work was created; authors had to opt-in to copyright by registering their works with a central office or by putting copyright notices on copies of their works sold in the market. When authors failed to comply with formalities, the works were generally in the public domain, freely available for reuses without seeking any permission. This enriched culture because these works were available for educational uses, historical research, and creative reuses.

While many countries abandoned formality requirements in the late 19th and early 20th centuries, the U.S. maintained notice-on-copies and registra-

ILLUSTRATION BY ALICIA KUBISTA

tion-for-renewal formalities until 1989. The U.S. still requires registration of copyrights as a precondition for U.S. authors to bring infringement actions, as well as for eligibility for attorney fee and statutory damage awards.

Formalities do a good job weeding out who really cares about copyrights and who doesn't. So why did the U.S. abandon formalities?

Formalities Abandoned

The U.S. had no choice but to abandon copyright formality requirements in the late 1980s because it wanted to exercise leadership on copyright policy in the international arena.

Then and now the only significant international forum for copyright policy discussions was the Berne Union. It is comprised of nations that have agreed to abide by provisions of an international treaty known as the Berne Convention for the Protection of Literary and Artistic Works. Article 5(2) of this treaty forbids member states from conditioning the enjoyment or exercise of copyrights on formalities, such as those long practiced in the U.S.

The Berne Union was first founded in the late 19th century, at a time when the U.S. had little interest in international copyrights. By the mid-1980s, however, U.S. copyright industries were the strongest and most successful in the world. They had become not only significant contributors to the gross domestic product, but also a rapidly growing exporter of U.S. products. This made them care about the copyright rules adopted in other countries.

In the late 1980s, these industries persuaded one of their own—President Ronald Reagan—that the U.S. needed to join the Berne Convention in order to exercise influence on international copyright policy. And so in 1989, under Reagan's leadership, the U.S. joined the Berne Convention and abandoned the notice-on-copies and registration requirements that had served the nation well since its founding.

Are there too many copyrights in the world, and if so, what should be done to weed out unnecessary copyrights?

Why Is Berne Hostile to Formalities?

In the late 1880s when the Berne Union was first formed, each of the 10 participating countries had its own unique formality requirements for copyright protection. One of the goals of the Berne Union was to overcome obstacles to international trade in copyrighted works such as burdens of complying with multiple formalities.

The initial solution to the problem of too many formalities was a Berne Convention rule that provided if an author had complied with formalities of his/her own national copyright law, other Berne Union countries would respect that and not insist on compliance with their formality requirements.

That was a reasonably good solution as far as it went, but it created some confusion. It was sometimes unclear, for instance, whether works of foreign authors sold in, say, France, had complied with the proper formalities in the works' country of origin. If a work was simultaneously published in two countries, was the author required to comply with two sets of formalities or only one of them? It was also difficult for a publisher to know whether a renewal formality in a work's country of origin had been satisfied.

In part because of such confusions, the Berne Convention was amended in 1908 to forbid Berne Union members from conditioning the enjoyment and exercise of copyright on compliance with formalities.

While the main reason for abandoning formalities was pragmatic, another factor contributing to the abandonment of formalities was the influence in Europe of a theory that authors had natural rights to control the exploitation of their works. Sometimes this theory was predicated on the labor expended by authors in creating their works, and sometimes on the idea that each work was a unique expression of the author's personality that deserved automatic respect from the law.

In the absence of organized constituencies in favor of preserving formalities, the natural rights theory of copyrights prevailed in much of Europe, and with it, the idea that formalities were inconsistent with the natural rights of authors in their works.

Because the Berne Convention's ban on formalities has been incorporated by reference into another major international treaty, the Agreement on Trade-Related Aspects of Intellectual Property Rights (widely known as the TRIPS Agreement), it would seem the world is now stuck with a no-formality copyright regime. But should it be so?

Has Technology Changed the Formalities Equation?

In recent decades, two major changes have contributed to a renewed interest in copyright formalities.

One is that advances in information technologies and the ubiquity of global digital networks have meant that more people than ever before are creating and disseminating literary and artistic works, many of which are mashups or remixes of existing works.

A second is that the Internet and Web have made it possible to establish scalable global registries and other information resources that would make compliance with formalities inexpensive and easy (at least if competently done), thereby overcoming the problems that led to the Berne Convention ban on formalities.

Lawrence Lessig, among others,[1,3] has argued that reinstituting copyright formalities would be a very good idea. This would enable free reuses of many existing commercially fallow works that would contribute to and build on our cultural heritage. It would also help libraries and archives to preserve that part of our cultural heritage still in-copyright and to provide access to works of historical or scientific interest now unavailable because of overlong copyrights. Many innovative new services could be created to facilitate new insights and value from existing works, such as those contemplated in the Google Book Search settlement (for example, nonconsumptive research services to advance knowledge in hu-

manities as well as scientific fields).

Copyright formalities serve a number of positive functions.[4] They provide a filter through which to distinguish which works are in-copyright and which are not. They signal to prospective users that the works' authors care about copyright. They provide information about the work being protected and its owner through which a prospective user can contact the owner to obtain permission to use the work. And by enabling freer uses of works not so demarked, formalities contribute to freer flows of information and to the ongoing progress of culture.

One recent report[2] has recommended that the U.S. Copyright Office should develop standards for enabling the creation of multiple interoperable copyright registries that could serve the needs of particular authorial communities, while also serving the needs of prospective users of copyrighted works by providing better information about copyright ownership and facilitating licensing. Perhaps unregistered works should receive protection against wholesale copying for commercial purposes, while registered works might qualify for a broader scope of protection and more robust remedies.

Conclusion

Copyright industry representatives frequently decry the lack of respect that the public has for copyrights. Yet, in part, the public does not respect copyright because some aspects of this law don't make much sense.

An example is the rule that every modestly original writing, drawing, or photograph that every person creates is automatically copyrighted and cannot be reused without permission for 100 years or more (depending on how long the author lives after a work is created).

If too many works are in-copyright for too long, then our culture suffers and we also lose the ability to distinguish in a meaningful way between those works that need copyright protection and those that don't.

This column has explained that formalities in copyright law serve a number of positive functions and has argued that reinstituting formalities would go a long way toward addressing the problems arising from the existence of too many copyrights that last for too many years. Obviously the new formalities must be carefully designed so they do not unfairly disadvantage authors and other owners.

Copyright industry representatives frequently decry the lack of respect that the public has for copyrights. Yet, in part, the public does not respect copyright because some aspects of this law don't make much sense.

Although the obstacles to adoption of reasonable formalities may be formidable, they are surmountable if the will can be found to overcome them and if the technology infrastructure for enabling them is built by competent computing professionals. One intellectual obstacle to reinstituting formalities is addressed in a forthcoming book,[4] which explains that formality requirements are more consistent with natural rights theories than many commentators have believed. Treaties can be amended and should be when circumstances warrant the changes.

References

1. Lessig, L. *The Future of Ideas: The Fate of the Commons in a Connected World.* Random House, New York, 2001.
2. Samuelson, P. et al. The Copyright Principles Project: Directions for reform. *Berkeley Technology Law Journal 25*:0000 (2010).
3. Springman, C. Reform(aliz)ing copyright. *Stanford Law Review 57*:568 (2004).
4. van Gompel, S. *Formalities in Copyright Law: An Analysis of their History, Rationales and Possible Future.* Kluwer Law International, Alphen aan den Rijn, The Netherlands, forthcoming 2011.

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

Copyright held by author.

Calendar of Events

July 17–21
International Symposium on Software Testing and Analysis, Toronto, Canada, Sponsored: SIGSOFT, Contact: Matthew B. Dwyer, Email: dwyer@cse.unl.edu, Phone: 402-472-2186

July 18–21
International Conference on e-Business, Seville, Spain, Contact: David A. Marca, Email: dmarca@openprocess.com, Phone: 617-641-9474

July 18–21
International Conference on Security and Cryptology, Seville, Spain, Contact: Pierangela Samarati, Email: pierangela.samarti@unimi.it, Phone: +39-0373-898-061

July 18–21
International Conference on Signal Processing and Multimedia Applications, Seville, Spain, Contact: Mohammad S. Obaidat, Email: obaidat@monmouth.edu, Phone: 201-837-8112

July 18–21
International Symposium on Smart Graphics, Bremen, Germany, Contact: Rainer Malaka, Email: malaka@tzi.de, Phone: +49-421-21864402

July 20–22
Symposium on Geometry Processing 2011, Lausanne, Switzerland, Contact: Mark Pauly, Email: mark.pauly@epfl.ch

July 22–24
International Conference on Advances in Computing and Communications, Kochi, India, Contact: Sabu M. Thampi, Email: smtlbs@in.com

July 25–27
19th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Singapore, Contact: Cai Wentong, Email: aswtcai@ntu.edu.sg

DOI:10.1145/1965724.1965737

Maria (Mia) Ong

Broadening Participation
The Status of Women of Color in Computer Science

Addressing the challenges of increasing the number of women of color in computing and ensuring their success.

TO REMAIN ECONOMICALLY and globally competitive, the U.S. needs to increase its advanced domestic science and technology work force.[1] As U.S. colleges are already majority female and are increasingly enrolling more minority students, women of color represent a growing potential source of domestic talent to meet the needs of the country. Thus, it is in the interest of all of us to ensure that women of color are well represented in science, technology, engineering, and mathematics (STEM) fields.

There is also the social justice argument for promoting women of color in STEM. The history of exclusion in science and technology fields and in the U.S. at large has resulted in an unfortunate outcome of underrepresentation that should be actively addressed. It is important to continue to recognize and challenge sexism and racism that remains pervasive—though perhaps more subtle than 30 years ago—and which is experienced by women of color in multiplicative ways. Moreover, women of color are often the breadwinners, main supporters of children, and community leaders, so their successes and failures in a well-paid and well-respected field such as computer science could have significant impacts on more general community issues.

As the accompanying table shows, the current outlook presents challenges for addressing the need of attracting and retaining women, especially women of color, into computing. Among U.S. citizens and permanent residents receiving 2008 degrees in the computer sciences, women of color fared worse compared to their White female counterparts at both the bachelor's and Ph.D. levels. Within every racial group, men outearned women in terms of CS degrees awarded, with two exceptions: Blacks at the Ph.D. level, where both men and women both earned 12 degrees, and American Indian/Alaska Natives at the Ph.D. level, where men and women both earned no degrees.[2]

Of serious concern is the decline of Hispanic women earning Ph.D.s in CS. An examination of doctorate attainment over the past decade reveals that their numbers peaked in 2004 at nine Ph.D.s but have declined since, and they received only two of the CS Ph.D.s awarded in 2008. Of continuing disquiet is the status of American Indian/Alaska Native women in CS. Between



The Spelman College Spelbots provide hands-on robotics education and research for women computer science students by competing in U.S. and International RoboCup 4-Legged competitions.

PHOTOGRAPH COURTESY OF ANDREW WILLIAMS

2000 and 2008 this group has only earned *a total of seven* Ph.D.s.[2]

The "Inside the Double Bind" Study

Policies aimed at increasing women of color in computing should be based on empirical research on this population. Unfortunately, not much research exists. While there have been many studies since 1970 on the experiences of *women* in STEM and on those of *minorities* in STEM, the unique experiences of women of color, who encounter the challenges of race and gender simultaneously, are often excluded from the research agenda. Studies that do exist have been difficult to find because they are scattered throughout journals, book chapters, reports, and unpublished dissertations.

The NSF-funded project, "Inside the Double Bind: A Synthesis of Empirical Literature on Women of Color in STEM," aimed to gather, analyze, and synthesize empirical research that had been produced between 1970 and 2008. The project team, co-led by Gary Orfield (UCLA) and myself, identified 116 works of empirical research literature produced since 1970 on women of color in STEM higher education and careers. The resulting "Inside the Double Bind" synthesis[3,4] highlights general empirical findings and identifies research gaps in STEM. Specific findings on women of color in computer science are summarized here.

We identified 19 sources on women of color in computer science—not many at all, considering that our search covered nearly 40 years' worth of literature. Studies in computing are relatively new: 16 of the works have been produced since 2002. Most of the literature focuses on higher education, and the research covers an array of topics, including the "digital divide" that separates girls and women of color from others, social challenges for women of color students, the roles of minority-serving institutions, and nontraditional pathways to CS degrees. The reader should be forewarned that our searches were thorough but not exhaustive, and with only 19 identified works, there are many gaps and incomplete descriptions about the status and experiences of women of color in computing. Some policy implications and future directions for research in this area are discussed later in this column.

Preparation and the "digital divide." Several research studies pointed to the "digital divide" that leaves girls and women of color underexposed to technology and basic computer skills in their upbringing. The underexposure, researchers claim, may be due to a number of factors, including socioeconomic inequalities and gendered beliefs that females lack potential for technical fluency. This divide can put them at a disadvantage compared to their White and male peers in knowledge and in comfort in dealing with computers, thus hindering their entry and retention into computer science fields.

Social challenges for women of color in CS. Fields that are heavily White and male, such as physics, engineering, and computer science, pose some unique social challenges for women of color students. At predominantly White institutions (PWIs), they often experience being the only woman or minority—or, at most, one of a few—in their class or laboratory. Research suggests that in CS, their sense of isolation is often heightened by what they perceive as an unwelcoming environment and others' lowered expectations of them. In my current study, a comment by a young professional woman of color who had majored in computer science provides a vivid illustration of this experience: *In my computer science class, a lot of the projects were group [work] and so I found two... [minority] groupmates, who were heaven-sent. And we stuck by each other and actually, after we found each other, planned all of our schedules in sync with each other, so we took the same classes in order to get through the undergraduate experience together. Because a part of being a minority is that people don't want to work with you. They don't look at you and sense that you are a smart person they want to work with. So finding people who believe in you and you believe in, and then sticking together, was really important.* ("Serena," in Ong and Hodari.[5])

This woman's strategy of working with other minorities helped her to persist through her undergraduate program, but sadly, the cumulative social challenges she encountered ultimately deterred her from pursuing computer science in graduate school or as a career. This story of attrition is far too common. Fortunately, though, an increasing number of organizations and CS departments are putting tremendous amounts of time and energy to establish more welcoming social environments for all of their members.

Family and school balance. There is a serious dearth of research about family-school and family-work balance for women of color in STEM and in CS, but what we've learned so far is worth noting. The few studies we identified on the topic reveal that a com-

Computer sciences degrees awarded to U.S. citizens and permanent residents (2008).

	Bachelor's Degrees	Ph.D.s
Female	**6,473 (17.4%)**	**153 (22.9%)**
White	3,235 (8.7%)	89 (13.3%)
Asian/Pacific Islander	597 (1.6%)	17 (2.5%)
Black	1,338 (3.6%)	12 (1.8%)
Hispanic	551 (1.5%)	2 (0.3%)
American Indian/Alaska Native	55 (0.1%)	0 (0.0%)
Other or unknown race/ethnicity	697 (1.9%)	33 (4.9%)
Male	**30,639 (82.6%)**	**514 (77.1%)**
White	19,954 (53.8%)	357 (53.5%)
Asian/Pacific Islander	2,536 (6.8%)	70 (10.5%)
Black	2,673 (7.2%)	12 (1.8%)
Hispanic	2,372 (6.4%)	14 (2.1%)
American Indian/Alaska Native	166 (0.4%)	0 (0.0%)
Other or unknown race/ethnicity	2,938 (7.9%)	61 (9.1%)

Source: National Science Foundation, 2011. Note: Percentages reflect the proportion of the total number of CS bachelor's degrees and Ph.D.s awarded, respectively, to U.S. citizens and permanent residents.

mon challenge for women of color students involves tensions between their demanding CS programs and external pressures to manage and participate in the family structure and to contribute to the family income. Exacerbating the issue are rigid course schedules, faculty who do not understand the cultural expectations upon these students, family members who do not understand the time commitment required to pursue a computer science degree, and lack of job opportunities for students in CS-related fields.

The role of minority-serving institutions. Minority-serving institutions (MSIs), including Historically Black Colleges and Universities (HBCUs), Hispanic-serving institutions (HSIs), and Tribal Colleges and Universities (TCUs), have a strong history of producing a disproportionate number of minority female STEM majors who continue on to Ph.D.s. The field of computer science is no exception. While more research is needed in this area, especially for HSIs and TCUs, existing research attributes the persistence of women of color in CS to MSIs' nurturing environments, faculty who believe in their students, a collaborative peer culture, and special programs such as summer research experiences. Researchers also credit the persistence of women of color in computing to the personal drive of the women themselves.

Nontraditional pathways. More than their White female counterparts, women of color take nontraditional paths to computer science. Many come to CS education later in their lives, long after leaving school with non-CS degrees or no degree at all, and perhaps after starting a family or working full-time. Many begin their computer science education in community colleges, and while some directly transfer afterward to a four-year institution, others periodically "stop out," taking months or years off before returning to study. Studies reveal that persistence through programs by nontraditional women of color result from a combination of individuals' drive for economic and academic success and programs that accommodate and encourage them. More research is needed in this area to address profiles of nontraditional students, academic programs and activities that attract and retain them, and types of degrees and employment they gain.

Future research needs to address educational and career choices and career trajectories of women of color.

Policy Implications and Future Directions for Research

The existing research indicates some potential, immediate steps for institutional policy and action. To help women of color traverse the digital divide and feel they belong in CS, institutions might offer real-world opportunities to gain computer expertise—and thereby a sense of empowerment—in the classroom. They could also provide meaningful and well-paid CS-related employment, such as research and tutoring opportunities, and develop and sustain a supportive learning community that includes women of color and other marginalized students. Practices of organizations and departments that have already made great strides in this area should be documented, widely disseminated, and adapted by others. Further, institutions should explore ways to adapt some practices of MSIs and programs that successfully serve nontraditional students in computer science. To address tensions between family and academic demands, departments might offer more flexibility in their programs, including offering some online courses and scheduling courses more than once a year; allow for a fully integrated, part-time academic track; and increase the number of CS research stipends and work opportunities. Finally, high-level recognition of the many accomplishments of women of color in computing should be given, so that these women may serve as role models to girls and young women of color who may follow in their footsteps.

New research will reveal effective ways to bring more women of color into the field. Future studies should include women in *all* racial/ethnic groups, but especially for those groups about whom information is scarce: Latinas/Hispanics, American Indians/Alaska Natives, and Asian Americans/Pacific Islanders. Future research needs to address educational and career choices and career trajectories of women of color, and more should be learned about the paths of nontraditional students into computing careers. Many more studies on women of color in computing regarding balance between family and school or work should be conducted. Future research should highlight elements of success for women of color in CS, rather than dwelling on challenges. For example, at the institutional, departmental, and programmatic levels, effective recruitment and retention practices at MSIs, predominantly White institutions, and community colleges need to be better studied so that others may learn from them. Addressing these knowledge gaps will point us to practical solutions to increase the numbers of women of color in computing and to ensure their success. C

References

1. National Academies. *Rising Above the Gathering Storm, Revisited: Rapidly Approaching Category 5.* National Academies Press, Washington, D.C., 2010.
2. National Science Foundation, National Center for Science and Engineering Statistics. *Women, Minorities, and Persons with Disabilities in Science and Engineering: 2011*, tables 5-7 and 7-7, NSF 11-309. Arlington, VA, 2011; http://www.nsf.gov/statistics/wmpd/.
3. Ong, M., Wright, C., Espinosa, L., and Orfield, G. *Inside the Double Bind: A Synthesis of Empirical Research on Women of Color in Science, Technology, Engineering, and Mathematics.* White Paper presented to the National Science Foundation, Washington, D.C. (NSF/REESE Project DRL-0635577), March 31, 2010; http://www.terc.edu/work/1513.html.
4. Ong, M., Wright, C., Espinosa, E., and Orfield, G. Inside the double bind: A synthesis of empirical research on undergraduate and graduate women of color in science, technology, engineering, and mathematics. *Harvard Educational Review 81*, 2 (Summer 2011), 172–208.
5. Ong, M. and Hodari, A.K. Beyond the double bind: Women of color in STEM. NSF/REESE research project funded by NSF-DRL 0909762, 2009–2012.

Maria (Mia) Ong (mia_ong@terc.edu) is a social scientist at TERC in Cambridge, MA, specializing in the experiences of women of color in STEM in higher education and careers. She is a member of the Committee on Equal Opportunities in Science and Engineering (CEOSE), a congressionally mandated advisory committee to the National Science Foundation, and a member of the Social Science Advisory Board of the National Center for Women in Information Technology (NCWIT).

The author wishes to thank the IDB Project Team, especially Christine Bath, and Richard Ladner and an anonymous reviewer. This work was supported by NSF-DRL grants # 0635577and 0909762, and NSF-REU award # 0635577. Any opinions, findings, conclusions, or recommendations are solely those of the author.

Copyright held by author.

DOI:10.1145/1965724.1965738

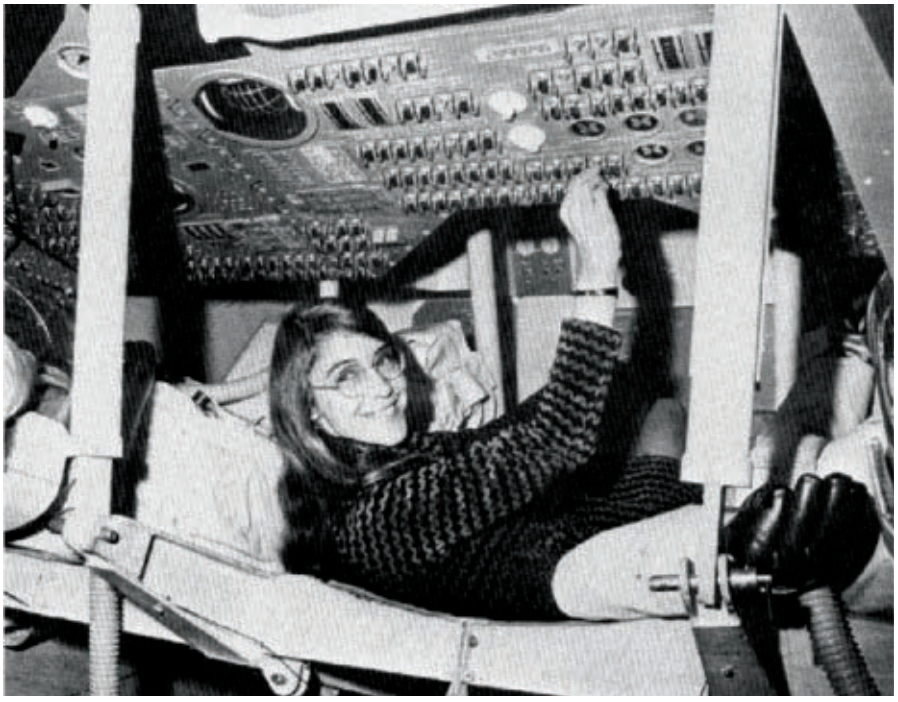
Mordechai (Moti) Ben-Ari

Viewpoint
Non-Myths About Programming

Viewing computer science in a broader context to dispel common misperceptions about studying computer science.

This viewpoint is based on my keynote speech at the Sixth International Computing Education Research Workshop, held in Aarhus, Denmark last summer. The talk began with the presentation of a short play, *Aunt Jennifer*, in which Tiffany, a high school student, attributes her mother's dreary and poverty-stricken life as a checkout clerk in a supermarket to *rotten luck*, while attributing the pleasant life of her Aunt Jennifer, a software engineer, to *good luck*. Despite her high grades in mathematics, Tiffany rejects her guidance counselor's offer to help her obtain a scholarship to study computer science.[a]

The decline of interest in studying computer science is usually attributed to a set of perceptions that students have about the subject. Many educators react to these perceptions as if they were *myths* and try to refute them. I believe the perceptions of students are roughly true when viewed in isolation, and that the proper way to address these *non-myths* is to look at them within the context of "real life." When examined in a broader context, a more valid image of computer science can be sketched, and this can be used to provide more accurate guidance to students who are deliberating whether to study computer science.



PHOTOGRAPH COURTESY OF NASA

Margaret Hamilton, chief software engineer for the development of the NASA Apollo program flight software, sitting in a mockup of the Apollo space capsule while checking programs she and her team developed. Hamilton received an Exceptional Space Act Award, one of only 128 awards granted from 1990 through 2003.

Here, I will express the non-myths in terms of programming.

Non-Myth #1: Programming is Boring

It is one of the unfortunate facts of life that all professions become routine and even boring once you develop a certain level of skill. Of course there are innumerable "McJobs"—intrinsically boring occupations in factories and service industries—that many people must do. But even prestigious professions are not exempt from boredom: I have heard physicians and attorneys complain about boredom. Consider physicians: either you become a general practitioner and at least 9 out of 10 patients come to you with routine, "boring," complaints, or you become a specialist, adept at performing a small number of procedures. After you have done them hundreds or thousand times, surely boredom sets in.

a The script of the play can be downloaded from http://stwww.weizmann.ac.il/g-cs/benari/articles/aunt-jennifer.pdf.

We can partly blame television for the impression that certain occupations are never routine or boring. The patient is always diagnosed and cured within 45 minutes, which is precisely the amount of time it takes to catch and convict a criminal. Occasionally, there are flashes of reality even on TV. "Law and Order" shows how detectives crack a case by following one small, frustrating clue after another. But even here, the 45-minute straightjacket rules. Lt. Van Buren instructs her detectives: "Well, the victim was drunk, so check every bar within 10 blocks." Immediately, the scene cuts to the bartender who provides the next clue, but we don't see the hours of fruitless investigation by the detectives and the junior police officers that led to this moment.

The issue is not whether a subject is boring or not, but your ability to live with particular types of routine that can lead to boredom. Tiffany should be asking herself whether she prefers the routine of working as a psychologist—listening day-in, day-out to people complaining that their parents screwed up their lives—over the routine of constructing dozens of menu entries for the interface of an application.

Non-Myth #2: You Spend Most of Your Working Life in Front of a Computer Screen

For someone to refuse to study computer science for this reason is simply ridiculous. Many people sit in front of computers all day. Computer screens are ubiquitous in all professions in finance, administration, government offices, customer service, and so forth. I am certain my travel agent spends more time looking at her computer screen than I do. From watching movies like *Wall Street* and *Working Girl*, I gather that securities traders spend their lives looking at *six* screens simultaneously.

Our medical system has recently undergone extensive computerization: a patient's history, test results, and diagnostic images are stored on a network of computers. During a visit to a doctor, the patient sits quietly while the doctor reads the history, studies test results, orders X-rays, writes prescriptions, and summarizes the visit, all on a computer. Of course doctors continue to perform physical examinations, but many modern diagnostic and surgical procedures involve "scopes" of various kinds, so that the physician is frequently looking at a computer screen.

The decline of interest in studying computer science is usually attributed to a set of perceptions that students have about the subject.

Tiffany is free to decide that options trading is more exciting than programming, but that choice is not going to save her from the constant use of computers. Certainly, sitting in front of a computer developing software for an insurance company is preferable to sitting in front of a computer entering data from insurance claims.

Non-Myth #3: You Have to Work Long Hours

People who work in high-tech industries complain about long hours, but this is true of many occupations, including prestigious professions, in particular, in the early stages before you achieve a high level of competence and the freedom to work independently. The competition among young attorneys to clock hours is notorious. Young scientists work long hours in an effort to expand their list of publications during the short period before they are reviewed for tenure.

In 1984, Libby Zion, an 18-year-old student, died in a New York hospital from a fatal drug interaction. She was being cared for by young, overworked, interns and residents, who were not aware of a medication she had been taking. New York subsequently enacted a law forbidding residents from working more than 80 hours a week. In comparison, spending 50 hours a week working as a software engineer doesn't seem so bad.

A career as an airline pilot sounds more adventurous than a career as a programmer, but Tiffany should not choose to become a pilot in the expectation of fewer hours at work. Spending long hours in a cubicle in a hi-tech firm, where your hours are flexible and you are free to go out for lunch or to the gym, is not as difficult as being cooped up in the small cockpit of an airplane for many hours at a time, on a schedule over which you have no control.

Non-Myth #4: Programming Is Asocial

Yes, but it depends what you mean by asocial. It is true that a programmer spends long hours by herself in front of a computer screen, although there are also meetings with team members and customers. There certainly are "social" professions where you are in constant contact with other people. The problem is that in most cases the human contact is *superficial* and *asymmetrical*, because you don't "chat" with your "clients." You may not even want to develop a warm relationship with your clients, for example, if you are a police detective interrogating hardened criminals.

A physician is almost always in contact with other people, but much of that is superficial contact with patients. A consultation may take just 15 or 20 minutes, once every few weeks or months. Certainly, the contact is asymmetrical: I tell my doctor every detail of my life that is related to my health, while she tells me nothing about hers.

Nursing is considered to be one of the most caring of professions, but the reality of modern medical care is far from the romantic image. I recall being hospitalized for tests and feeling stressed out, but Chrissie Williams and Donna Jackson (nurses from the BBC medical soap opera "Holby City") did not come over to hold my hand and reassure me. The nurses at the hospital were themselves stressed out with the responsibility for 40 patients, and they barely had time to perform the myriad technical aspects of the job such as administering medication and measuring vital signs.

It is reasonable for Tiffany to choose to become a social worker because she likes helping people direct-

ly, but she must remember that she will not become a friend to her clients.

Non-Myth #5: Programming Is Only for Those Who Think Logically

Well, yes. The nature of programming needs clarification. I define programming as any activity where a computation is described according for formal rules. Painting a picture is not programming: first, it obviously does not describe a computation, and, second, you are free to break whatever rules there are. At worst, they will call you an "Impressionist" and not buy your paintings until after you are dead. Constructing a Web site and building a spreadsheet are both *programming*, because you have to learn the rules for describing the desired output (even if the rules concern a sequence of menu selections and drag-and-drop operations), and you have to debug incorrect results that result from not following the rules.

Tiffany's good grades in mathematics imply she has the ability to think logically. She may prefer to study music so she can play violin in a symphony orchestra, but she should certainly consider studying computer science and her guidance counselor should insist this alternative be thoroughly explored.

Non-Myth #6: Software Is Being Outsourced

Of course it is. However, the share of software being outsourced is relatively small compared with that in manufacturing. This is not a fluke but an intrinsic aspect of software. Almost by definition, "soft"-ware is used whenever flexibility and adaptation to requirements is needed. If a machine tool is going to turn out the same screw throughout its entire lifetime, it can be outsourced and programmed in "hard"-ware.

Software development can also be a path to other professional activities like systems design and marketing, since *software reifies the proprietary knowledge of a firm*. A bank might outsource the building of its Web site, but it is not likely to outsource the development of software to implement algorithms for pricing options or analyzing risk, because this proprietary knowledge is what contributes directly to the bank's success.

It would be reasonable for Tiffany to prefer designing jewelry over studying computer science, but not because software is being outsourced. It is more likely that her jewelry business will fail when confronted with outsourced products than it is that her programming job at Boeing or Airbus will be outsourced.

Non-Myth #7. Programming Is a Well-Paid Profession

That's great. Potential earnings shouldn't be the only consideration when choosing a profession, but it is not immoral to consider what sort of future you will be offering your family. It would be a good idea to remind Tiffany that the chasm between the lifestyles of her mother and Aunt Jennifer is not the result of luck.

I recently read the controversial book *Freakonomics* by Steven D. Levitt and Stephen J. Dubner.[1] The third chapter—"Why Do Drug Dealers Still Live with Their Moms?"—based upon the work of sociologist Sudhir Venkatesh[3] is quite relevant to the issue of potential earnings. As a graduate student, Venkatesh was able to observe and document the lives of the members of a drug gang, and he eventually obtained their financial records. These were analyzed by Levitt, an economist, who came up with the following conclusion, expressed as a question: *So if crack dealing is the most dangerous job in America, and if the salary was only $3.30 an hour, why on earth would anyone take such a job?* The answer: *Well, for the same reason that a pretty Wisconsin farm girl moves to Hollywood. For the same reason that a high-school quarterback wakes up at 5 A.M. to lift weights. They all want to succeed in an extremely competitive field in which, if you reach the top, you are paid a fortune (to say nothing of the attendant glory and power).* The result: *The problem with crack dealing is the same as in every other glamour profession: a lot of people are competing for a very few prizes. Earning big money in the crack gang wasn't much more likely than the Wisconsin farm girl becoming a movie star or the high-school quarterback playing in the NFL.*

Ambition to succeed in a glamour profession is not something to be deplored, but a young person must receive advice and support on what to do if she is not the 1 in 10,000 who succeeds. If Tiffany wants to become a professional singer, I would not try to dissuade her, but I would prefer that she pursue a CS degree part time while she tries to advance her singing career.

The Real World Is Not So Bad

I found the striking image appearing the beginning of this Viewpoint on the NASA Web site. The image shows Margaret Hamilton sitting in a mockup of the Apollo space capsule. Hamilton was the chief software engineer for the development of the Apollo flight software. She and her team developed new techniques of software engineering, which enabled their software to perform flawlessly on all Apollo missions. Later, she went on to establish her own software company.

Hamilton looks like she is having a lot of fun checking out the programs that she and her team developed. I am sure the long hours and whatever routine work the job involved were placed into perspective by the magnitude of the challenge, and there is no question she felt immense satisfaction when her software successfully landed Neil Armstrong and Buzz Aldrin on the moon. I do not know if Hamilton felt locked out of the male-dominated "clubhouse,"[2] but my guess is that the difficulty of the task, the short schedule and the weight of the responsibility felt by the whole team would have made such issues practically nonexistent.

Teachers, parents, and guidance counselors have the responsibility to explain the facts of life to talented young people: computer science and programming may seem like boring activities suitable only for asocial geeks, but a career like Margaret Hamilton's is more fulfilling and more rewarding than what awaits those who do not study science and engineering based upon superficial perceptions of these professions. C

References

1. Levitt, S.D. and Dubner, S.J. *Freakonomics: A Rogue Economist Explores the Hidden Side of Everything.* Allan Lane, London, 2005.
2. Margolis, J. and Fisher, A. *Unlocking the Clubhouse: Women in Computing.* MIT Press, Cambridge, MA, 2002.
3. Venkatesh, S. *Gang Leader for a Day: A Rogue Sociologist Crosses the Line.* Allan Lane, London, 2008.

Mordechai (Moti) Ben-Ari (benari@acm.org) is an associate professor in the Department of Science Teaching at Weizmann Institute of Science in Rehovot, Israel, and an ACM Distinguished Educator.

I would like to thank Mark Guzdial for his helpful comments on an earlier version of this Viewpoint.

Copyright held by author.

DOI:10.1145/1965724.1965739

Article development led by acmqueue
queue.acm.org

How the embeddability of Lua impacted its design.

BY ROBERTO IERUSALIMSCHY, LUIZ HENRIQUE DE FIGUEIREDO, AND WALDEMAR CELES

Passing a Language Through the Eye of a Needle

SCRIPTING LANGUAGES ARE an important element in the current landscape of programming languages. A key feature of a scripting language is its ability to integrate with a system language.[7] This integration takes two main forms: *extending* and *embedding*. In the first form, you extend the scripting language with libraries and functions written in the system language and write your main program in the scripting language. In the second form, you embed the scripting language in a host program (written in the system language) so that the host can run scripts and call functions defined in the scripts; the main program is the host program. In this setting, the system language is usually called the host language.

Many languages (not necessarily scripting languages) support extending through a *foreign function interface* (FFI). An FFI is not enough to allow a function in the system language to do all that a function in the script can do. Nevertheless, in practice FFI covers most common needs for extending, such as access to external libraries and system calls. Embedding, on the other hand, is more difficult to support, because it usually demands closer integration between the host program and the script, and FFI alone does not suffice.

In this article we discuss how embeddability can impact the design of a language, and in particular how it impacted the design of Lua from day one. Lua[3,4] is a scripting language with a particularly strong emphasis on embeddability. It has been embedded in a wide range of applications and is a leading language for scripting games.[2]

The Eye of a Needle

At first sight, the embeddability of a scripting language seems to be a feature of the implementation of its interpreter. Given any interpreter, we can attach an API to it to allow the host program and the script to interact. The design of the language itself, however, has a great influence on the way it can be embedded. Conversely, if you design a language with embeddability in mind, this mind-set will have a great influence on the final language.

The typical host language for most scripting languages is C, and APIs for these languages are therefore mostly composed of functions plus some types and constants. This imposes a natural but narrow restriction on the design of an API for a scripting language: it must offer access to language features through this eye of a needle. Syntactical constructs are particularly difficult to get through. For example, in a scripting language where methods must be written lexically inside their classes, the host language cannot add methods to a class unless the API offers suitable mechanisms.

Similarly, it is difficult to pass lexical scoping through an API, because host functions cannot be lexically inside scripting functions.

A key ingredient in the API for an embeddable language is an `eval` function, which executes a piece of code. In particular, when a scripting language is embedded, all scripts are run by the host calling `eval`. An `eval` function also allows a minimalist approach for designing an API. With an adequate `eval` function, a host can do practically anything in the script environment: it can assign to variables (`eval"a = 20"`), query variables (`eval"return a"`), call functions (`eval"foo(32,'stat')"`), and so on. Data structures such as arrays can be constructed and decomposed by evaluating proper code. For example, again assuming a hypothetical `eval` function, the C code shown in Figure 1 would copy a C array of integers into the script.

Despite its satisfying simplicity and completeness, an API composed of a single `eval` function has two drawbacks: it is too inefficient to be used intensively, because of the cost of parsing and interpreting a chunk at each interaction; and it is too cumbersome to use, because of the string manipulation needed to create commands in C and the need to serialize all data that goes through the API. Nevertheless, this approach is often used in real applications. Python calls it "Very High-Level Embedding."[8]

For a more efficient and easier-to-use API, we need more complexity. Besides an `eval` function for executing scripts, we need direct ways to call functions defined by scripts, to handle errors in scripts, to transfer data between the host program and the scripting environment, and so on. We will discuss these various aspects of an API for an embeddable language and how they have affected and been affected by the design of Lua, but first we discuss how the simple existence of such an API can affect a language.

Given an embeddable language with its API, it is not difficult to write a library in the host language that exports the API back into the scripting language. So, we have an interesting form of reflection, with the host language acting as a mirror. Several mechanisms in Lua use this technique. For example, Lua offers a function called `type` to query the type of a given value. This function is implemented in C outside the interpreter, through an external library. The library simply exports to Lua a C function (called `luaB _ type`) that calls the Lua API to get the type of its argument.

On the one hand, this technique simplifies the implementation of the interpreter; once a mechanism is available to the API, it can easily be made available to the language. On the other hand, it forces language features to pass through the eye of the needle, too. We will see a concrete example of this trade-off when we discuss exception handling.

Control

The first problem related to control that every scripting language must solve is the "who-has-the-main-function" problem. When we use the scripting language embedded in a host, we want the language to be a library, with the main function in the host. For many applications, however, we want the language as a standalone program with its own internal main function.

Lua solves this problem with the use of a separate standalone program. Lua itself is entirely implemented as a library, with the goal of being embedded in other applications. The `lua` command-line program is just a small application that uses the Lua library as any other host to run pieces of Lua code. The code in Figure 2 is a barebones version of this application. The real application, of course, is longer than that, as it has to handle options, errors, signals, and other real-life details, but it still has fewer than 500 lines of C code.

Although function calls form the



ILLUSTRATION BY J.F. PODEVIN

bulk of control communication between Lua and C, there are other forms of control exposed through the API: iterators, error handling, and coroutines. Iterators in Lua allow constructions such as the following one, which iterates over all lines of a file:

```
for line in io.lines(file) do
   print(line)
end
```

Although iterators present a new syntax, they are built on top of first-class functions. In our example, the call `io.lines(file)` returns an *iteration function*, which returns a new line from the file each time it is called. So, the API does not need anything special to handle iterators. It is easy both for Lua code to use iterators written in C (as is the case of `io.lines`) and for C code to iterate using an iterator written in Lua. For this case there is no syntactic support; the C code must do explicitly all that the `for` construct does implicitly in Lua.

Error handling is another area where Lua has suffered a strong influence from the API. All error handling in Lua is based on the `longjump` mechanism of C. It is an example of a feature exported from the API to the language.

The API supports two mechanisms for calling a Lua function: u*nprotected* and *protected*. An unprotected call does not handle errors: any error during the call `long jumps` through this code to land in a protected call farther down the call stack. A protected call sets a recovery point using `setjmp`, so that any error during the call is captured; the call always returns with a proper error code. Such protected calls are very important in an embedded scenario where a host program cannot afford to abort because of occasional errors in a script. The bare-bones application just presented uses `lua_pcall` (protected call) to call each compiled line in protected mode.

The standard Lua library simply exports the protected-call API function to Lua under the name of `pcall`. With `pcall`, the equivalent of a *try-catch* in Lua looks like this:

```
local ok, errorobject = pcall(function()
  --here goes the protected code
  ...
end)
if not ok then
  --here goes the error handling code
  --(errorobject has more information about
  the error)
  ...
end
```

This is certainly more cumbersome than a try-catch primitive mechanism built into the language, but it has a perfect fit with the C API and a very light implementation.

The design of coroutines in Lua is another area where the API had a great impact. Coroutines come in two flavors: *symmetric* and *asymmetric*.[1] Symmetric coroutines offer a single control-transfer primitive, typically called *transfer*, that acts like a *goto*: it can transfer control from any coroutine to any other. Asymmetric coroutines offer two control-transfer primitives, typically called *resume* and *yield*, that act like a pair *call–return*: a `resume` can transfer control to any other coroutine; a `yield` stops the current coroutine and goes back to the one that resumed the one yielding.

It is easy to think of a coroutine as a call stack (a *continuation*) that encodes which computations a program must do to finish that coroutine. The *transfer* primitive of symmetric coroutines corresponds to replacing the entire call stack of the running coroutine by the call stack of the transfer target. On the other hand, the *resume* primitive adds the target stack on top of the current one.

A symmetric coroutine is simpler than an asymmetric one but poses a big problem for an embeddable language such as Lua. Any active C function in a script must have a corresponding activation register in the C stack. At any point during the execution of a script, the call stack may have a mix of C functions and Lua functions. (In particular, the bottom of the call stack always has a C function, which is the host program that initiated the script.) A program cannot remove these C entries from the call stack, however, because C does not offer any mechanism for manipulating its call stack. Therefore, the program cannot make any transfer.

Figure 1. Passing an array through an API with *eval*.

```
void copy (int ar[], int n) {
      int i;
      eval("ar = {}"); /* create an empty array */
      for (i =0; i <n; i++){
        char buff[100];
        sprintf(buff, "ar[%d] = %d", i + 1, ar[i]);
        eval(buff); /* assign i-th element */
      }
}
```

Figure 2. The bare-bones Lua application.

```
#include <stdio.h>
#include "lauxlib.h"
#include "lualib.h"

int main (void) {
  char line[256];
  lua_State *L = luaL_newstate(); /* create a new state */
  luaL_openlibs(L);          /* open the standard libraries */

  /* reads lines and executes them */
  while (fgets(line, sizeof(line), stdin) != NULL) {
    luaL_loadstring(L, line); /* compile line to a function */
    lua_pcall(L, 0, 0, 0);             /* call the function */
  }

  lua_close(L);
  return 0;
}
```

Asymmetric coroutines do not have this problem, because the *resume* primitive does not affect the current stack. There is still a restriction that a program cannot yield *across* a C call—that is, there cannot be a C function in the stack between the `resume` and the *yield*. This restriction is a small price to pay for allowing portable coroutines in Lua.

Data

One of the main problems with the minimalist *eval* approach for an API is the need to serialize all data either as a string or a code segment that rebuilds the data. A practical API should therefore offer other more efficient mechanisms to transfer data between the host program and the scripting environment.

When the host calls a script, data flows from the host program to the scripting environment as arguments, and it flows in the opposite direction as results. When the script calls a host function, we have the reverse. In both cases, data must be able to flow in both directions. Most issues related to data transfer are therefore relevant both for embedding and extending.

To discuss how the Lua–C API handles this flow of data, let's start with an example of how to extend Lua. Figure 3 shows shows the implementation of function `io.getenv`, which accesses environment variables of the host program.

For a script to be able to call this function, we must *register* it into the script environment. We will see how to do this in a moment; for now, let us assume that it has been registered as a global variable `getenv`, which can be used like this:

```
print(getenv("PATH"))
```

The first thing to note in this code is the prototype of `os _ getenv`. The only parameter of that function is a Lua state. The interpreter passes the actual arguments to the function (in this example, the name of the environment variable) through a data structure inside this state. This data structure is a stack of Lua values; given its importance, we refer to it as *the stack*.

When the Lua script calls `getenv`, the Lua interpreter calls `os _ getenv` with the stack containing only the arguments given to `getenv`, with the first argument at position 1 in the stack. The first thing `os _ getenv` does is to call `luaL _ checkstring`, which checks whether the Lua value at position 1 is really a string and returns a pointer to the corresponding C string. (If the value is not a string, `luaL _ checkstring` signals an error using a `longjump`, so that it does not return to `os _ getenv`.)

Next, the function calls `getenv` from the C library, which does the real work. Then it calls `lua _ pushstring`, which converts the C string value into a Lua string and pushes that string onto the stack. Finally, `os _ getenv` returns 1. This return tells the Lua interpreter how many values on the top of the stack should be considered the function results. (Functions in Lua may return multiple results.)

Now let's return to the problem of how to register `os _ getenv` as `getenv` in the scripting environment. One simple way is by changing our previous example of the basic standalone Lua program as follows:

```
lua _ State *L = luaL _ newstate();
/* creates a new state */
luaL _ openlibs(L);
/* opens the standard libraries */

+ lua _ pushcfunction(L, os _ getenv);
+ lua _ setglobal(L, "getenv");
```

The first added line is all the magic we need to extend Lua with host functions. Function `lua _ pushcfunction` receives a pointer to a C function and pushes on the stack a (Lua) function that, when called, calls its corresponding C function. Because functions in Lua are first-class values, the API does not need extra facilities to register global functions, local functions, methods, and so forth. The API needs only the single injection function `lua _ pushcfunction`. Once created as a Lua function, this new value can be manipulated just as any other Lua value. The second added line in the new code calls `lua _ setglobal` to set the value on the top of the stack (the new function) as the value of the global variable `getenv`.

Besides being first-class values, functions in Lua are always anonymous. A declaration such as

```
function inc (x) return x + 1 end
```

is syntactic sugar for an assignment:

```
inc = function (x) return x + 1 end
```

The API code we used to register function `getenv` does exactly the same thing as a declaration in Lua: it creates an anonymous function and assigns it to a global variable.

In the same vein, the API does not need different facilities to call different kinds of Lua functions, such as global functions, local functions, and methods. To call any function, the host first uses the regular data-manipulation facilities of the API to push the function onto the stack, and then pushes the arguments. Once the function (as a first-class value) and the arguments are in the stack, the host can call it with a single API primitive, regardless of where the function came from.

One of the most distinguishing features of Lua is its pervasive use of tables. A table is essentially an associative array. Tables are the only data-structure mechanisms in Lua, so they play a much larger role than in other languages with similar constructions. Lua uses tables not only for all its data structures (records and arrays among others), but also for other language mechanisms, such as modules, objects, and environments.

The example in Figure 4 illustrates the manipulation of tables through the

Figure 3. A simple C function.

```
static int os_getenv (lua_State *L) {
  const char *varname = luaL_checkstring(L, 1);
  const char *value = getenv(varname);
  lua_pushstring(L, value);
  return 1;
}
```

API. Function os _ environ creates and returns a table with all environment variables available to a process. The function assumes access to the environ array, which is predefined in POSIX systems; each entry in this array is a string of the form NAME=VALUE, describing an environment variable.

The first step of os _ environ is to create a new table on the top of the stack by calling lua _ newtable. Then the function traverses the array environ to build a table in Lua reflecting the contents of that array. For each entry in environ, the function pushes the variable name on the stack, pushes the variable value, and then calls lua _ settable to store the pair in the new table. (Unlike lua _ pushstring, which assumes a zero-terminated string, lua _ pushlstring receives an explicit length.)

Function lua _ settable assumes that the key and the value for the new entry are on the top of the stack; the argument −3 in the call tells where the table is in the stack. (Negative numbers index from the top, so −3 means three slots from the top.)

Function lua _ settable pops both the key and the value, but leaves the table where it was in the stack. Therefore, after each iteration, the table is back on the top. The final return1 tells Lua that this table is the only result of os _ environ.

A key property of the Lua API is that it offers no way for C code to refer directly to Lua objects; any value to be manipulated by C code must be on the stack. In our last example, function os _ environ creates a Lua table, fills it with some entries, and returns it to the interpreter. All the time, the table remains on the stack.

We can contrast this approach with using some kind of C type to refer to values of the language. For example, Python has the type PyObject; JNI (Java Native Interface) has jobject. Earlier versions of Lua also offered something similar: a lua _ Object type. After some time, however, we decided to change the API.[6]

The main problem of a lua _ Object type is the interaction with the garbage collector. In Python, the programmer is responsible for calling macros such as Py _ INCREF and DECREF to increment and decrement the reference count of objects being manipulated by the API. This explicit counting is both complex and error prone. In JNI (and in earlier versions of Lua), a reference to an object is valid until the function where it was created returns. This approach is simpler and safer than a manual counting of references, but the programmer loses control of the lifetime of objects. Any object created in a function can be released only when the function returns. In contrast, the stack allows the programmer to control the lifetime of any object in a safe way. While an object is in the stack, it cannot be collected; once out of the stack, it cannot be manipulated. Moreover, the stack offers a natural way to pass parameters and results.

The pervasive use of tables in Lua has a clear impact on the C API. Anything in Lua represented as a table can be manipulated with exactly the same operations. As an example, modules in Lua are implemented as tables. A Lua module is nothing more than a table containing the module functions and occasional data. (Remember, functions are first-class values in Lua.) When you write something like math.sin(x), you think of it as calling the sin function from the math module, but you are actually calling the contents of field "sin" in the table stored in the global variable math. Therefore, it is very easy for the host to create modules, to add functions to existing modules, to "import" modules written in Lua, and the like.

Objects in Lua follow a similar pattern. Lua uses a prototype-based style for object-oriented programming, where objects are represented by tables. Methods are implemented as functions stored in prototypes. Similarly to modules, it is very easy for the host to create objects, to call methods, and so on. In class-based systems, instances of a class and its subclasses must share some structure. Prototype-based systems do not have this requirement, so host objects can inherit behavior from scripting objects and vice versa.

Figure 4. A C function that returns a table.

```
extern char **environ;
static int os_environ (lua_State *L) {
  int i;

  /* push a new table onto the stack */
  lua_newtable(L);

  /* repeat for each environment variable */
  for (i = 0; environ[i] != NULL; i++) {

    /* find the '=' in NAME=VALUE */
    char *eq = strchr(environ[i], '=');

    if (eq) {
      /* push name */
      lua_pushlstring(L, environ[i], eq -environ[i]);

      /* push value */
      lua_pushstring(L, eq + 1);

      /* table[name] = value */
      lua_settable(L, -3);
    }
  }

  /* result is the table */
  return 1;
}
```

eval and Environments

A primary characteristic of a dynamic language is the presence of an *eval* construction, which allows the execution of code built at runtime. As we discussed, an *eval* function is also a basic element in an API for a scripting language. In particular, *eval* is the basic means for a host to run scripts.

Lua does not directly offer an *eval* function. Instead, it offers a *load* function. (The code in Figure 2 uses the luaL _ loadstring function, which

is a variant of *load*.) This function does not execute a piece of code; instead, it produces a Lua function that, when called, executes the given piece of code.

Of course, it is easy to convert *eval* into *load* and vice versa. Despite this equivalence, we think *load* has some advantages over `eval`. Conceptually, *load* maps the program text to a value in the language instead of mapping it to an action. An *eval* function is usually the most complex function in an API. By separating "compilation" from execution, it becomes a little simpler; in particular, unlike *eval, load* never has side effects.

The separation between compilation and execution also avoids a combinatorial problem. Lua has three different load functions, depending on the source: one for loading strings, one for loading files, and one for loading data read by a given reader function. (The former two functions are implemented on top of the latter.)

Because there are two ways to call functions (protected and unprotected), we would need six different `eval` functions to cover all possibilities.

Error handling is also simpler, as static and dynamic errors occur separately. Finally, *load* ensures that all Lua code is always inside some function, which gives more regularity to the language.

Closely related to the `eval` function is the concept of environment. Every Turing-complete language can interpret itself; this is a hallmark of Turing machines. What makes `eval` special is that it executes dynamic code in the same environment as the program that is using it. In other words, an `eval` construction offers some level of reflection. For example, it is not too difficult to write a C interpreter in C. But faced with a statement such as x=1, this interpreter has no way of accessing variable x in the program, if there is one. (Some non-ANSI facilities, such as those related to dynamic-linking libraries, allow a C program to find the address of a given global symbol, but the program still cannot find anything about its type.)

An environment in Lua is simply a table. Lua offers only two kinds of variables: local variables and table fields. Syntactically, Lua also offers global variables: any name not bound to a local declaration is considered global. Semantically, these unbound names refer to fields in a particular table associated with the enclosing function; this table is called the *environment* of that function. In a typical program, most (or all) functions share a single environment table, which then plays the role of a global environment.

Global variables are easily accessible through the API. Because they are table fields, they can be accessed through the regular API to manipulate tables. For example, function `lua_setglobal`, which appears in the bare-bones Lua application code shown earlier, is actually a simple macro written on top of table-manipulation primitives.

Local variables, on the other hand, follow strict lexical-scoping rules, so they do not take part in the API at all. Because C code cannot be lexically nested inside Lua code, C code cannot access local variables in Lua (except through some debug facilities). This is practically the only mechanism in Lua that cannot be emulated through the API.

There are several reasons for this exception. Lexical scoping is an old and powerful concept that should follow the standard behavior. Moreover, because local variables cannot be accessed from outside their scopes, lexical scoping offers programmers a foundation for access control and encapsulation. For example, any file of Lua code can declare local variables that are visible only inside the file. Finally, the static nature of local variables allows the compiler to place all local variables in registers in the register-based virtual machine of Lua.[5]

Conclusion

We have argued that providing an API to the outside world is not a detail in the implementation of a scripting language, but instead is a decision that may affect the entire language. We have shown how the design of Lua was affected by its API and vice versa.

The design of any programming language involves many such trade-offs. Some language attributes, such as simplicity, favor embeddability, while others, such as static verification, do not. The design of Lua involves several trade-offs around embeddability. The support for modules is a typical example. Lua supports modules with a minimum of extra mechanisms, favoring simplicity and embeddability at the expense of some facilities such as unqualified imports. Another example is the support for lexical scoping. Here we chose better static verification to the detriment of its embeddability. We are happy with the balance of trade-offs in Lua, but it was a learning experience for us to pass through the eye of that needle. C

Related articles on queue.acm.org

Purpose-Built Languages
Mike Shapiro
http://queue.acm.org/detail.cfm?id=1508217

A Conversation with Will Harvey
Chris Dibona
http://queue.acm.org/detail.cfm?id=971586

People in Our Software
John Richards, Jim Christensen
http://queue.acm.org/detail.cfm?id=971596

References

1. de Moura, A., Ierusalimschy, R. Revisiting coroutines. *ACM Trans. Programming Languages and Systems 31*, 2 (2009), 6.1–6.31.
2. DeLoura, M. The engine survey: general results. *Gamasutra*; http://www.gamasutra.com/blogs/MarkDeLoura/20090302/581/The_Engine_Survey_General_results.php.
3. Ierusalimschy, R. *Programming in Lua, 2nd Ed.* Lua.org, Rio de Janeiro, Brazil, 2006.
4. Ierusalimschy, R., de Figueiredo, L. H., Celes, W. Lua—An extensible extension language. *Software: Practice and Experience 26*, 6 (1996), 635–652.
5. Ierusalimschy, R., de Figueiredo, L. H., Celes, W. The implementation of Lua 5.0. *Journal of Universal Computer Science 11*, 7 (2005): 1159–1176.
6. Ierusalimschy, R., de Figueiredo, L. H., Celes, W. The evolution of Lua. In *Proceedings of the 3rd ACM SIGPLAN Conference on History of Programming Languages* (San Diego, CA, June 2007).
7. Ousterhout, J.K. Scripting: Higher-level programming for the 21st century. *IEEE Computer 31*, 3 (1998), 23–30.
8. Python Software Foundation. Extending and embedding the Python interpreter, Release 2.7 (Apr. 2011); http://docs.python.org/extending/.

Roberto Ierusalimschy is an associate professor of computer science at PUC-Rio (Pontifical Catholic University of Rio de Janeiro), where he works on programming-language design and implementation. He is the leading architect of the Lua programming language and the author of *Programming in Lua* (now in its second edition).

Luiz Henrique de Figueiredo is a full researcher and a member of the Vision and Graphics Laboratory at the National Institute for Pure and Applied Mathematics in Rio de Janeiro. He is also a consultant for geometric modeling and software tools at Tecgraf, the Computer Graphics Technology Group of PUC-Rio, where he helped create Lua.

Waldemar Celes is an assistant professor in the computer science department at Pontifical Catholic University of Rio de Janeiro (PUC-Rio) and a former postdoctoral associate at the Program of Computer Graphics, Cornell University. He is part of the computer graphics technology group of PUC-Rio, where he coordinates the visualization group. He is also one of the authors of the Lua programming language.

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965740

Article development led by acmqueue
queue.acm.org

Domain-specific languages bridge the semantic gap in programming.

BY DEBASISH GHOSH

DSL for the Uninitiated



ILLUSTRATION BY HANK OSUNA

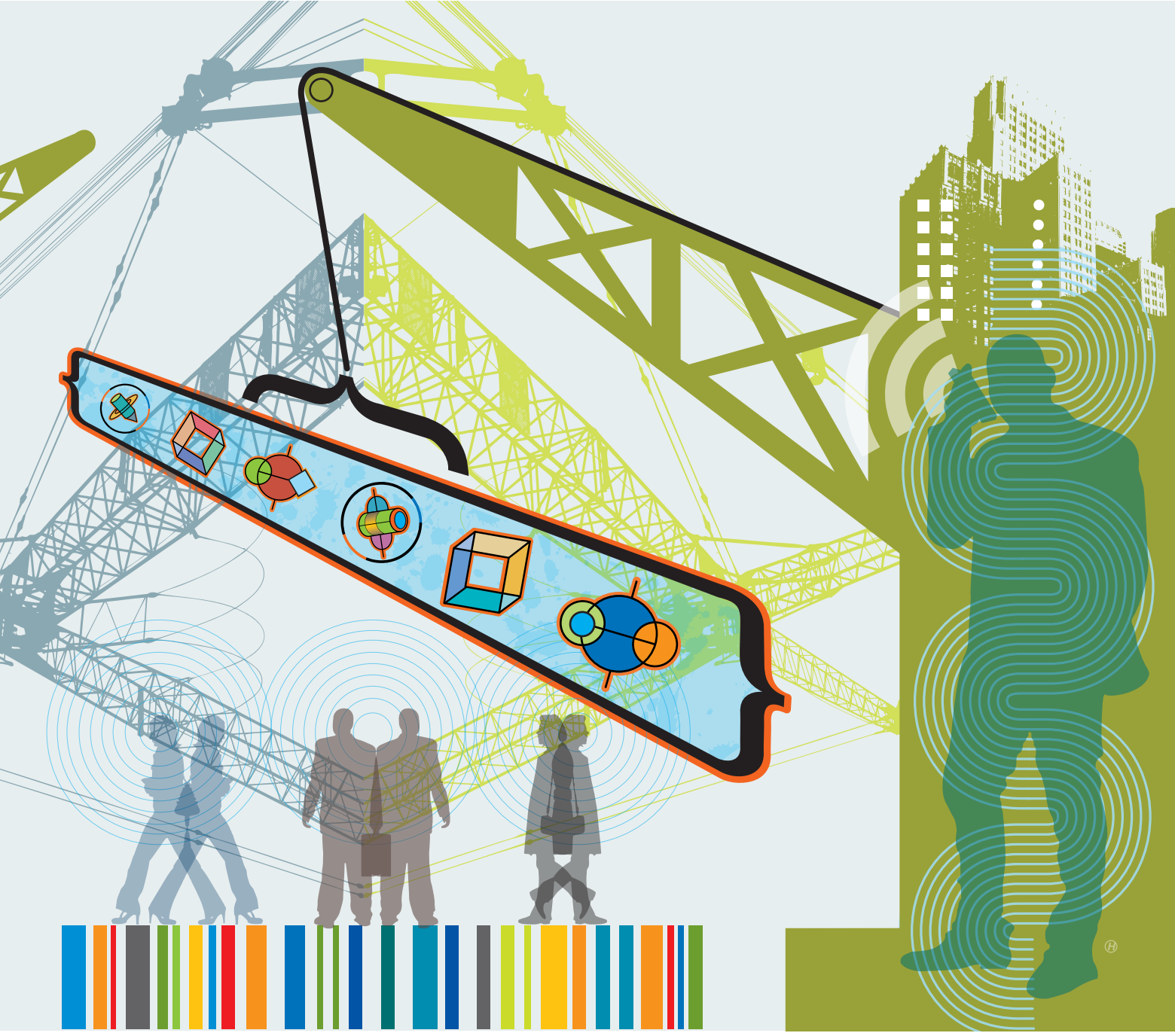
ONE OF THE main reasons why software projects fail is the lack of communication between the business users, who actually know the problem domain, and the developers who design and implement the software model. Business users understand the domain terminology, and they speak a vocabulary that may be quite alien to the software people; it's no wonder that the communication model can break down right at the beginning of the project life cycle.

A domain-specific language (DSL)[1,3] bridges the semantic gap between business users and developers by encouraging better collaboration through shared vocabulary. The domain model the developers build uses the same terminologies as the business. The abstractions the DSL offers match the syntax and semantics of the problem domain. As a result, users can get involved in verifying business rules throughout the life cycle of the project.

This article describes the role a DSL plays in modeling expressive business rules. We start with the basics of domain modeling and then introduce DSLs, which are classified according to implementation techniques. We then explain in detail the design and implementation of an embedded DSL from the domain of securities trading operations.

Domain Modeling

When you model a domain,[7] you identify the various entities and their collaborations. Each entity has a name through which it's identified in that particular domain; the business analyst who is supposed to be an expert in the domain will refer to that entity only by that specific name. When you translate the problem domain artifacts into your solution domain, you construct a software model of the same problem.



As a designer of the new software solution, you expect it to work in the same way as the original problem.

Toward a common vocabulary. It's common knowledge that most projects that fail do so because they lack a proper communication structure between the business users and the implementers. The difference in terminology used by the various stakeholders of the project hinders meaningful collaboration.

A more effective approach is for all parties associated with designing and implementing the system to adopt a common vocabulary early in the life cycle of the project. This can serve as the binding force that unifies the implementation. This means that the business users' daily terminology also appears in the use cases the modeler creates; the programmer uses the same terms while naming abstractions; the data architect does the same in designing data models; and the tester names test cases using the same common vocabulary. In his book on domain-driven design, Eric Evans calls this the *ubiquitous language*.[8]

What's a DSL? In a common vocabulary, it's not only the nouns of the domain that get mapped to the solution space; you need to use the same language of the domain in describing all collaborations within the domain. The mini-language for the domain is modeled within the bounds of your software abstractions, and the software that you develop speaks the language of the domain. Consider the following example from the domain of securities trading operations:

```
newOrder.to.buy(100.shares.of('IBM')){
  limitPrice 300
  allOrNone true
  valueAs {qty, unitPrice -> qty
  * unitPrice - 500}
}
```

This is a loud expression of the language a trader speaks on the floors of the exchange, captured succinctly as an embedded abstraction within your programming language. This is a DSL,[1] a programming language targeted to a specific problem domain

that models the syntax and semantics at the same level of abstraction as the domain itself.[4]

You may be wondering how this particular DSL example developed from the domain model and the common vocabulary business users speak. It involved four major steps:

1. In collaboration with the business users, you derive the common vocabulary of the domain that needs to be used in all aspects the development cycle.

2. You build the domain model using the common vocabulary and the programming language abstractions of the underlying host language.

3. Again in collaboration with the business users, you develop syntactic constructs that glue together the various domain model elements, publishing the syntax for the DSL users. This is a major advantage over a process where you come up with a shared vocabulary up front and then drive the development of the application solely based on that dictionary. In a DSL-based development, you actually develop DSL constructs using the shared vocabulary as the building blocks of your business rules. The actual rules get developed on top of these syntactic constructs.

4. Then you develop the business rules using the syntax of the previous step. In some cases the actual domain users may also participate in the development.

An Introduction to DSL

Designing a DSL is not nearly as daunting a task as designing a general-purpose programming language. A DSL has a very limited focus, and its surface area is restricted to only the current domain being modeled. In fact, most of the common DSLs used today are designed as pure embedded programs within the structure of an existing programming language. Later we show how to accomplish this embedding process to create a mini-language while using the infrastructure of the underlying implementation language.

Martin Fowler classified DSLs based on the way they are implemented.[3] A DSL implemented on top of an underlying programming language is called an *internal* DSL, embedded within the language that implements it (hence, it is also known as an *embedded* DSL). An internal DSL script is, in essence, a program written in the host language and uses the entire infrastructure of the host.

A DSL designed as an independent language without using the infrastructure of an existing host language is called an *external* DSL. It has its own syntax, semantics, and language infrastructure implemented separately by the designer (hence, it is also called a *standalone* DSL).

This article focuses primarily on internal, or embedded, DSLs.

Figure 1. Anatomy of a DSL.

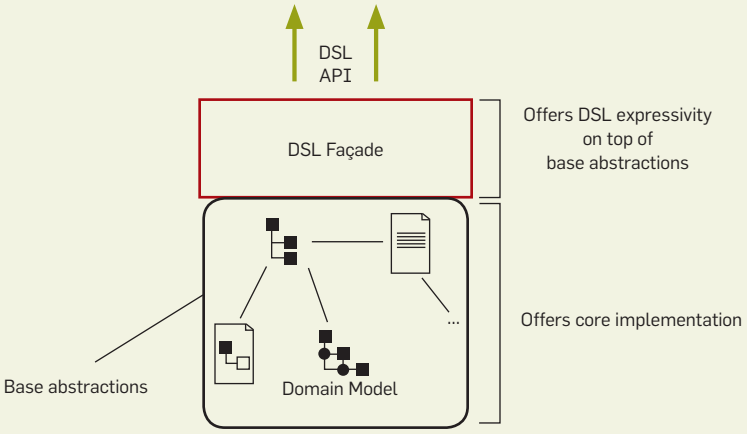
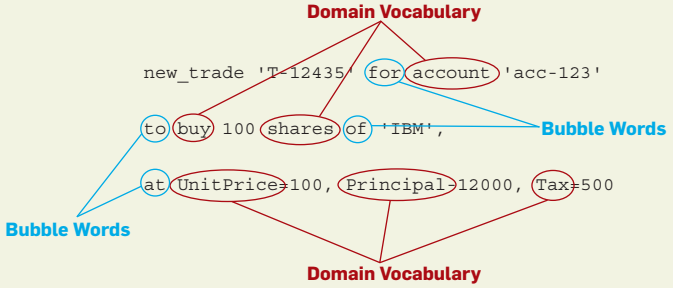


Figure 2. DSL snippet showing domain vocabulary and bubble words.



Advantages of Using a DSL

A DSL is designed to make the business rules of the domain more explicit in the programs. Here are some of the advantages of a DSL:

▸ *Easier collaboration with business users.* Since a DSL shares a common vocabulary with the problem domain, the business users can collaborate with the programmers more effectively throughout the life cycle of the project. They can participate in the development of the actual DSL syntax on top of the domain model and can help in developing some of the business rules using that syntax. Even when the business users cannot program using the syntax, they can validate the implementation of the rules when they are being programmed and can participate in developing some of the test scripts ready to be executed.

▸ *Better expressiveness in domain rules.* A well-designed DSL is developed at a higher level of abstraction. The user of the DSL does not have to care about low-level implementation strategies such as resource allocation or management of complex data structures. This makes the DSL code easier to maintain by programmers who did not develop it.

▸ *Concise surface area of DSL-based*

APIs. A DSL contains the essence of the business rules, so a DSL user can focus on a very small surface area of the code base to model a problem domain artifact.

▸ *DSL-based development can scale.* With a nontrivial domain model, DSL-based development can provide higher payoffs than typical programming models. You need to invest some time up front to design and implement the DSL, but then it can be used productively by a mass of programmers, many of whom may not be experts in the underlying host language.

Disadvantages of Using a DSL

As with any development model, DSL-based development is not without its share of pitfalls. Your project can end up as a complete mess by using badly designed DSLs. Some of these disadvantages are:

▸ *A hard design problem.* Like API design, DSL design is for experts. You need to understand the domain and usage pattern of target users and make the APIs expressive to the right level of abstraction. Not every member of your team can deliver good-quality DSL design.

▸ *Up-front cost.* Unless the project is at least of moderate complexity, designing DSLs may not be cost effective. The up-front cost incurred may offset the time saved from enhanced productivity in the later stages of the development cycle.

▸ *A tendency to use multiple languages.* Unless carefully controlled, this polyglot programming can lead to a language cacophony and result in bloated design.

Structure of a DSL

Here, we look at how to design an internal DSL and embed it within an underlying host language. We address the generic anatomy of an embedded DSL and discuss how to keep the DSL syntax decoupled from the core domain model. Finally, we develop a sample DSL embedded in Scala.

A linguistic abstraction on top of a semantic model. A DSL offers specialized syntactic constructs that model the daily language of a business user. This expressiveness is implemented as a lightweight syntactic construct on top of a rich domain model. Figure 1 provides a diagram of this anatomy.

It's common knowledge that most projects that fail do so because they lack a proper communication structure between the business users and the implementers.

In the figure, the base abstractions refer to the domain model designed using the idioms of the underlying host language. The base abstractions are implemented independent of the DSL that will eventually sit on top of them. This makes it possible to host multiple DSLs on top of a single domain model. Consider the following example of a DSL that models an instruction to do a security trade in a stock exchange:

```
new _ trade 'T-12435' for account 'acc-123'
  to buy 100 shares of 'IBM',
  at UnitPrice=100, Principal=12000,
  Tax=500
```

This is an internal DSL embedded within Ruby as the host language and is very similar to the way a trader speaks at a trading desk. Note that since it's an embedded DSL, it can use the complete infrastructure that Ruby offers such as syntax processing, exception handling, and garbage collection.

The entities are named using a vocabulary a trader understands. Figure 2 annotates the DSL, showing some of the domain vocabulary it uses and some of the "bubble words" we have introduced for the user, giving it more of an English-like feeling.

To implement this DSL, you need an underlying domain model consisting of a set of abstractions in Ruby. This is what we call the *semantic model* (or *domain model*). The previous DSL code snippet interacts with the semantic model through a custom-built interpreter specific to the language we offer to our users. This helps decouple the model from the language designed on top of it. This is one of the best practices to follow when designing a DSL.

Developing an Embedded DSL

An embedded DSL inherits the infrastructure of an existing host language, adapting it in ways that help you abstract the domain you are modeling. As previously mentioned, you build the DSL as an interpreter over the core domain abstractions that you develop using the syntax and semantics of the underlying language.

Choosing the host language. A DSL

offers abstractions at a higher level. Therefore, it is important the language you use to implement your DSL offers similar abstraction capabilities. The more expressive the language is, the less will be the semantic gap between the native abstractions of the language and the custom abstractions you build over it for your DSL. When you choose a language for embedding your DSL, keep an eye on the level of abstractions it offers.

Let's consider an example of designing a small DSL for a specific domain using Scala as the host language. Scala[2,5] is an object functional language designed by Martin Odersky and offers a host of functional and object-oriented features for abstraction design. It has a flexible syntax with type inferencing, an extensible object system, a decent module system, and powerful functional programming capabilities that enable easier development of expressive DSLs. Other features that make Scala a suitable language for embedding DSLs include lexically scoped open classes, implicit parameters, and statically checked duck typing capabilities using structural types.[2]

The problem domain. This example involves a business rule from the domain of securities trading operations, where traders buy and sell securities in a stock exchange (also known as the market) on behalf of their clients, based on some placed order. A client order is executed in the exchange and generates a trade. Depending on whether it is a buy or a sell trade, cash is exchanged between the client and the trader. The amount of cash exchanged is referred to as the net cash value of the trade and varies with the market where the trade is executed.

The business rule used in our example determines the cash-value computation strategy for a specific trade. We built a DSL on top of the core abstractions of the domain model that makes the business rules explicit within the program and can be easily verified by the business users. The core abstractions shown here are simplified for demonstration purposes; the actual production-level abstractions would be much more detailed and complex. The main idea is to show how DSLs can be embedded within a powerful language such as Scala to offer domain-friendly APIs to users.

A DSL offers specialized syntactic constructs that model the daily language of a business user.

The solution domain model. The domain model offers the core abstractions of the business. In our example we use the power of algebraic data types in Scala to model some of the main objects. Trade is the primary abstraction of the domain. Here's how it is modeled using Scala case classes:

```
case class Trade(
  account: Account,
  instrument: Instrument,
  refNo: String,
  market: Market,
  unitPrice: BigDecimal,
  quantity: BigDecimal,
  tradeDate: Date = Calendar.get-
  Instance.getTime,
  valueDate: Option[Date] = None,
  taxFees:Option[List[(TaxFeeId,
  BigDecimal)]] = None,
  netAmount:  Option[BigDecimal]
  = None)
```

In reality, a trade abstraction will have many more details. Similar to `Trade`, we can also use case classes to implement abstractions for `Account` and `Instrument`. We elide them for the time being, as their detailed implementations may not be relevant in this context.

Another abstraction we will use here is Market, also kept simple for the example:

```
sealed trait Market
case object HongKong extends
Market
case object Singapore extends
Market
case object NewYork extends Market
case object Tokyo extends Market
```

These examples use case classes for algebraic data types and case objects to model singletons. Scala case classes offer a few nice features that make the code succinct and concise:

- Constructor parameters as public fields of the class
- Default implementations of `equals`, `toString`, and `hashCode` based on constructor fields
- A companion object containing an `apply()` method and an extractor based on constructor fields

Case classes also offer pattern matching by virtue of their magical autogeneration of the extractors. We used pattern matching on case classes when we designed our DSL. For more details on how case classes make good algebraic data types, refer to Programming in Scala.[2]

The Embedded DSL

Before we dig into the implementation of the DSL that models the net cash-value calculation of a trade, here are some of the business rules that we must consider in the design:

- Net cash-value calculation logic varies with the market where the trade is being executed.
- We can have specific market rules for individual markets such as Hong Kong or Singapore.
- We can have default rules that apply to all other markets.
- If required, the user can also specify custom strategies and domain-specific optimizations for cash-value calculation in the DSL.

In the example, the DSL constructs are designed as linguistic abstractions on top of the domain model. Business users have a major role to play in collaborating with the developers to ensure the right amount of expressiveness is put in the published syntax. It must be loosely coupled from the core abstractions (`Trade`, `Account`, `Instrument`, and so on) and must speak the domain language of the users. The DSL syntax also needs to be composable, so that users can extend the language with custom domain logic on top of what the base language offers.

Once you have the syntactic constructs, you can use them to develop the application business rules. In the following example we develop the business rule for the cash-value calculation logic of trades on top of the syntax the DSL publishes.

Scala offers a rich type system we can use to model some of the business rules. We model the cash-value calculation logic of a trade as a function from `Trade` to `NetAmount`, which is expressed in Scala as `Trade => NetAmount`. Now each such strategy of calculation is driven by a `Market`, which means every such function is defined only for a specific value of the `Market`. We model this as:

```
PartialFunction[Market, Trade
=> NetAmount].
```

Besides expressing the market-based dispatch structure of the calculation logic as an abstract data type, `PartialFunction` in Scala is extensible and can be chained together using combinators such as `andThen` and `orElse`. For more details on how to compose using `PartialFunction`, refer to the Scala Web site.[5]

For convenience let's define a couple of type aliases that abstract the users from the actual underlying data structure that the DSL uses:

```
type NetAmount = BigDecimal
type CashValueCalculationStrategy
= PartialFunction[Market, Trade
=> NetAmount]
```

As the problem domain suggests, we can have a specialized strategy of the cash-value calculation logic for specific markets. As an example, here is how we model a DSL for the `HongKong` market:

```
val forHongKong: CashValueCal-
culationStrategy = {
  case HongKong => { trade =>
  //.. logic for cash value calcu-
  lation for HongKong
  }
}
```

Note how this abstraction is free of unnecessary complexity. It is defined only for the `HongKong` market and returns a function that accepts a trade and returns a calculated cash value. (The actual logic of calculation is elided and may not be relevant to the current context.) Similarly, we can define another specialization for the Singapore market:

```
val forSingapore: CashValueCal-
culationStrategy = {
  case Singapore => { trade =>
  //.. logic for cash value calcu-
  lation for Singapore
  }
}
```

Let's see how the default strategy is selected through a match-any-market parameter:

```
val forDefault: CashValueCalcu-
lationStrategy = {
  case _ => { trade =>
  //.. logic for cash value calcu-
  lation for other markets
  }
}
```

This strategy is selected for *any* market for which it is used. The "_" is a placeholder that matches any market passed to it.

A DSL is useful when the user can compose multiple DSL abstractions to form larger ones. In our case we have designed individual snippets for selecting the appropriate strategy that calculates the net cash value of a trade. How do we compose them so the user can use the DSL without caring about the individual market-specific dispatch logic?

We use an `orElse` combinator that traverses the chain of individual `PartialFunctions` and selects the first matching market. If no market-specific strategy is found, then it selects the default. Here is how we wire these snippets together:

```
lazy val cashValueComputation:
CashValueCalculationStrategy =
  forHongKong orElse
    forSingapore orElse forDe-
    fault
```

This is the DSL that does a dynamic dispatch for the appropriate cash-value calculation strategy together with a fallback for the default. It addresses the first three business rules enumerated at the beginning of the section. The abstraction above is concise, speaks the domain language, and makes the sequencing of the dispatch logic very explicit. A business user who is not a programmer will be able to verify the appropriate domain rule.

One of the benefits of a well-designed DSL is extensibility. The fourth business rule is a use case for that. How can we extend our DSL to allow users to plug in custom cash-value calculation logic they may want to add for another market? Or they may want to override the current logic for an existing market to add some newly introduced market rules. We can compose the user-specified strategy with our existing one using the `orElse` combinator.

```
// pf is the user supplied custom
logic
lazy val cashValue = { pf: Cash-
ValueCalculationStrategy =>
  pf orElse cashValueComputation
}
```

This DSL is very intuitive: it invokes the custom strategy that the user supplied. If it fails to find a match, then it invokes our earlier strategy. Consider the case where the user defines a custom strategy for the Tokyo market and would like to use it instead of the default fallback strategy:

```
val pf: CashValueCalculation-
Strategy = {
  case Tokyo => { trade =>
    //.. custom logic for Tokyo
  }
}
```

Now the user can do the following to supply the preferred strategy to the calculation logic:

```
val trade = //.. trade instance
cashValue(pf)(trade.market)(trade)
```

Our example uses the rich type system of Scala and its powerful functional abstractions to design a DSL that is embedded within the type system of the host language. Note how we express domain-specific rules (such as the need for the calculation logic to vary with specific markets) declaratively, using only the constraints of the static type system. The resulting DSL has the following characteristics:

- It has a small surface area so that it's easier to comprehend, troubleshoot, and maintain.
- It is expressive enough to make the business user understand and verify the correctness.
- It is extensible in that it allows custom plug-in logic (which may include domain-specific optimizations) to be composed into the base combinator in a completely noninvasive way.

Productivity and DSLs

An embedded DSL encourages programming at a higher level of abstraction. The underlying infrastructure of the host language, the details of the type system, the lower-level data structures, and other concerns such as resource management are completely abstracted from the DSL users, so they can focus on building the business functionalities and using the syntax and semantics of the domain.

In our example, the combinator `orElse` of `PartialFunction` hides all details of composing multiple strategies of the cash-value calculation logic. Also, the DSL can be extended for composition with custom logic without any incidental complexity. Thus, the user can focus on implementing the custom abstractions.

We have discussed in detail how to embed a DSL into its host language and make use of the type system to model domain-specific abstractions. You can also design embedded DSLs using dynamically typed languages such as Groovy, Ruby, or Clojure. These languages offer strong metaprogramming facilities that allow users to generate code during compile time or runtime. DSLs developed using these features also lead to enhanced developer productivity, since you get to write only the core business functionalities using the DSL, and the verbose boilerplates are generated by the language infrastructure. Consider the following example of defining a domain object in Rails:

```
class Trade < ActiveRecord::Base
  has _ one :ref _ no
  has _ one :account
  has _ one :instrument
  has _ one :currency
  has _ many :tax _ fees
  ## ..
  validates _ presence _ of :ac-
  count, :instrument, :currency
  validates _ uniqueness _ of
  :ref _ no
  ## ..
end
```

This example defines a `Trade` abstraction and its associations with other entities in a declarative way. The methods `has _ one` and `validates _ presence _ of` express the intent clearly without any verbosity. These are *class methods* in Ruby[6] that use metaprogramming to generate appropriate code snippets during runtime. The DSL that you use for defining `Trade` remains concise, as well as expressive, while all incidental complexities are abstracted away from the surface area of the exposed API.

You can be productive with DSLs with either statically or dynamically typed languages. You just need to use the idioms that make the language powerful. *DSLs in Action*[1] has a detailed treatment of how to use the power of multiple languages idiomatically to design and implement DSLs.

Conclusion

The main value DSLs add to the development life cycle of a project is to encourage better collaboration between the developers and business users. There are multiple ways to implement DSLs. Here, I discussed one that uses embedding within a statically typed programming language. This allows you to use the infrastructure of the host language and focus on developing domain-friendly linguistic abstractions. The abstractions you develop need to be composable and extensible, so the user can build larger abstractions out of smaller ones. Finally, the abstractions need to speak the domain vocabulary, closely matching the semantics the domain user uses. C

Related articles on queue.acm.org

No Source Code? No Problem!
Peter Phillips, George Phillips
http://queue.acm.org/detail.cfm?id=945155

Languages, Levels, Libraries, and Longevity
John R. Mashey
http://queue.acm.org/detail.cfm?id=1039532

Testable System Administration
Mark Burgess
http://queue.acm.org/detail.cfm?id=1937179

References

1. Ghosh, D. *DSLs in Action.* Manning Publications, 2010.
2. Odersky, M., Spoon, L., Venners, B. *Programming in Scala.* Artima, 2010.
3. Fowler, M. *Domain Specific Languages*, Addison Wesley, 2010.
4. Fowler, M. Introducing Domain-Specific Languages. DSL Developer's Conference, 2009; http://msdn.microsoft.com/en-us/data/dd727707.aspx.
5. Scala; http://www.scala-lang.org.
6. Thomas, D., Fowler, C., Hunt, A. *Programming Ruby 1.9.* Pragmatic Press, 2009.
7. Coplien, J. O. *Multiparadigm Design in C++.* Addison-Wesley Professional, Reading, PA, 1988.
8. Evans, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley Professional, Reading, PA, 2003.

Debasish Ghosh (dghosh@acm.org) is the chief technology evangelist at Anshinsoft, where he specializes in leading delivery of enterprise-scale solutions for clients ranging from small to Fortune 500 companies. He is the author of *DSLs In Action* (Manning, 2010) and writes a programming blog at http://debasishg.blogspot.com.

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965741

Article development led by acmqueue
queue.acm.org

A discussion with Nico Kicillof, Wolfgang Grieskamp, and Bob Binder.

ACM CASE STUDY

Microsoft's Protocol Documentation Program: Interoperability Testing at Scale

IN 2002, MICROSOFT began the difficult process of verifying much of the technical documentation for its Windows communication protocols. The undertaking came about as a consequence of a consent decree Microsoft entered into with the U.S. Department of Justice and several state attorneys general that called for the company to make available certain client-server communication protocols for third-party licensees. A series of RFC-like technical documents were then written for the relevant Windows client-server and server-server communication protocols, but to ensure interoperability Microsoft needed to verify the accuracy and completeness of those documents. From the start, it was clear this wouldn't be a typical quality assurance (QA) project. First and foremost, a team would be required to test documentation, not software, which is an inversion of the normal QA process; and the documentation in question was extensive, consisting of more than 250 documents—30,000 pages in all. In addition, the compliance deadlines were tight. To succeed, the Microsoft team would have to find an efficient testing methodology, identify the appropriate technology, and train an army of testers—all within a very short period of time.

This case study considers how the

WOLFGANG GRIESKAMP

One of the challenges for our project was to make sure the functions performed by Windows servers could also be performed by other servers.



team arrived at an approach to that enormous testing challenge. More specifically, it focuses on one of the testing methodologies used—model-based testing—and the primary challenges that have emerged in adopting that approach for a very large-scale project. Two lead engineers from the Microsoft team and an engineer who played a role in reviewing the Microsoft effort tell the story.

Now with Google, **Wolfgang Grieskamp** at the time of this project was part of Microsoft's Windows Server and Cloud Interoperability Group (Winterop), the group charged with testing Microsoft's protocol documentation and, more generally, with ensuring that Microsoft's platforms are interoperable with software from the world beyond Microsoft. Previously, Grieskamp was a researcher at Microsoft Research, where he was involved in efforts to develop model-based testing capabilities.

Nico Kicillof, who worked with Grieskamp at Microsoft Research to develop a model-based testing tool called Spec Explorer, continues to guide testing efforts as part of the Winterop group.

Bob Binder is an expert on matters related to the testing of communication protocols. He too has been involved with the Microsoft testing project, having served as a test methodology consultant who also reviewed work performed by teams of testers in China and India.

For this case study, Binder spoke with Kicillof and Grieskamp regarding some of the key challenges they've faced over the course of their large-scale testing effort.

BOB BINDER: When you first got involved with the Winterop Team [the group responsible for driving the creation, publication, and QA of the Windows communication protocols], what were some of the key challenges?

NICO KICILLOF: The single greatest challenge was that we were faced with testing protocol *documentation* rather than protocol *software*. We had prior expertise in testing software, but this project called for us to define some new processes we could use to test more than 30,000 pages of documentation against existing software implementations already released to the world at large, even in some cases where the original developers were no longer with Microsoft. And that meant the software itself would be the gold standard we would be measuring the documentation against, rather than the other way around. That represented a huge change of perspective.

WOLFGANG GRIESKAMP: What was needed was a new methodology for doing that testing. What's more, it was a new methodology we needed to apply to a very large set of documents in relatively short order. When you put all that together, it added up to a really big challenge. I mean, coming up with something new is one thing. But then to be faced with immediately ap-

ILLUSTRATION BASED ON A PHOTOGRAPH COURTESY OF WOLFGANG GRIESKAMP

plying it to a mission-critical problem and getting a lot of people up to speed just as fast as possible—that was really something.

BINDER: What did these documents contain, and what were they intended to convey?

GRIESKAMP: They're actually similar to the RFCs (request for comments) used to describe Internet protocol standards, and they include descriptions of the data messages sent by the protocol over the wire. They also contain descriptions of the protocol behaviors that should surface whenever data is sent—that is, how some internal data states ought to be updated and the sequence in which that is expected to occur. Toward that end, these documents follow a pretty strict template, which is to say they have a very regular structure.

BINDER: How did your testing approach compare with the techniques typically used to verify specifications?

GRIESKAMP: When it comes to testing one of these documents, you end up testing each normative statement contained in the document. That means making sure each testable normative statement conforms to whatever it is the existing Microsoft implementation for that protocol actually does. So if the document says the server should do *X*, but you find the actual server implementation does *Y*, there's obviously a problem.

In our case, for the most part, that would mean we've got a problem in the document, since the implementation—right or wrong—has already been out in the field for some time. That's completely different from the approach typically taken, where you would test the software against the spec before deploying it.

BINDER: Generally speaking, a protocol refers to a data-formatting standard and some rules regarding how the messages following those formats ought to be sequenced, but I think the protocols we're talking about here go a little beyond that. In that context, can you explain more about the protocols involved here?

GRIESKAMP: We're talking about network communication protocols that apply to traffic sent over network connections. Beyond the data packets themselves, those protocols include many rules governing the interactions between client and server—for example, how the server should respond whenever the client sends the wrong message.

One of the challenges for our project was to make sure the functions performed by Windows servers could also be performed by other servers. Suppose you have a Windows-based server that's sharing files and a Windows-based client accessing them. That's all Microsoft infrastructure, so they *should* be able to talk to each other without any problems. Tests were performed some time ago to make sure of that. But now suppose the server providing the share is running Unix, and a Windows client is running in that same constellation. You still should be able to access the share on the Unix file server in the same way, with the same reliability and quality as if it were a Windows-based file server. In order to accomplish that, however, the Unix-based server would need to follow the same protocol as the Windows-based server. That's where the challenge tends to get a little more interesting.

KICILLOF: That sets the context for saying something about the conditions under which we had to test. In particular, if you're accounting for the fact that the Windows server might eventually be replaced by a Unix server, you have to think in terms of black-box testing. We can't just assume we know how the server is implemented or what its code looks like. Indeed, many of these same tests have been run against non-Microsoft implementations as part of our effort to check for interoperability.

GRIESKAMP: Besides running these tests internally to make sure the Windows server actually behaves the way our documents say it ought to, we also make those same tests available for PlugFests, where licensees who have implemented comparable servers are invited to run the tests against their servers. The goal there is to achieve interoperability, and the most fundamental way to accomplish that is to initiate tests on a client that can basically be run against any arbitrary server in the network, be it a Windows server, a Unix server, or something else.

BINDER: Many of the protocols you've tested use the Microsoft remote procedure call stack—in addition to standard protocols such as SOAP and TCP/IP. What types of challenges have you encountered in the course of dealing with these different underlying stacks?

GRIESKAMP: First off, we put the data more or less directly on the wire so we can just bypass some of those layers. For example, there are some layers in the Windows stack that allow you to send data over TCP without establishing a direct TCP connection, but we chose not to use that. Instead, we talk directly to the TCP socket to send and receive messages.

That allows us to navigate around one part of the stack problem. Another issue is that some protocols travel over other protocols—just as TCP, for example, usually travels over IP, which in turn travels over Ethernet. So what we did to account for that was to assume a certain componentization in our testing approach. That allows us to test the protocol just at the level of abstraction we're concerned with—working on the assumption the underlying transport layers in the stack are behaving just as they ought to be. If we weren't able to make that assumption, our task would be nearly impossible.

Because of the project's unique constraints, the protocol documentation team needed to find a testing methodology that was an ideal fit for their problem. Early efforts focused on collecting data from real interactions between systems and then filtering that information to compare the behaviors of systems under test with those described in the protocol documentation. The problem with this approach was that it was a bit like boiling the ocean. Astronomical amounts of data had to be collected and sifted through to obtain sufficient information to cover thoroughly all the possible protocol states and behaviors described in the documentation—bearing in mind that this arduous process would then have to be repeated for more than 250 protocols altogether.

Eventually the team, in consultation with the U.S. Technical Committee responsible for overseeing their efforts, began to consider model-based testing. In contrast to traditional forms of testing, model-based testing involves generating automated tests from an accurate model of the system under

test. In this case, the system under test would not be an entire software system but rather just the protocols described in the documentation, meaning the team could focus on modeling the protocols' state and behavior and then target the tests that followed on just those levels of the stack of interest for testing purposes.

A team at Microsoft Research had been experimenting with model-based testing since 2002 and had applied it successfully, albeit on a much smaller scale, to a variety of testing situations—including the testing of protocols for Microsoft's Web Services implementation. In the course of those initial efforts, the Microsoft Research team had already managed to tackle some of the thorniest concerns, such as for the handling of nondeterminism. They also had managed to create a testing tool, Spec Explorer, which would prove to be invaluable to the Winterop team.

BINDER: Please say a little about how you came to settle on model-based testing as an appropriate testing methodology.

GRIESKAMP: In looking at the problem from the outset, it was clear it was going to be something huge that required lots of time and resources. Our challenge was to find a smart technology that would help us achieve quality results while also letting us optimize our use of resources. A number of people, including some of the folks on the Technical Committee, suggested model-based testing as a promising technology we should consider. All of that took place before either Nico or I joined the team.

The team then looked around to find some experts in model-based testing, and it turned out we already had a few in Microsoft Research. That led to some discussions about a few test cases in which model-based testing had been employed and the potential the technology might hold for this particular project. One of those test cases had to do with the SMB (Server Message Block) file-sharing protocol. The results were impressive enough to make people think that perhaps we really should move forward with model-based testing. That's when some of us with model-based testing experience ended up being brought over from Microsoft Research to help with the validation effort.

KICILLOF: The specific approach to model-based testing we had taken in Microsoft Research was one that proved to be well suited to this particular problem. Using the tool we had created, Spec Explorer, you could produce models of software that specified a set of rules spelling out how the software was expected to behave and how the state was expected to change as a consequence of each potential interaction between the software and its environment. On the basis of that, test cases could then be generated that included not only pre-scripted test sequences but also the oracle, which is a catalog of all the outcomes that might be expected to follow from each step taken.

In this way it was possible to create tests that would allow you to check along the entire sequence to make sure the system was responding in just the ways you expected it to. And that perfectly matches the way communication protocol documents are written, because they're intended to be interpreted as the rules that govern which messages you should expect to receive, as well as the messages that should then be sent in response.

BINDER: That implies a lot of interesting things. It's easy enough to say, "We have a model and some support for automating exploration of the model." But how did you manage to obtain that model in the first place? What was the process involved in going through the fairly dense prose in each one of those protocol documents and then translating all that into a model?

GRIESKAMP: The first step with model-based testing involved extracting normative statements from all those documents. That had to be done manually since it's not something we're yet able to automate—and we won't be able to automate it until computers are able to read and understand natural human language.

The next step involved converting all those normative statements into a "requirement specification," which is a big table where each of the normative statements has been numbered and all its properties have been described. After that followed another manual step in which a model was created that attempted to exercise and then capture all those requirements. This demanded some higher-level means for measuring so you could make sure you had actually managed to account for all the requirements. For your average protocol, we're talking here about something on the order of many hundreds of different requirements. In some cases, you might even have many thousands of requirements, so this is a pretty large-scale undertaking.

But the general idea is to go from the document to the requirements, and from there to either a model or a traditional test design—whichever one is consistent with your overall approach.

Microsoft encountered challenges because of its choice to adopt model-based testing for the project. On the one hand, the technology and methodology Microsoft Research had developed seemed to fit perfectly with the problem of testing protocol documents. On the other hand, it was an immature technology that presented a steep learning curve. Nonetheless, with the support of the Technical Committee, the team decided to move forward with a plan to quickly develop the technology from Microsoft Research into something suitable for a production-testing environment.

Not surprisingly, this did not prove easy. In addition to the ordinary setbacks that might be expected to crop up with any software engineering project on an extremely tight deadline, the Microsoft protocol documentation team faced the challenge of training hundreds of test developers in China and India on the basics of a new, unfamiliar testing methodology.

Even after they had a cadre of well-trained testers in place, many hurdles still remained. While the tool-engineering team faced the pressure of stabilizing and essentially productizing the Spec Explorer software at breakneck speed, the testing team had to start slogging through hundreds of documents, extracting normative statements, building requirements specifications, and constructing models to generate automated test suites. Although Spec Explorer provides a way to automate tests, there still were several important steps in the process that required human judgment. These ar-



NICO KICILLOF

Increasing the interoperability of our products is a worthy goal in and of itself. We're obviously in a world of heterogeneous technology where customers expect products to interoperate.

eas ended up presenting the team with some of its greatest challenges.

BINDER: How did you manage to convince yourselves you could take several hundred test developers who had virtually no experience in this area and teach them a fairly esoteric technique for translating words into rule systems?

GRIESKAMP: That really was the core risk in terms of taking the model-based testing approach. Until recently, model-based testing technology had been thought of as something that could be applied only by experts, even though it has been applied inside Microsoft for years in many different ways.

Many of the concerns about model-based testing have to do with the learning curve involved, which is admittedly a pretty steep one, but it's not a particularly high one. That is, it's a different paradigm that requires a real mental shift, but it's not really all that complex. So it's not as though it's accessible only to engineers with advanced degrees—everybody can do it. But the first time you're confronted with it, things do look a little unusual.

BINDER: Why is that? What are some of those key differences people have to get accustomed to?

KICILLOF: The basic difference is that a model actually consists of a rule system. So the models we build are made up of rules indicating that under some certain enabling condition, some corresponding update should be performed on state.

From a developer's perspective, however, a program is never just a set of rules. There's a control flow they create and have complete control over. A programmer will know exactly what's to be executed first and what's then supposed to follow according to the inputs received.

What's fortuitous in our case is that we're working from protocol specifications that are themselves sets of rules that let you know, for example, that if you've received message *A*, then you should update your abstract data model and your internal state in a certain way, after which you should issue message *B*. It doesn't explain how a protocol flows from that point on. The combination of all those rules is what determines the actual behavior of the protocol. So there was often a direct correspondence between certain statements in each of these technical documents and the kinds of models we've had to build. That's made it really easy to build the models, as well as to check to make sure they've been built correctly according to the statements found in the documents.

GRIESKAMP: Because this isn't really all that complex, our greatest concern had to do with just getting people used to a new way of thinking. So to get testers past that initial challenge, we counted a lot on getting a good training program in place. That at first involved hiring people to provide the

ILLUSTRATION BASED ON A PHOTOGRAPH COURTESY OF NICO KICILLOF

BOB BINDER

How did you manage to convince yourselves you could take several hundred test developers who had no experience in this area and teach them a fairly esoteric technique for translating words into rule systems?



training for each and every new person our vendors in China and India hired to perform the testing for us. That training covered not only our model-based testing approach, but also some other aspects of the overall methodology.

BINDER: How long did it take for moderately competent developers who had never encountered model-based testing before to get to the point where they could actually be pretty productive?

KICILLOF: On average, I'd say that took close to a month.

BINDER: Once your testers were trained, how did your testing approach evolve? Did you run into any significant problems along the way?

GRIESKAMP: It proved to be a fairly smooth transition since we were just working with concepts that were part of the prototype we had already developed back at Microsoft Research. That said, it actually was just a prototype when this team took it over, so our main challenge was to stabilize the technology. You know how prototypes are—they crash and you end up having to do workarounds and so forth. We've had a development team working to improve the tool over the past three years, and thousands of fixes have come out of that.

Another potential issue had to do with something that often crops up in model-based testing: a state-explosion problem. Whenever you model—if you naively define some rules to update your state whenever certain conditions are met and then you just let the thing run—there's a good chance you're going to end up getting overrun by all those state updates. For example, when using this tool, if you call for an exploration, that should result in a visualization of the exploration graph that you can then inspect. If you're not careful, however, you could end up with thousands and thousands of states the system will try to explore for you. There's just no way you're going to be able to visualize all of that.

Also, in order to see what's actually going on, you need to have some way of pruning down the potential state space such that you can slice out those areas you know you're going to need to test. That's where one of our biggest challenges was: finding the right way to slice the model.

The idea here was to find the right slicing approach for any given problem, and the tool provides a lot of assistance for accomplishing that. It didn't come as a surprise to us that this issue of finding the right way to slice the space would end up being a problem—we had expected that. We actually had already added some things to the tool to deal with that, which is probably one of the reasons the project has proved to be a success.

KICILLOF: The secret is to use test purposes as the criterion for slicing.

BINDER: With that being only a subset of all the behaviors you would be looking at in some particular use case?

GRIESKAMP: Right. So that's why it has

ILLUSTRATION BASED ON A PHOTOGRAPH COURTESY OF BOB BINDER

to be clear that whenever you're doing some slicing, you're cutting away some of the system potential, which means you may lose some test coverage. That's why this ends up being so challenging. As Nico was saying, however, since the slicing is also closely coupled with your test purposes, you still ought to end up being able to cover all the requirements in your documentation.

KICILLOF: Yes, coupling to test purposes is key because if the slicing were done just according to your use cases, only the most common usage patterns of the system might end up being tested. But that's not the case here.

Also, throughout the tool chain, we provide complete traceability between the statements taken from the specification and the steps noted in a test log. We have tools that can tell you whether the way you've decided to slice the model leaves out any requirements you were intending to test. Then at the end you get a report that tells you whether your slicing proved to be excessive or adequate.

By all accounts, the testing project has been extremely successful in helping ensure that Microsoft's protocol documents are of sufficiently high quality to satisfy the company's regulatory obligations related to Windows Client and Windows Server communications. But the effort hasn't stopped there, as much the same approach has been used to test the protocol documentation for Office, SharePoint Server, SQL Server, and Exchange Server.

This work, done with the goal of providing for interoperability with Microsoft's high-volume products, was well suited to the model-based testing technology that was productized to support the court-ordered protocol documentation program. Because projects can be scaled by dividing the work into well-defined units with no cross dependencies, the size of a testing project is limited only by the number of available testers. Because of this scalability, projects can also be completed efficiently, which bodes well for the technology's continued use within Microsoft—and beyond. What's more, Microsoft's protocol documentation testing effort appears to have had a profound effect on the company's overall worldview and engineering culture.

BINDER: Within Microsoft, do you see a broader role for the sort of work you're doing? Or does it pretty much just begin and end with compliance to the court decree?

KICILLOF: It goes beyond the decree. Increasing the interoperability of our products is a worthy goal in and of itself. We're obviously in a world of heterogeneous technology where customers expect products to interoperate.

That's also changing the way products are developed. In fact, one of our goals is to improve the way protocols are created inside Microsoft. That involves the way we design protocols, the way we document protocols such that third parties can use them to talk to our products, and the way we check to make sure our documentation is correct.

GRIESKAMP: One aspect of that has to do with the recognition that a more systematic approach to protocol development is needed. For one thing, we currently spend a lot of money on quality assurance, and the fact that we used to create documentation for products after they had already been shipped has much to do with that. So, right there we had an opportunity to save a lot of money.

Specification or model-driven development is one possible approach for optimizing all of this, and we're already looking into that. The idea is that from each artifact of the development process you can derive documentation, code stubs, and testable specifications that are correct by definition. That way, we won't end up with all these different independently created artifacts that then have to be pieced together after the fact for testing purposes.

For model-based testing in particular, I think this project serves as a powerful proof point of the efficiencies and economies that can be realized using this technology. That's because this is by far the largest undertaking in an industrial setting where, within the same project, both traditional testing methodologies and model-based testing have been used. This has created a rare opportunity to draw some side-by-side comparisons of the two.

We have been carefully measuring various metrics throughout, so we can now show empirically how we managed essentially to double our efficiency by using model-based testing. The ability to actually document that is a really big deal.

BINDER: Yes, that's huge.

GRIESKAMP: There are people in the model-based testing community who have been predicting tenfold gains in efficiency. That might, in fact, be possible if all your users have Ph.Ds or are super adept at model-based testing. But what I think we've been able to show is a significant—albeit less dramatic—improvement with a user population made up of normal people who have no background in model-based testing whatsoever. Also, our numbers include all the ramp-up and education time we had to invest to bring our testers up to speed.

Anyway, after accounting for all that plus the time taken to do a document study and accomplish all kinds of other things, we were able to show a 42% reduction in effort when using the model-based testing approach. I think that ought to prove pretty compelling not just for Microsoft's management but also for a lot of people outside Microsoft. C

Related articles on queue.acm.org

Too Darned Big to Test
Keith Stobie
http://queue.acm.org/detail.cfm?id=1046944
concurrency_s_shysters

Comments are More Important than Code
Jef Raskin
http://queue.acm.org/detail.cfm?id=1053354

Finding Usability Bugs with Automated Tests
Julian Harty
http://queue.acm.org/detail.cfm?id=1925091

Further Reading

1. Grieskamp, W., Kicillof, N., MacDonald, D., Nandan, A., Stobie, K., Wurden, F., Zhang, D. Model-based quality assurance of the SMB2 protocol documentation. In *Proceedings of the 8th International Conference on Quality Software* (2008).
2. Grieskamp, W., Kicillof, N., MacDonald, D., Stobie, K., Wurden, F., Nandan, A. Model-based quality assurance of Windows protocol documentation. In *Proceedings of the 1st International Conference on Software Testing, V & V* (2008).
3. Grieskamp, W., Kicillof, N., Stobie, K., Braberman, V. Model-based quality assurance of protocol documentation: Tools and methodology. *Journal of Software Testing, Verification, Validation and Reliability 21* (Mar. 2011), 55–71.
4. Stobie, K., Kicillof, N., Grieskamp, W. Discretizing technical documentation for end-to-end traceability tests. In *Proceedings of the 2nd International Conference on Advances in System Testing and Validation Lifecycle* (Best paper award, 2010).

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965742

The composer still composes but also gets to take a programming-enabled journey of musical discovery.

BY MICHAEL EDWARDS

Algorithmic Composition: Computational Thinking in Music

IN THE WEST, the layman's vision of the creative artist is largely bound in romantic notions of inspiration sacred or secular in origin. Images are plentiful; for example, a man standing tall on a cliff top, the wind blowing through his long hair, waiting for that particular iconoclastic idea to arrive through the ether.[a] Tales, some even true, of genii penning whole operas in a matter of days, further blur the reality of the usually slowly wrought process of composition. Mozart, with his celebrated speed of writing, is a famous example who to some extent fits the cliché, though perhaps not quite as well as legend would have it.[b]

Non-specialists may be disappointed that composition includes seemingly arbitrary, uninspired formal methods and calculation.[c] What we shall see here is that calculation has been part of the Western composition tradition for at least 1,000 years, This article outlines the history of algorithmic composition from the pre- and post-digital computer age, concentrating, but not exclusively, on how it developed out of the avant-garde Western classical tradition in the second half of the 20th century. This survey is more illustrative than all-inclusive, presenting examples of particular techniques and some of the music that has been produced with them.

A Brief History

Models of musical process are arguably natural to human musical activity. Listening involves both the enjoyment of the sensual sonic experience and the setting up of expectations and possibilities of what is to come: musicologist Erik Christensen described it as follows: "Retention in short-term

a I'm thinking in particular of Caspar David Friedrich's painting *From the Summit* in the Hamburg Kunsthalle.

b Mozart's compositional process is complex and often misunderstood, complicated by myth, especially regarding his now refuted ability to compose everything in his head[15] and his own statements (such as "I must finish now, because I've got to write at breakneck speed—everything's composed—but not written yet" in a letter to his father, Dec. 30, 1780). Mozart apparently distinguished between composing (at the keyboard, in sketches) and writing (preparing a full and final score), hence the confusion about the length of time taken to write certain pieces of music.

c For example, in the realm of pitch: transposition, inversion, retrogradation, intervallic expansion, compression; and in the realm of rhythm: augmentation, diminution, addition.

» key insights

- **Music composition has always been guided by the composer's own computational thinking, sometimes even more than by traditional understanding of inspiration.**
- **Formalization of compositional technique in software can free the mind from musical and cultural clichés and lead to startlingly original results.**
- **Algorithmic composition systems cover all aesthetics and styles, with some open-ended variants offering an alternative to the fixed, never-changing compositions that for most of us define the musical limits.**

ILLUSTRATION BY STUDIO TONNE

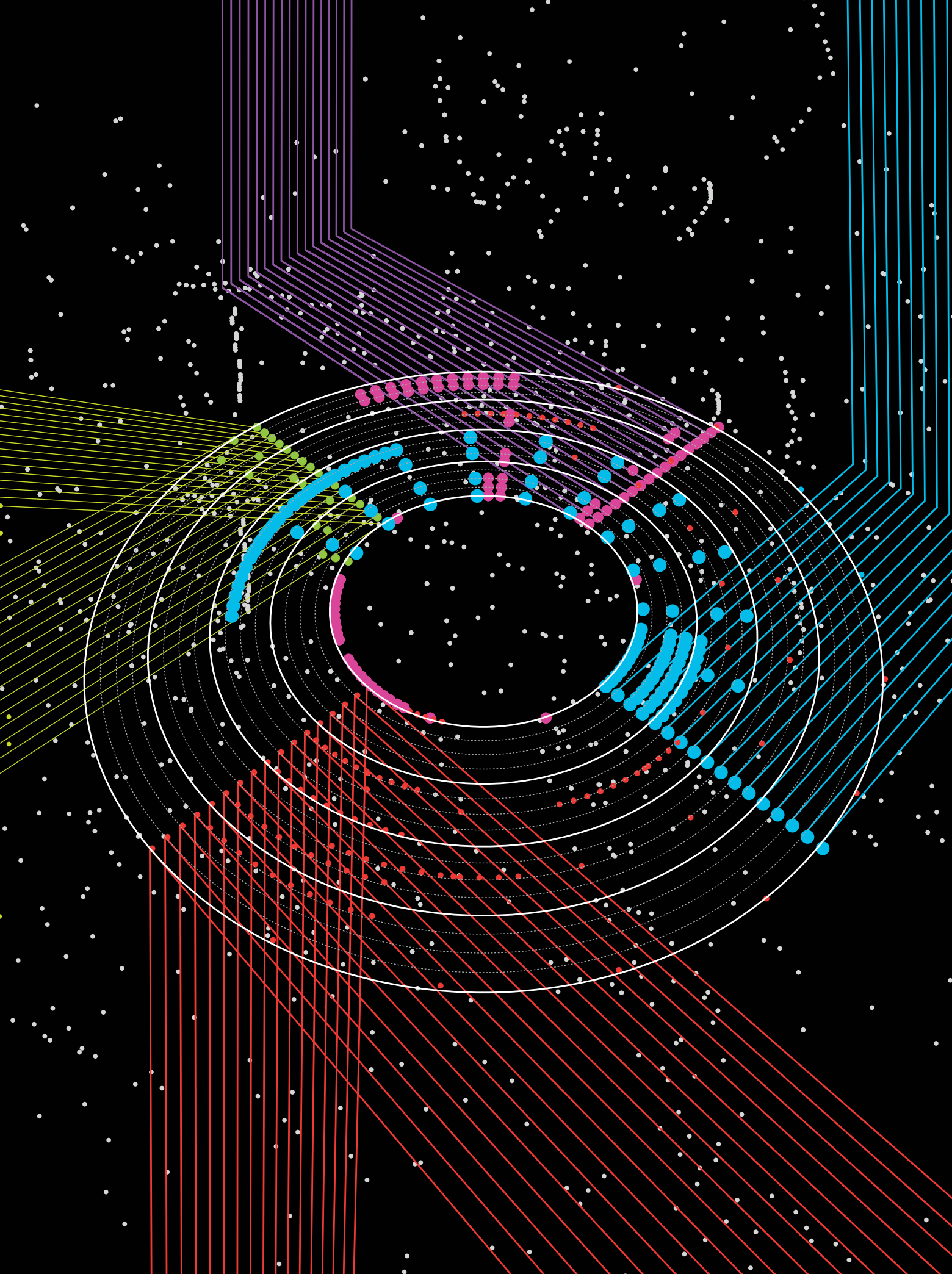


Figure 1. First part of Mozart's *Musikalisches Würfelspiel* ("Musical Dice"): Letters over columns refer to eight parts of a waltz; numbers to the left of rows indicate possible values of two thrown dice; and numbers in the matrix refer to bar numbers of four pages of musical fragments combined to create the algorithmic waltz.

	A	B	C	D	E	F	G	H
2	96	22	141	41	105	122	11	30
3	32	6	128	63	146	46	134	81
4	69	95	158	13	153	55	110	24
5	40	17	113	85	161	2	159	100
6	148	74	163	45	80	97	36	107
7	104	157	27	167	154	68	118	91
8	152	60	171	53	99	133	21	127
9	119	84	114	50	140	86	169	94
2	98	142	42	156	75	129	62	123
11	3	87	165	61	135	47	147	33
12	54	130	10	103	28	37	106	5



Figure 2. Part of an advertisement for *The Geniac Electric Brain*, a DIY music-computer kit.

memory permits the experience of coherent musical entities, comparison with other events in the musical flow, conscious or subconscious comparison with previous musical experience stored in long-term memory, and the continuous formation of expectations of coming musical events."[9]

This second active part of musical listening is what gives rise to the possibility and development of musical form; composer György Ligeti wrote, "Because we spontaneously compare any new feature appearing in consciousness with the features already experienced, and from this comparison draw conclusions about coming features, we pass through the musical edifice as if its construction were present in its totality. The interaction of association, abstraction, memory, and prediction is the prerequisite for the formation of the web of relations that renders the conception of musical form possible."[30]

For centuries, composers have taken advantage of this property of music cognition to formalize compositional structure. We cannot, of course, conflate formal planning with algorithmic techniques, but that the former should lead to the latter was, as I argue here, an historical inevitability.

Around 1026, Guido d'Arezzo (the inventor of staff notation) developed a formal technique to set a text to music. A pitch was assigned to each vowel so the melody varied according to the vowels in the text.[22] The 14th and 15th centuries saw development of the quasi-algorithmic isorhythmic technique, where rhythmic cycles (talea) are repeated, often with melodic cycles (color) of the same or differing lengths, potentially, though not generally in practice, leading to very long forms before the beginning of a rhythmic and melodic repeat coincide. Across ages and cultures, repetition, and therefore memory (of short motifs, longer themes, and whole sections) is central to the development of musical form. In the Western context, this repetition is seen in various guises, including the Classical rondo (with section structures, such as ABACA); the Baroque fugue; and the Classical sonata form, with its return not just of themes but to tonality, too.

Compositions based on number ratios are also found throughout Western musical history; for example, Guillaume Dufay's (1400–1474) isorhythmic motet *Nuper Rosarum Flores*, written for the consecration of Florence Cathedral, March 25, 1436. The temporal structure of the motet is based on the ratios 6:4:2:3, these being the proportions of the nave, the crossing, the apse, and the height of the arch of the cathedral. A subject of much debate is how far the use of proportional systems was conscious on the part of various composers, especially with regards to Fibonacci numbers and the Golden Section.[d] Evidence of Fibonacci relationships haas been found in, for instance, the music of Bach,[32] Schubert,[19] and Bartók,[27] as well as in various other works of the 20th century.[25]

Mozart is thought to have used algorithmic techniques explicitly at least once. His *Musikalisches Würfelspiel* ("Musical Dice")[e] uses musical fragments that are to be combined randomly according to dice throws (see Figure 1). Such formalization procedures are

d Fibonacci was an Italian mathematician (c.1170–c.1250) for whom the famous number series is named. This is a simple progression where successive numbers are the sum of the previous two: (0), 1, 1, 2, 3, 5, 8, 13, 21... Ascending the sequence, the ratio of two adjacent numbers gets closer to the so-called Golden Ratio (approximately 1:1.618).

e Attributed to Mozart though not officially authenticated despite being designated K. Anh. 294d in the *Köchel Catalogue* of his works.

not limited to religious or art music. The *Quadrille Melodist*, sold by Professor J. Clinton of the Royal Conservatory of Music, London (1865) was marketed as a set of cards that allowed a pianist to generate quadrille music (similar to a square dance). The system could apparently make 428 million quadrilles.[34]

Right at the outset of the computer age, algorithmic composition moved straight into the popular, kit-builder's domain. The Geniac Electric Brain allowed customers to build a computer with which they could generate automatic tunes (see Figure 2).[36] Such systems find their modern counterpart in the automatic musical accompaniment software Band-in-a-Box (http://band-in-a-box.com/).

The avant-garde. After World War II, many Western classical music composers continued to develop the serial[f] technique invented by Arnold Schönberg (1874–1951) et al. Though generally seen as a radical break with tradition, in light of the earlier historical examples just presented, serialism's detailed organization can be viewed as no more than a continuation of the tradition of formalizing musical composition. Indeed, one of the new generation's criticisms of Schönberg was that he radicalized only pitch structure, leaving other parameters (such as rhythm, dynamic, even form) in the 19th century.[6] They looked to the music of Schönberg's pupil Anton von Webern for inspiration in organizing these other parameters according to serial principles. Hence the rise of the total serialists: Boulez, Stockhausen, Pousseur, Nono, and others in Europe, and Milton Babbitt and his students at Princeton.[g]

Several composers, notably Xenakis (1922–2001) and Ligeti (1923–2006), offered criticism of and alternatives to serialism, but, significantly, their music was also often governed by complex, even algorithmic, procedures.[h] The complexity of new composition systems made their implementation in computer programs ever more attractive. Furthermore, development of software algorithms in other disciplines made cross-fertilization rife. Thus some techniques are inspired by systems outside the realm of music (such as chaos theory (Ligeti, *Désordre*), neural networks (Gerhard E. Winkler, *Hybrid II "Networks"*),[39] and Brownian motion (Xenakis, *Eonta*).

Much of the resistance to algorithmic composition that persists to this day stems from the misguided bias that the computer, not the composer, composes the music.

Computer-Based Algorithmic Composition

Lejaren Hiller (1924–1994) is widely recognized as the first composer to have applied computer programs to algorithmic composition. The use of specially designed, unique computer hardware was common at U.S. universities in the mid-20th century. Hiller used the Illiac computer at the University of Illinois, Urbana-Champaign, to create experimental new music with algorithms. His collaboration with Leonard Isaacson resulted in 1956 in the first known computer-aided composition, *The Illiac Suite for String Quartet*, programmed in binary, and using, among other techniques, Markov Chains[i] in "random walk" pitch-generation algorithms.[38]

Famous for his own random-process-influenced compositions, if not his work with computers, composer John Cage recognized the potential of Hiller's systems earlier than most. The two collaborated on *HPSCHD*, a piece for "7 harpsichords playing randomly-processed music by Mozart and other composers, 51 tapes of computer-generated sounds, approximately 5,000 slides of abstract

f Serialism is an organizational system in which pitches (first of all) are organized into so-called 12-tone rows, where each pitch in a musical octave is present and, ideally, equally distributed throughout the piece. This technique was developed most famously by Schönberg in the early 1920s at least in part as a response to the difficulty of structuring atonal music, music with no tonal center or key (such as C major).

g Here, we begin to distinguish between pieces that organize pitch only according to the series (dodecaphony) from those extending organization into music's other parameters—strictly speaking serialism, also known as integral or total serialism.

h For a very approachable introduction to the musical thought of Ligeti and Xenakis, see *The Musical Timespace*, chapter 2,[9] particularly pages 36–39.

i First presented in 1906, Markov chains are named for the Russian mathematician Andrey Markov (1856–1922), whose research into random processes led to his eponymous theory, and today are among the most popular algorithmic composition tools. Being stochastic processes, where future states are dependent on current and perhaps past states, they are applicable to, say, pitch selection.

designs and space exploration, and several films."[16] It premiered at the University of Illinois, Urbana-Champaign, in 1969. Summarizing perspicaciously an essential difference between traditional and computer-assisted composition, Cage said in an interview during the composition of *HPSCHD*, "Formerly, when one worked alone, at a given point a decision was made, and one went in one direction rather than another; whereas, in the case of working with another person and with computer facilities, the need to work as though decisions were scarce—as though you had to limit yourself to one idea—is no longer pressing. It's a change from the influences of scarcity or economy to the influences of abundance and—I'd be willing to say—waste."[3]

Stochastic versus deterministic procedures. A basic historical division in the world of algorithmic composition is between indeterminate and determinate models, or those that use stochastic/random procedures (such as Markov chains) and those where results are fixed by the algorithms and remain unchanged no matter how often the algorithms are run. Examples of the latter are cellular automata (though they can be deterministic or stochastic[34]); Lindenmayer Systems (see the section on the deterministic versus stochastic debate in this context); Charles Ames's constrained search algorithms for selecting material properties against a series of constraints[1]; and the compositions of David Cope that use his *Experiments in Musical Intelligence* system.[10] The latter is based on the concept of "recombinacy," where new music is created from existing works, thus allowing the recreation of music in the style of various classical composers, to the shock and delight of many.

Figure 3. Simple L-System rules.

```
1 → 2 3
2 → 1 3
3 → 2 1
```

Figure 4. Step-by-step generation of results from simple L-System rules and a seed.

```
Seed: 2
1 3
2 3 | 2 1
1 3 | 2 1 | 1 3 | 2 3
2 3 | 2 1 | 1 3 | 2 3 | 2 3 | 2 1 | 1 3 | 2 1
```

Algorithmic composition is often viewed as a sideline in contemporary musical activity, as opposed to a logical application and incorporation of compositional technique into the digital domain.

Xenakis. Known primarily for his instrumental compositions but also as an engineer and architect, Iannis Xenakis was a pioneer of algorithmic composition and computer music. Using language typical of the sci-fi age, he wrote, "With the aid of electronic computers, the composer becomes a sort of pilot: he presses buttons, introduces coordinates, and supervises the controls of a cosmic vessel sailing in the space of sound, across sonic constellations and galaxies that he could formerly glimpse only in a distant dream."[40]

Xenakis's approach, which led to the *Stochastic Music Programme* (henceforth SMP) and radically new pieces (such as *Pithoprakta*, 1956), used formulae originally developed by scientists to explain the behavior of gas particles (Maxwell's and Boltzmann's Kinetic Theory of Gases).[31] He saw his stochastic compositions as clouds of sound, with individual notes[j] as the analogue of gas particles. The choice and distribution of notes was determined by procedures involving random choice, probability tables weighing the occurrence of specific events against those of others. Xenakis created several works with SMP, often more than one with the output of a single computer batch process,[k] probably due to limited access to the IBM 7090 he used. His *Eonta* (1963–1964) for two trumpets, three tenor trombones, and piano was composed with SMP. The program was applied in particular to the creation of the massively complex opening piano solo.

Like another algorithmic composition and computer-music pioneer, Gottfried Michael Koenig (1926–), Xenakis had no compunction adapting the output of his algorithms as he saw fit. Regarding *Atrées* (1962), Xenakis's biographer Nouritza Matossian claims Xenakis used "75% computer material,

j Notes are a combination of pitch and duration, rather than just pitch.

k Matossian wrote, "With a single 45-minute program on the IBM 7090, he [Xenakis] succeeded in producing not only eight compositions that stand up as integral works but also in leading the development of computer-aided composition."[31]

composing the remainder himself."[31] At least in Koenig's *Projekt 1* (1964)[l] Koenig saw transcription (from computer output to musical score) as an important part of the process of algorithmic composition, writing, "Neither the histograms nor the connection algorithm contains any hints about the envisaged, 'unfolded' score, which consists of instructions for dividing the labor of the production changes mode, that is, the division into performance parts. The histogram, unfolded to reveal the individual time and parameter values, has to be split up into voices."[24]

Hiller, on the other hand, believed that if the output of the algorithm is deemed insufficient, then the program should be modified and the output regenerated.[34] Several programs that facilitate algorithmic composition include direct connection to their own or to third-party computer sound generation.[m] This connection obviates the need for transcription and even hinders this arguably fruitful intervention. Furthermore, such systems allow the traditional or even conceptual score to be redundant. Thus algorithmic composition techniques allow a fluid and unified relationship between macrostructural musical form and microstructural sound synthesis/processing, as evidenced again by Xenakis in his *Dynamic Stochastic Synthesis* program *Gendy3* (1992).[40]

More current examples. Contemporary (late 20th century) techniques tend to be hybrids of deterministic and stochastic approaches. Systems using techniques from artificial intelligence (AI) and/or linguistics are the generative-grammar[n]-based system Bol Processor software[4] and expert systems (such as Kemal Ebcioglu's CHORAL[11]). Other statistical approaches that use, say, Hidden Markov Models (as in Jordanous and Smaill[20]), tend to need a significant amount of data to train the system; they therefore rely on and generate pastiche copies of the music of a particular composer (that must be codified in machine-readable form) or historical style. While naturally significant to AI research, linguistics, and computer science, such systems tend to be of limited use to composers writing music in a modern and personal style that perhaps resists codification because of its notational and sonic complexity and, more simply, its lack of sufficient and stylistically consistent data—the so-called sparse-data problem. But this is also to some extent indicative of the general difficulty of modeling language and human cognition; the software codification of the workings of a spoken language understood by many and reasonably standardized is one thing; the codification of the quickly developing and widely divergent field of contemporary music is another thing altogether. Thus we can witness a division between composers concerned with creating new music with personalized systems and researchers interested in developing systems for machine learning and AI. The latter may quite understandably find it more useful to generate music in well-known styles not only because there is extant data but also because familiarity of material simplifies some aspects of the assessment of results. Naturally though, more collaboration between composers and researchers could lead to fruitful, aesthetically progressive results.

Figure 5. Larger result set from simple L-System rules.

```
2 3 2 1 1 3 2 3 2 3 2 1 1 3 2 1 1 3 2 1 1 3 2 3 2 3 2
1 1 3 2 3 2 3 2 1 1 3 2 3 2 3 2 1 1 3 2 1 1 3 2 1 1 3
2 3 2 3 2 1 1 3 2 1 1 3 2 1 1 3 2 3 2 3 2 1 1 3 2 1 1
3 2 1 1 3 2 3 2 3 2 1 1 3 2 3 2 3 2 1 1 3 2 3 2 3 2 1
```

Outside academia. Application of algorithmic-composition techniques is not restricted to academia or to the classical avant garde. Pop/ambient musician Brian Eno (1948–) is known for his admiration and use of generative systems in *Music for Airports* (1978) and other pieces. Eno was inspired by the American minimalists, in particular Steve Reich (1936–) and his tape piece *It's Gonna Rain* (1965). This is not computer music but process music, whereby a system is devised—usually repetitive in the case of the minimalists—and allowed to run, generating music in the form of notation or electronic sound. Eno said about his *Discreet Music* (1975), "Since I have always preferred making plans to executing them, I have gravitated towards situations and systems that, once set into operation, could create music with little or no intervention on my part. That is to say, I tend towards the roles of planner and programmer, and then become an audience to the results."[18]

Improvisation systems. Algorithmic composition techniques are, then, clearly not limited to music of a certain aesthetic or stylistic persuasion. Nor are they limited to a completely fixed view of composition, where all the pitches and rhythms are set down in advance. George Lewis's *Voyager* is a work for human improvisors and "computer-driven, interactive 'virtual improvising orchestra.'"[29] Its roots are, according to Lewis, in the African-American tradition of multi-dominance, described by him (borrowing from Jeff Donaldson) as involving multiple simultaneous structural streams, these being in the case of *Voyager* at "both the logical structure of the software and its performance articulation."[29] Lewis programmed *Voyager* in the Forth language popular with computer musicians in the 1980s. Though in *Voyager* the computer is used to analyze and respond to a human improviser, such input is not essential for the program to generate music (via MIDI[o]). Lewis wrote, "I conceive a performance of *Voyager* as multiple parallel streams of music generation, emanating from both the computers and the humans—a nonhierarchical, improvisational, subject-subject model of discourse, rather than a stimulus/response setup."[29] A related improvisation system, OMAX, from the Institut de Recherche et Coordina-

l Written to test the rules of serial music but involving random decisions.[23]

m Especially modern examples (such as Common Music, Pure Data, and SuperCollider).

n Such systems are generally inspired by Chomsky's grammar models[8] and Lerdahl's and Jackendorff's applications of such approaches to generative music theory.[28]

o Musical Instrument Digital Interface, or MIDI, the standard music-industry protocol for interconnecting electronic instruments and related devices.

tion Acoustique/Musique in Paris, is available within the now more widely used computer-music systems Max/MSP and Open-Music. OMAX uses AI-based machine-learning techniques to parse incoming musical data from human musicians, then the results of analysis to generate new material in an improvisatory context.[2]

slippery chicken. In my own case, work on the specialized algorithmic composition program *slippery chicken*[13] is ongoing since 2000. Written in Common Lisp and its object-oriented extension, the Common Lisp Object System, it is mainly deterministic but also has stochastic elements. It has been used to create musical structure for pieces since its inception and is now at the stage where it can generate, in a single pass, complete musical scores for traditional instruments or with the same data write sound files using samples[p] or MIDI file realizations of the instrumental score.[q] The project's main aim is to facilitate a melding of electronic and instrumental sound worlds, not just at the sonic but at the structural level. Hence certain processes common in one medium (such as audio slicing and looping) are transferred to another (such as the slicing up of notated musical phrases and instigation of sub-phrase loops). Also offered are techniques for innovative combination of rhythmic and pitch data, which is, in my opinion, one of the most difficult aspects of making convincing musical algorithms.

Lindenmayer systems. Like writing a paper, composing music, especially with computer-based algorithms, is most often an iterative process. Material is first set down in raw form, only to be edited, developed, and reworked over several passes before the final refined form is achieved. For the composer, stochastic procedures, if not simply to be used to generate material to be reworked by hand or in some other fashion, represent particular problems. If an alteration of the algorithm is deemed necessary, no matter how small, then rerunning the procedure is essential. But rerunning will generate a different set of randomly controlled results, perhaps now lacking some characteristics the composer deemed musically significant after the first pass.[r]

Deterministic procedures can be more apposite. For instance, Lindenmayer Systems[s] (henceforth L-Systems) whose simplicity and elegance yet resulting self-similarity make them ideal for composition. Take a simple example, where a set of rules is defined and associates a key with a result of two further keys that in turn form indices for an arbitrary number of iterations of key substitution (see Figure 3).

Given a starting seed for the lookup and substitution procedure (or rewriting, as it is more generally known), an infinite number of results can be generated (see Figure 4).

Self-similarity is clear when larger result sets are produced; see Figure 5, noting the repetitions of sequences (such as 2 1 1 3 and 2 3 2 3). These numbers can be applied to any musical parameter or material, including pitch, rhythm, dynamic, phrase, and harmony. Seen musically, the results of such simple L-Systems tend toward stasis in that only results that are part of the original rules are returned, and all results are present throughout the returned sequence. However, the result is dependent on the rules defined: subtle manipulations of more complex/numerous rules can result in musically interesting developments. For instance, composers have used more finessed L-Systems—where the result

Figure 6. Fibonacci-based transition from material 0 to material 1. Note the first appearance of 1 is at position 13, with the next eight positions after that, the next again five positions after that, and so on; all these numbers are so-called Fibonacci numbers.

```
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0
0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1
                                1 1 1 1 1 1
```
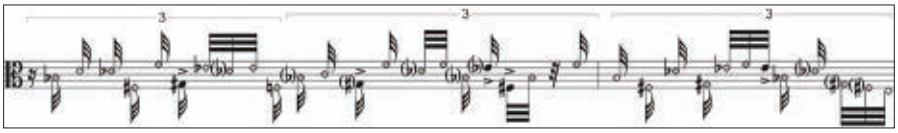


Figure 7. Extract beginning bar 293 of the author's *Tramontana* for viola and computer.

Figure 8. Foreground melodic pattern (scale steps) of *Désordre*.[26]

```
Right hand (white notes), 26 notes, 14 bars
Phrase a:   0  0  1  0  2  1 -1
Phrase a': -1 -1  2  1  3  2 -2
Phrase b:   2  2  4  3  5  4 -1  0  3  2  6  5

Left hand (black notes), 33 notes, 18 bars
Phrase a:   0  0  1  0  2  2  0
Phrase a':  1  1  2  1 -2 -2 -1
Phrase b:   1  1  2  2  0 -1 -4 -3  0 -1  3  2  1 -1  0 -3 -2 -3 -5
```

p Samples are usually short digital sound files of individual or arbitrary number of notes/sonic events.

q To accomplish this, the software interfaces with parts of the open-source software systems Common Music, Common Lisp Music, and Common Music Notation all freely available from http://ccrma.stanford.edu/software.

r This is a simplistic description. Most stochastic procedures involve encapsulation of various tendencies over arbitrarily large data sets, the random details of which are insignificant compared to the structure of the whole. Still, some details may take on more musical importance than intended, and losing them may detrimentally affect the composition. The composer could avoid such problems by using a random number generator with fixed and stored seed, guaranteeing the pseudo-random numbers are generated in the same order each time the process is restarted. Better still would be to modify the algorithm to take these salient, though originally unforeseen features, into account.

s Named for biologist Aristid Lindenmayer (1925–1989) who developed this system (or formal language, based on grammars by Noam Chomsky[33]) that can model various natural-growth processes (such as those of plants).

of a particular rule may be dependent on a sub-rule—leading to more organic, developing forms. Hanspeter Kyburz's (1960–) *Cells* for saxophone and ensemble is an example. Martin Supper[38] described Kyburz's use of L-Systems, using results from 13 generations of L-System rewrites to select precomposed musical motifs. Like Hiller before him, Kyburz uses algorithmic composition techniques to generate and select musical material for the preparation of instrumental scores. However, the listener is probably unaware of the application of software in the composition of such music.

Transitioning L-Systems: Tramontana. As I tend to write music that is concerned with development and transition, my use of L-Systems is somewhat more convoluted. My own *Tramontana* (2004) for viola and computer[14] uses L-Systems in its concluding section. Unlike normal L-Systems, however, I employ Transitioning L-Systems, my own invention, whereby the numbers returned by the L-System are used as lookup indices into a table whose result depends on transitions between related but developing material. The transitions themselves use Fibonacci-based "folding-in" structures where the new material is interspersed gradually until it becomes dominant; for example, a transition from material 0 to material 1 might look like Figure 6.

In the case of the concluding section of *Tramontana*, there is slow development from fast, repeated chords toward more and more flageolets[t] on the C and G strings. Normal pitches and half flageolets[u] begin to dominate, with a tendency toward more of the former. At this point, flageolets on the D string are also introduced. All these developments are created with transitioning L-Systems. The score (see Figure 7 for a short extract) was generated with Bill Schottstaedt's *Common Music Notation* software, taking advantage of its ability to include algorithmically placed nonstandard note heads and other musical signs. Perhaps worth noting is that even before I began work with computers, I was already composing in such a manner. Now, with *slippery chicken* algorithms, these structures can be programmed to generate the music, test, re-work, and re-generate. A particular advantage of working with the computer here is that it is a simple matter to extend or shorten sections, something that would, with pencil and paper, be so time-consuming as to be prohibitive.

> CURTIS ROADS, 1996
>
> **It takes a good composer to design algorithms that result in music that captures the imagination.**

Musical Example: Ligeti's *Désordre*

György Ligeti (1923–2006) is known to the general public mainly through his music in several Stanley Kubrick films: *2001: A Space Odyssey*, which included *Lux Aeterna* and *Requiem* (without Ligeti's permission, prompting a protracted but failed lawsuit); *The Shining*, which included *Lontano*; and *Eyes Wide Shut*, which included *Musica Ricercata*.

After leaving his native Hungary in the late 1950s, Ligeti worked in the same studios as Cologne electronic music pioneers Karlheinz Stockhausen and Gottfried Michael Koenig though produced little electronic music of his own. However, his interest in science and mathematics led to several instrumental pieces influenced by, for example, fractal geometry and chaos theory. But these influences did not lead to a computer-based algorithmic approach.[v] He was quoted in Steinitz[37] saying, "Somewhere underneath, very deeply, there's a common place in our spirit where the beauty of mathematics and the beauty of music meet. But they don't meet on the level of algorithms or making music by calculation. It's much lower, much deeper—or much higher, you could say."

Nevertheless, as a further example, we shall consider the structure of György Ligeti's *Désordre* from his first book of *Piano Etudes* for several reasons:

Structures. The structures of *Désordre* are deceptively simple in concept

t Familiar to guitarists, flageolets, and harmonics are special pitches achieved by touching the string lightly with a left-hand finger at a nodal point in order to bring out higher frequencies related to the fundamental of the open string by integer multiples.

u Half flageolets are achieved by pressing the string, as with a full flageolet, but not at a nodal point; the result is a darker, dead-sounding pitch.

v Ligeti's son, Lukas, confirmed to me that his father was interested conceptually in computers, reading about them over the years, but never worked with them in practice.

yet beautifully elegant in effect, where the clearly deterministic algorithmic thinking lends itself quite naturally to software implementation;

Algorithmic composition. Ligeti was a major composer, admired by experts and non-experts alike, and is generally not associated with algorithmic composition; indeed, *Désordre* was almost certainly composed "algorithmically" by hand, with pencil and paper, as opposed to at a computer keyboard. As such, *Désordre* illustrates the clear link in the history of composition to algorithmic/computational thinking, bringing algorithmic composition into mainstream musical focus; and

Algorithmic models. I have implemented algorithmic models of the first part of *Désordre* in the open-source software system *Pure Data*, which, along with the following discussion, is based on analyses by Tobias Kunze,[26] used here with permission, and Hartmut Kinzler.[21] It is freely downloadable from my Web site http://www.michael-edwards.org/software/desordre.zip[12]; tinkering with the initial data states is instructive and fun.

Désordre's algorithms. The main argument of *Désordre* consists of foreground and background textures:

Foreground (accented, loud). Two simultaneous instances of the same basic process, melodic/rhythmic, one in each hand, both doubled at the octave, and white note (righthand) and black-note[w] (pentatonic, lefthand) modes; and

Background (quiet). Continuous, generally rising quaver (eighth-note) pulse notes, centered between the foreground octaves, one in each hand, in the same mode as the foreground hand.

In the first part of the piece the basic foreground process consists of a melodic pattern cycle consisting of the scale-step shape in Figure 8. This cycle is stated on successively higher (right-hand, 14 times, one diatonic step transposition) and lower (lefthand, 11 times, two diatonic steps transposition) degrees. Thus, a global, long-term movement is created from the middle of the piano outward, to the high and low extremes.

w White and black here refer to the color of the keys on the modern piano.

Figure 9. Foreground rhythmic pattern (quaver/eighth-note durations) of *Désordre*.[26]

right hand:	left hand:
cycle 1: a: 3 5 3 5 5 3 7	3 5 3 5 5 3 8
a': 3 5 3 5 5 3 7	3 5 3 5 5 3 8
b: 3 5 3 5 5 3 3 4 5 3 3 5	3 5 3 5 5 3 3 5 5 3 3 5 3 5 3 5 5 3 8
cycle 2: 3 5 3 4 5 3 8	3 5 3 5 5 3 8
3 5 3 4 5 3 8	3 5 3 5 5 3 8
3 5 3 4 5 3 3 5 5 3 3 4	3 5 3 5 5 3 3 5 5 3 3 5 3 5 3 5 5 3 8
cycle 3: 3 5 3 5 5 3 7	3 5 3 5 5 3 8
3 5 3 5 5 3 7	3 5 3 5 5 2 7
3 5 3 5 5 3 3 4 5 3 3 5	3 4 3 4 4 2 2 4 4 2 2 3 2 3 1 3 3 1 4
cycle 4: 3 5 3 4 5 2 7	1 3 1 2 2 1 3
2 4 2 4 4 2 5	1 2 1 2 2 1 3
2 3 2 3 3 1 1 3 3 1 1 3	1 2 1 2 2 1 1 2 2 1 1 2 1 2 1 2 2 1 3
cycle 5: 1 2 1 2 2 1 3	1 3 1 2 2 1 3
1 2 1 2 2 1 3	1 2 1 2 2 1 3
1 2 1 2 2 1 1 2 2 1 1 2	1 2 1 2 2 1 1 2 2 1 1 2 1 2 1 2 2 1 2
...	...

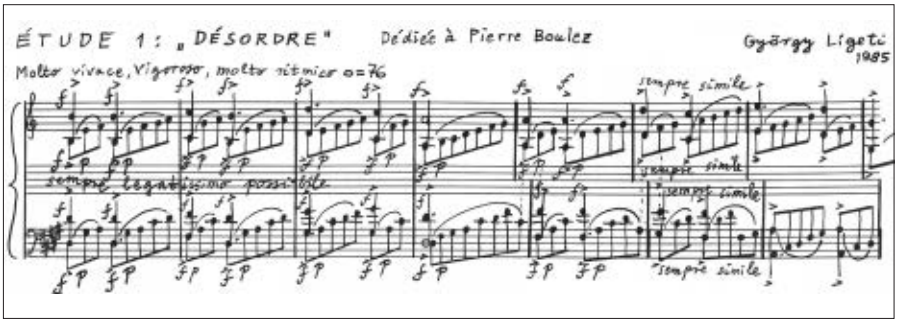


Figure 10. *Désordre*. First system of score © 1986 Schott Music GmbH & Co. KG, Mainz, Germany. Reproduced by permission. All rights reserved.

The foreground rhythmic process consists of slower-moving, irregular combinations of quaver-multiples that tend to reduce in duration over the melodic cycle repeats to create an acceleration toward continuous quaver pulses (see Figure 9).

The similarity between the two hands' foreground rhythmic structure is obvious, but the duration of seven quavers in the right hand at the end of cycle 1a, as opposed to eight in the left, makes for the clearly audible decoupling of the two parts. This is the beginning of the process of disorder, or chaos, and is reflected in the unsynchronized bar lines of the score starting at this point (see Figure 10).

In *Désordre* we experience a clear, compelling, yet not entirely predictable musical development of rhythmic acceleration coupled with a movement from the middle piano register to the extremes of high and low, all expressed through two related and repeating melodic cycles with slightly differing lengths resulting in a combination that dislocates and leads to metrical disorder. I invite the reader to investigate this in more detail by downloading my software implementation.[12]

Conclusion

There has been (and still is) considerable resistance to algorithmic composition from all sides, from musicians to the general public. This resistance bears comparison to the reception of the supposedly overly mathematical serial approach introduced by the composers of the Second Viennese School of the 1920s and 1930s. Alongside the techniques of other music composed from the beginning of the 20th century onward, the serial principle itself is frequently considered to be the reason the music—so-called modern music, though now close to 100 years old—may not appeal. I propose that a more enlightened approach to the arts in general, especially those that present a challenge, would be a more inward-looking examination of the individual response, a deferral of judgment and acknowledgment that, first and foremost, a lack of familiarity with the style and content may lead to a neutral or negative audience

response. Only after further investigation and familiarization can deficiencies in the work be considered.[x]

Algorithmic composition is often viewed as a sideline in contemporary musical activity, as opposed to a logical application and incorporation of compositional technique into the digital domain. Without wishing to imply that instrumental composition is in a general state of stagnation, if the computer is the universal tool, there is surely no doubt that not applying it to composition would be, if not exactly an example of Luddism, then at least to risk missing important aesthetic developments that only the computer can facilitate, and that other artistic fields already take advantage of. That algorithmic thinking is present in Western composition for at least 1,000 years has been established. That such thinking should lend itself to formalization in software algorithms was inevitable.

However, Hiller's work and 1959 *Scientific American* article[17] led to much controversy and press attention. Hostility to his achievements[y] was such that the *Grove Dictionary of Music and Musicians*[z] did not include an article on it until shortly before his death in 1994. This hostility arose no doubt more from a misperception of compositional practice than from anything intrinsic to Hiller's work.

Much of the resistance to algorithmic composition that persists to this day stems from the misguided bias that the computer, not the composer, composes the music. In the vast majority of cases where the composer is also the programmer, this is simply not true. As composer and computer musician Curtis Roads pointed out more than 15 years ago, it takes a good composer to design algorithms that result in music that captures the imagination.[34]

Furthermore, using algorithmic-composition techniques does not by necessity imply less composition work or a shortcut to musical results; rather, it is a change of focus from note-to-note composition to a top-down formalization of compositional process. Composition is, in fact, often slowed by the requirement that musical ideas be expressed and their characteristics encapsulated in a highly structured and non-musical general programming language. Learning the discipline of programming is itself a time-consuming and, for some composers, an insurmountable problem.

Perhaps counterintuitively, such formalization of personal composition technique allows the composer to proceed from concrete musical or abstract formal ideas into realms hitherto unimagined, sometimes impossible to achieve through any other means than computer software. As composer Helmut Lachenmann wrote, "A composer who knows exactly what he wants, wants only what he knows—and that is one way or another too little."[35] The computer can help composers overcome recreating what they already know by aiding more thorough investigations of the material, once procedures are programmed, modifications and manipulations are simpler than with pencil and paper. By "pressing buttons, introducing coordinates, and supervising the controls," to quote Xenakis again,[40] the composer is able to stand back and develop compositional material en masse, applying procedures and assessing, rejecting, accepting, or further processing results of an often-surprising nature. Algorithmic composition techniques clearly further individual musical and compositional development through computer programming-enabled voyages of musical discovery. C

x To paraphrase Ludger Brümmer, from information theory we know that new information is perceived as chaotic or interesting but not expressive. New information must be structured before it can be understood, and, in the case of aesthetic experience, this structuring involves comparison to an ideal, or an established notion of beauty.[7]

y Concerning the reaction to *The Illiac Suite*, Hiller said "There was a great [deal] of hostility, certainly in the musical world...I was immediately pigeonholed as an ex-chemist who had bungled into writing music and probably wouldn't know how to resolve a dominant seventh chord"; interview with Vincent Plush, 1983.[5]

z The *Grove* is the English-speaking world's most widely used and arguably most authoritative musicological resource.

References

1. Ames, C. Stylistic automata in Gradient. *Computer Music Journal 7*, 4 (1983), 45–56.
2. Assayag, G., Bloch, G., Chemillier, M., Cont, A., and Dubnov, S. OMax brothers: A dynamic topology of agents for improvization learning. In *Proceedings of the First ACM Workshop on Audio and Music Computing Multimedia* (Santa Barbara, CA). ACM Press, New York, 2006, 125–132.
3. Austin, L., Cage, J., and Hiller, L. An interview with John Cage and Lejaren Hiller. *Computer Music Journal 16*, 4 (1992), 15–29.
4. Bel, B. Migrating musical concepts: An overview of the Bol processor. *Computer Music Journal 22*, 2 (1998), 56–64.
5. Bewley, J. *Lejaren A. Hiller: Computer Music Pioneer.* Music Library Exhibit, University of Buffalo, 2004; http://library.buffalo.edu/libraries/units/music/exhibits/hillerexhibitsummary.pdf
6. Boulez, P. Schönberg est mort. *Score 6* (Feb. 1952), 18–22.
7. Brümmer, L. Using a digital synthesis language in composition. *Computer Music Journal 18*, 4 (1994), 35–46.
8. Chomsky, N. *Syntactic Structures.* Mouton, The Hague, 1957.
9. Christensen, E. *The Musical Timespace, a Theory of Music Listening.* Aalborg University Press, Aalborg, Denmark, 1996.
10. Cope, D. *Experiments in Musical Intelligence.* A-R Editions, Madison, WI, 1996.
11. Ebcioglu, K. An expert system for harmonizing four-part chorales. *Computer Music Journal 12*, 3 (1988), 43–51.
12. Edwards, M. *A Pure Data implementation of Ligeti's Désordre.* Open-source music software; http://www.michaeledwards.org/software/desordre.zip
13. Edwards, M. *slippery chicken: A Specialized Algorithmic Composition Program.* Unpublished object-oriented Common Lisp software; http://www.michael-edwards.org/slippery-chicken
14. Edwards, M. *Tramontana.* Sheet music, Sumtone, 2004; http://www.sumtone.com/work.php?workid=101
15. Eisen, C. and Keefe, S.P., Eds. *The Cambridge Mozart Encyclopedia.* Cambridge University Press, Cambridge, England, 2006.
16. The Electronic Music Foundation. *HPSCHD*; http://emfnstitute.emf.org/exhibits/hpschd.html
17. Hiller, L. Computer music. *Scientific American 201*, 6 (Dec. 1959), 109–120.
18. Holmes, T. *Electronic and Experimental Music.* Taylor & Francis Ltd, London, 2003.
19. Howat, R. Architecture as drama in late Schubert. In *Schubert Studies*, B. Newbould, Ed. Ashgate Press, London, 1998, 168–192.
20. Jordanous, A. and Smaill, A. Investigating the role of score following in automatic musical accompaniment. *Journal of New Music Research 38*, 2 (2009), 197–209.
21. Kinzler, H. and Ligeti, G. Decision and automatism in *Désordre* 1er étude, premier livre. *Interface, Journal of New Music Research 20*, 2 (1991), 89–124.
22. Kirchmeyer, H. On the historical construction of rationalistic music. *Die Reihe 8* (1962), 11–29.
23. Koenig, G.M. *Project 1*; http://home.planet.nl/gkoenig/indexe.htm
24. Koenig, G.M. Aesthetic integration of computer-composer scores. *Computer Music Journal 7*, 4 (1983), 27–32.
25. Kramer, J. The Fibonacci series in 20th century music. *Journal of Music Theory 17* (1973), 111–148.
26. Kunze, T. *Désordre* (unpublished article); http://www.fictive.com/t/pbl/1999 desordre/ligeti.html
27. Lendvai, E. *Bela Bartók: An Analysis of His Music.* Kahn & Averill, London, 1971.
28. Lerdahl, F. and Jackendorff, R. *A Generative Theory of Tonal Music.* MIT Press, Cambridge, MA, 1983.
29. Lewis, G. Too many notes: Computers, complexity, and culture in Voyager. *Leonardo Music Journal 10* (2000), 33–39.
30. Ligeti, G. Über form in der neuen musik. *Darmstädter Beiträge zur neuen Musik 10* (1966), 23–35.
31. Matossian, N. *Xenakis.* Kahn & Averill, London, 1986.
32. Norden, H. Proportions in music. *Fibonacci Quarterly 2*, 3 (1964), 219–222.
33. Prusinkiewicz, P. and Lindenmayer, A. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, 1990.
34. Roads, C. *The Computer Music Tutorial.* MIT Press, Cambridge, MA, 1996.
35. Ryan, D. and Lachenmann, H. Composer in interview: Helmut Lachenmann. *Tempo 210* (1999), 20–24.
36. Sowa, J. *A Machine to Compose Music: Instruction Manual for GENIAC.* Oliver Garfield Co., New Haven, CT, 1956.
37. Steinitz, R. Music, maths & chaos. *Musical Times 137*, 1837 (Mar. 1996), 14–20.
38. Supper, M. A few remarks on algorithmic composition. *Computer Music Journal 25*, 1 (2001), 48–53.
39. Winkler, G.E. *Hybrid II: Networks.* CD recording, 2003. sumtone cd1: stryngebite; http://www.sumtone.com/recording.php?id=17
40. Xenakis, I. *Formalized Music.* Pendragon, Hillsdale, NY, 1992.

Michael Edwards (michael.edwards@ed.ac.uk) is a Reader in Music Technology in the School of Arts, Culture and Environment of the University of Edinburgh, Edinburgh, U.K.

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965743

SLAM is a program-analysis engine used to check if clients of an API follow the API's stateful usage rules.

BY THOMAS BALL, VLADIMIR LEVIN, AND SRIRAM K. RAJAMANI

A Decade of Software Model Checking with SLAM

LARGE-SCALE SOFTWARE DEVELOPMENT is a notoriously difficult problem. Software is built in layers, and APIs are exposed by each layer to its clients. APIs come with usage rules, and clients must satisfy them while using the APIs. Violations of API rules can cause runtime errors. Thus, it is useful to consider whether API rules can be formally documented so programs using the APIs can be checked at compile time for compliance against the rules.

Some API rules (such as agreement on the number of parameters and data types of each parameter) can be checked by compilers. However, certain rules involve hidden state; for example, consider the rule that the acquire method and release method of a spinlock must be done in strict alternation and the rule that a file can be read only after it is opened. We built the SLAM engine (SLAM from now on) to allow programmers to specify stateful usage rules and statically check if clients follow such rules. We wanted SLAM to be scalable and at the same time have a very low false-error rate. To scale the SLAM engine, we constructed abstractions that retain only information about certain predicates related to the property being checked. To reduce false errors, we refined abstractions automatically using counterexamples from the model checker. Constructing and refining abstractions for scaling model checking has been known for more than 15 years; Kurshan[35] is the earliest reference we know.

SLAM automated the process of abstraction and refinement with counterexamples for programs written in common programming languages (such as C) by introducing new techniques to handle programming-language constructs (such as pointers, procedure calls, and scoping constructs for variables).[2,4–8] Independently and simultaneously with our work, Clarke et al.[17] automated abstraction and refinement with counterexamples in the context of hardware, coining the term "counterexample-driven abstraction refinement," or CEGAR, which we use to refer to this technique throughout

» key insights

- **Even though programs have many states, it is possible to construct an abstraction of a program fine enough to represent parts of a program relevant to an API usage rule and coarse enough for a model checker to explore all the states.**
- **SLAM synthesizes and extends diverse ideas from model checking, theorem proving, and data-flow analysis to automate construction, checking, and refinement of abstractions.**
- **SLAM showed that such abstractions can be constructed automatically for real-world programs, becoming the basis of Microsoft's Static Driver Verifier tool.**

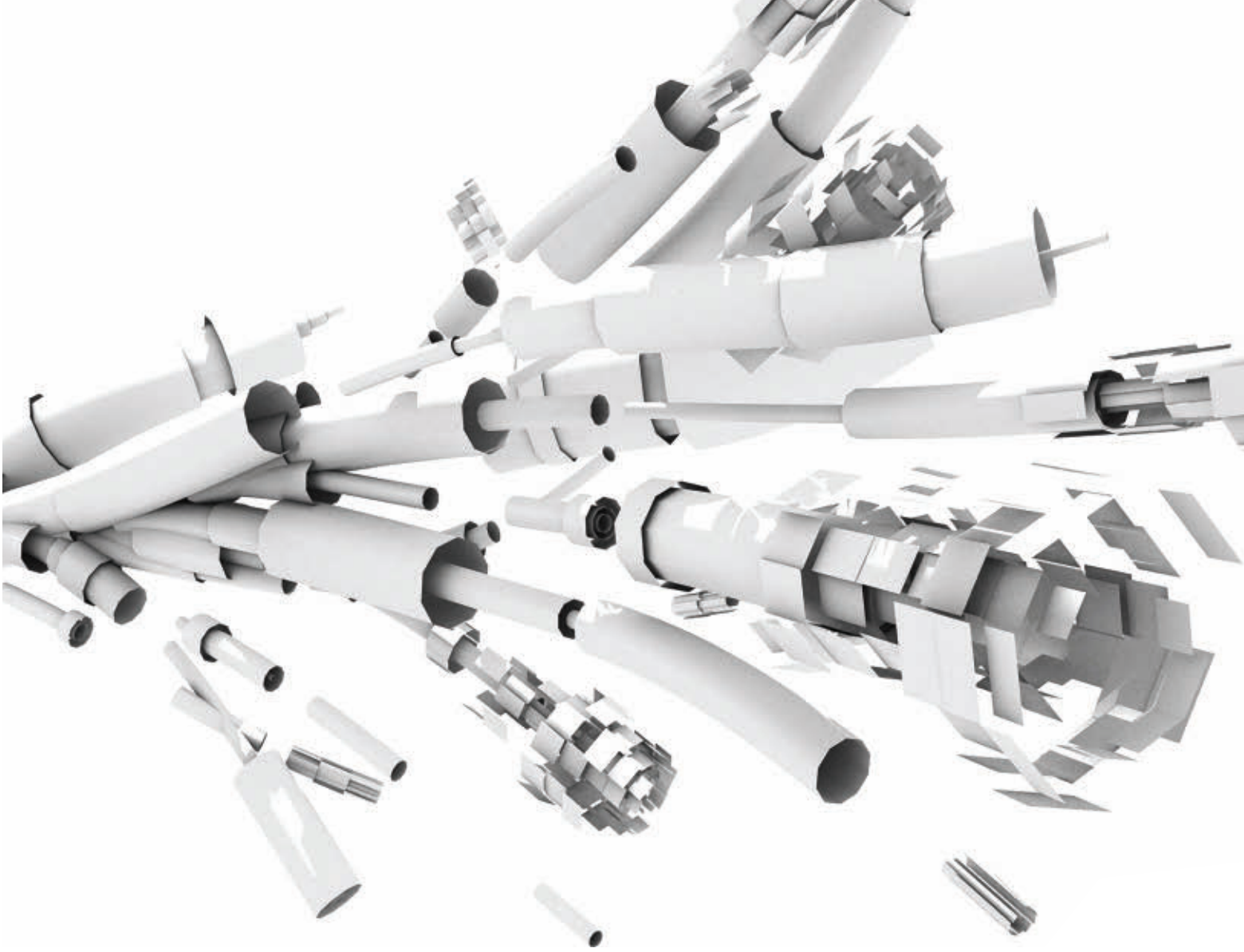


ILLUSTRATION BY RYAN ALEXANDER

this article. The automation of CEGAR for software is technically more intricate, since software, unlike hardware, is infinite state, and programming languages have more expressive and complex features compared to hardware-description languages. Programming languages allow procedures with unbounded call stacks (handled by SLAM using pushdown model-checking techniques), scoping of variables (exploited by SLAM for efficiency), and pointers allowing the same memory to be aliased by different variables (handled by SLAM using pointer-alias-analysis techniques).

We also identified a "killer-app" for SLAM—checking if Windows device drivers satisfy driver API usage rules. We wrapped SLAM with a set of rules specific to the Windows driver API and a tool chain to enable push-button validation of Windows drivers, resulting in a tool called "static driver verifier," or SDV. Such tools are strategically important for the Windows device ecosystem, which encourages and relies on hardware vendors making devices and writing Windows device drivers while requiring vendors to provide evidence that the devices and drivers perform acceptably. Because many drivers use the same Windows-driver API, the cost of manually specifying the API rules and writing them down is amortized over the value obtained by checking the same rules over many device drivers.

Here, we offer a 10-year retrospective of SLAM and SDV, including a self-contained overview of SLAM, our experience taking SLAM to a full-fledged SDV product, a description of how we built and deployed SDV, and results obtained from the use of SDV.

SLAM

Initially, we coined the label SLAM as an acronym for "software (specifications), programming languages, abstraction, and model checking." Over time, we used SLAM more as a forceful verb; to "SLAM" a program is to exhaustively explore its paths and eliminate its errors. We also designed the "Specification Language for Interface Checking," or SLIC,[9] to specify stateful API rules and created the SLAM tool as a flexible verifier to check if code that uses the API follows the SLIC rules. We wanted to build a verifier covering all possible behaviors of the program while checking the rule, as opposed to a testing tool that checks the rule on a subset of behaviors covered by the test.

In order for the solution to scale while covering all possible behaviors, we introduced Boolean programs. Boolean programs are like C programs in the sense that they have all the control constructs of C programs—sequencing, conditionals, loops, and procedure calls—but allow only Boolean variables (with local, as well as global,

scope). Boolean programs made sense as an abstraction for device drivers because we found that most of the API rules drivers must follow tend to be control-dominated, and so can be checked by modeling control flow in the program accurately and modeling only a few predicates about data relevant to each rule being checked.

The predicates that need to be "pulled into" the model are dependent on how the client code manages state relevant to the rule. CEGAR is used to discover the relevant state automatically so as to balance the dual objectives of scaling to large programs and reducing false errors.

SLIC specification language. We designed SLAM to check temporal safety properties of programs using a well-defined interface or API. Safety properties are properties whose violation is witnessed by a finite execution path. A simple example of a safety property is that a lock should be alternatively acquired and released. SLIC allows us to encode temporal safety properties in a C-like language that defines a safety automaton[44] that monitors a program's execution behavior at the level of function calls and returns. The automaton can read (but not modify) the state of the C program that is visible at the function call/return interface, maintain a history, and signal the occurrence of a bad state.

A SLIC rule includes three components: a static set of state variables, described as a C structure; a set of events and event handlers that specify state transitions on the events; and a set of annotations that bind the rule to various object instances in the program (not shown in this example). As an example of a rule, consider the locking rule in Figure 1a. Line 1 declares a C structure containing one field `state`, an enumeration that can be either `Unlocked` or `Locked`, to capture the state of the lock. Lines 3–5 describe an event handler for calls to `KeInitializeSpinLock`. Lines 7–13 describe an event handler for calls to the function `KeAcquireSpinLock`. The code for the handler expects the `state` to be in `Unlocked` and moves it to `Locked` (specified in line 9). If the `state` is already `Locked`, then the program has called `KeAcquireSpinLock` twice without an intervening call to `KeReleaseSpinLock` and is an error (line 9). Lines 15–21 similarly describe an event handler for calls to the function `KeReleaseSpinLock`[a]. Figure 1b is a piece of code that uses the functions `KeAcquireSpinLock` and `KeReleaseSpinLock`. Figure 1c is the same code after it has been instrumented with calls to the appropriate event handlers. We return to this example later.

a A more detailed example of this rule would handle different instances of locks, but we cover the simple version here for ease of exposition.

CEGAR via predicate abstraction. Figure 2 presents ML-style pseudocode of the CEGAR process. The goal of SLAM is to check if all executions of the given C program *P* (type *cprog*) satisfy a SLIC rule *S* (type spec).

The instrument function takes the program *P* and SLIC rule *S* as inputs and produces an instrumented program *P´* as output, based on the product-construction technique for safety properties described in Vardi and Wolper.[44] It hooks up relevant events via calls to event handlers specified in the rule *S*, maps the error statements in the SLIC rule to a unique error state in *P´*, and guarantees that *P* satisfies *S* if and only if the instrumented program *P´* never reaches the error state. Thus, this function reduces the problem of checking if *P* satisfies *S* to checking if *P´* can reach the error state.

The function *slam* takes a C program *P* and SLIC rule specification *S* as input and passes the instrumented C program to the tail-recursive function *cegar*, along with the predicates extracted from the specification *S* (specifically, the guards that appear in *S* as predicates).

The first step of the *cegar* function is to abstract program *P´* with respect to

Figure 1. (a) Simplified SLIC locking rule; (b) code fragment using spinlocks; (c) fragment after instrumentation.

```
1  state { enum {Unlocked, Locked} state; }
2
3  KeInitializeSpinLock.call {
4      state = Unlocked;
5  }
6
7  KeAcquireSpinLock.call {
8      if ( state == Locked ) {
9          error;
10     } else {
11         state = Locked;
12     }
13 }
14
15 KeReleaseSpinLock.call {
16     if ( !(state == Locked) ) {
17         error;
18     } else {
19         state = Unlocked;
20     }
21 }
22
```

(a)

```
1  ..
2  KeInitializeSpinLock();
3  ..
4  ..
5  if(x > 0)
6    KeAcquireSpinlock();
7  count = count+1;
8  devicebuffer[count] = localbuffer[count];
9  if(x > 0)
10   KeReleaseSpinLock();
11 ...
12 ...
```

(b)

```
1  ..
2  { state = Unlocked;
3    KeInitializeSpinLock();}
4  ..
5  ..
6  if(x > 0)
7    { SLIC_KeAcquireSpinLock_call();
8      KeAcquireSpinlock(); }
9  count = count+1;
10 devicebuffer[count] = localbuffer[count];
11 if(x > 0)
12   { SLIC_KeReleaseSpinLock_call();
13     KeReleaseSpinLock(); }
14 ...
15 ...
```

(c)

the predicate set preds to create a Boolean program abstraction B. The automated transformation of a C program into a Boolean program uses a technique called predicate abstraction, first introduced in Graf and Saïdi[29] and later extended to work with programming-language features in Ball et al.[2] and Ball et al.[3]

The program B has exactly the same control-flow skeleton as program P'. By construction, for any set of predicates preds, every execution trace of the C program P' also is an execution trace of $B = abstract(P', preds)$; that is, the execution traces of P' are a subset of those of B. The Boolean program B models only the portions of the state of P' relevant to the current SLIC rule, using nondeterminism to abstract away irrelevant state in P'.

Once the Boolean program B is constructed, the *check* function exhaustively explores the state space of B to determine if the (unique) error state is reachable. Even though all variables in B are Boolean, it can have procedure calls and a potentially unbounded call stack. Our model checker performs symbolic reachability analysis of the Boolean program (a pushdown system) using binary decision diagrams.[11] It uses ideas from interprocedural data flow analysis[42,43] and builds summaries for each procedure to handle recursion and variable scoping.

If the *check* function returns **AbstractPass**, then the error state is not reachable in B and therefore is also not reachable in P'. In this case, SLAM has proved that the C program P satisfies the specification S. However, if the *check* function returns **AbstractFail** with witness trace *trc*, the error state is reachable in the Boolean program B but not necessarily in the C program P'. Therefore, the trace *trc* must be validated in the context of P' to prove it really is an execution trace of P'.

The function *symexec* symbolically executes the trace *trc* in the context of the C program P'. Specifically, it constructs a formula $\phi(P', trc)$ that is satisfiable if and only if there exists an input that would cause program P' to execute trace *trc*. If *symexec* returns **Satisfiable**, then SLAM has proved program P does not satisfy specification S and returns the counterexample trace *trc*.

If the function *symexec* returns **Unsatisfiable**(prf), then it has found a proof prf that there is no input that would cause P' to execute trace *trc*. The function *refine* takes this proof of unsatisfiability, reduces it to a smaller proof of unsatisfiability, and returns the set of constituent predicates from this smaller proof. The function *refine* guarantees that the trace *trc* is not an execution trace of the Boolean program

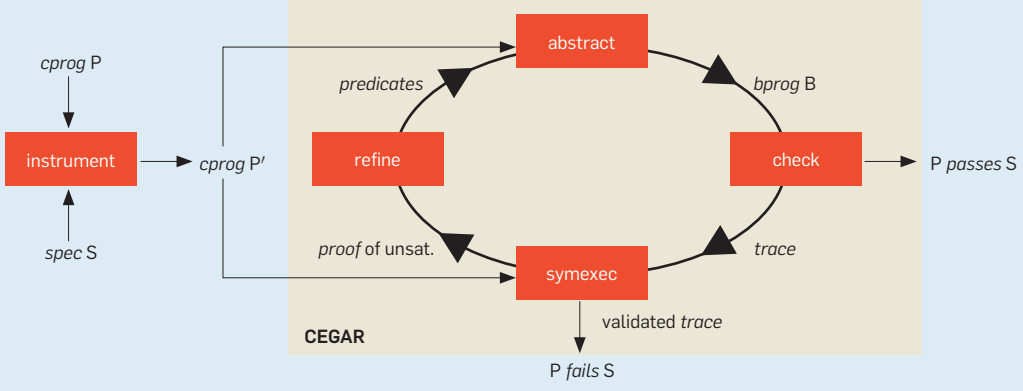
$$abstract(P', preds \cup refine(prf))$$

The ability to refine the (Boolean program) abstraction to rule out a spurious counterexample is known as the progress property of the CEGAR process.

Despite the progress property, the CEGAR process offers no guarantee of terminating since the program P' may have an intractably large or infinite number of states; it can refine the Boolean program forever without discovering a proof of correctness or proof of error.

However, as each Boolean program is guaranteed to overapproximate the behavior of the C program, stopping the CEGAR process before it terminates with a definitive result is no different from any terminating program analysis that produces false alarms. In practice, SLAM terminates with a definite result over 96% of the time on large classes of device drivers: for Windows Driver Framework (WDF) drivers, the figure is

Figure 2. Graphical illustration and ML-style pseudocode of CEGAR loop.




```
type cprog, spec, predicates, bprog, trace, proof

type result =
   Pass | Fail of trace

type chkresult =
   AbstractPass | AbstractFail of trace

type excresult =
   Satisable | Unsatisable of proof
```

```
let rec cegar (P':cprog) (preds :predicates) : result =
   let B: bprog = abstract (P',preds) in
   match check(B) with
   | AbstractPass -> Pass
   | AbstractFail(trc) ->
      match symexec(P', trc) with
      | Satisable -> Fail(trc)
      | Unsatisable(prf) -> cegar P' ( preds ∪ (refine prf))

let slam ( P:cprog) (S:spec) : result =
   cegar (instrument (P,S)) (preds S)
```

100%, and for Windows Driver Model (WDM) drivers, the figure is 97%.

Example. We illustrate the CEGAR process using the SLIC rule from Figure 1a and the example code fragment in Figure 1b. In the program, we have a single spinlock being initialized at line 4. The spinlock is acquired at line 8 and released at line 12. However, both calls KeAcquireSpinLock and KeReleaseSpinLock are guarded by the conditional ($x > 0$). Thus, tracking correlations between such conditionals is important for proving this property. Figures 3a and 3b show the Boolean program obtained by the first application of the *abstract* function to the code from Figures 1a and 1c, respectively.

Figure 3a is the Boolean program abstraction of the SLIC event handler code. Recall that the instrumentation step guarantees there is a unique error state. The function slic _ error at line 1 represents that state; that is, the function slic _ error is unreachable if and only if the program satisfies the SLIC rule. There is one Boolean variable named {state==Locked}; by convention, we name each Boolean variable with the predicate it stands for, enclosed in curly braces. In this case, the predicate comes from the guard in the SLIC rule (Figure 1a, line 8). Lines 5–8 and lines 10–13 of Figure 3a show the Boolean procedures corresponding to the SLIC event handlers SLIC _ KeAcquireSpinLock _ call and SLIC _ KeReleaseSpinLock _ call from Figure 1a.

Figure 3b is the Boolean program abstraction of the SLIC-instrumented C program from Figure 1c. Note the Boolean program has the same control flow as the C program, including procedure calls. However, the conditionals at lines 7 and 12 of the Boolean program are nondeterministic since the Boolean program does not have a predicate that refers to the value of variable *x*. Also note that the references to variables count, devicebuffer, and localbuffer are elided in lines 10 and 11 (replaced by skip statements in the Boolean program) since the Boolean program does not have predicates that refer to these variables.

The abstraction in Figure 3b, though a valid abstraction of the instrumented C, is not strong enough to prove the program conforms to the SLIC rule. In particular, the reachability analysis of the Boolean program performed by the check function will find that slic _ error is reachable via the trace 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, which skips the call to SLIC _ KeAcquireSpinLock _ call at line 8 and performs the call to SLIC _ KeReleaseSpinLock _ call at line 13. Since the Boolean variable state==Lock is false, slic _ error will be called in line 11 of Figure 3a.

SLAM feeds this error trace to the symexec function that executes it symbolically over the instrumented C program in Figure 1c and determines the trace is not executable since the branches in "if" conditions are correlated. In particular, the trace is not executable because there does not exist a value for variable *x* such that ($x > 0$) is false (skipping the body of the first conditional) and such that ($x > 0$) is true (entering the body of the second conditional). That is, the formula $\exists x.(x \leq 0) \wedge (x > 0)$ is unsatisfiable. The result of the refine function is to add the predicate {x>0} to the Boolean program to refine it. This addition results in the Boolean program abstraction in Figure 3c, including the Boolean variable {x>0}, in addition to {state==Locked}.

Using these two Boolean variables, the abstraction in Figure 3c is strong enough to prove slic _ error is unreachable for all possible executions of the Boolean program, and hence SLAM proves this Boolean program satisfies the SLIC rule. Since the Boolean program is constructed to be an overapproximation of the C program in Figure 1c, the C program indeed satisfies the SLIC rule.

From SLAM to SDV

SDV is a completely automatic tool (based on SLAM) device-driver developers can use at compile time. Requiring nothing more than the build script of the driver, the SDV tool runs fully automatically and checks a set of prepackaged API usage rules on the device driver. For every usage rule violated by the driver, SDV presents a possible execution trace through the driver that shows how the rule can be violated.

Figure 3. (a) Boolean program abstraction for locking and unlocking routines; (b) Boolean program: CEGAR iteration 1; (c) Boolean program: CEGAR iteration 2.

```
1  slic_error() { assert(false); }
2
3  bool {state==Locked};
4
5  SLIC_KeAcquireSpinLock_call() {
6     if( {state==Locked}) slic_error();
7     else {state==Locked} := true;
8  }
9
10 SLIC_KeReleaseSpinLock_call() {
11    if( !{state==Locked}) slic_error();
12    else {state==Locked} := false;
13 }
14
```

(a)

```
1  ...
2  ...
3  {state==Locked} := false;
4  KeInitializeSpinLock();
5  ...
6  ...
7  if(*)
8    { SLIC_KeAcquireSpinLock_call();
9      KeAcquireSpinLock(); }
10 skip;
11 skip;
12 if(*)
13   { SLIC_KeReleaseSpinLock_Call();
14     KeReleaseSpinLock(); }
15 ...
16 ...
```

(b)

```
1  bool {x > 0};
2  ...
3  {state==Locked} := false;
4  KeInitializeSpinLock();
5  ...
6  ...
7  if({x>0})
8    {  SLIC_KeAcquireSpinLock_call();
9       KeAcquireSpinLock(); }
10 skip;
11 skip;
12 if({x>0})
13   {  SLIC_KeReleaseSpinLock_Call();
14      KeReleaseSpinLock(); }
15 ..
16 ...
```

(c)

Model checking is often called "push-button" technology,[16] giving the impression that the user simply gives the system to the model checker and receives useful output about errors in the system, with state-space explosion being the only obstacle. In practice, in addition to state-space explosion, several other obstacles can inhibit model checking being a "push-button" technology: First, users must specify the properties they want to check, without which there is nothing for a model checker to do. In complex systems (such as the Windows driver interface), specifying such properties is difficult, and these properties must be debugged. Second, due to the state-explosion problem, the code analyzed by the model checker is not the full system in all its gory complexity but rather the composition of some detailed component (like a device driver) with a so-called "environment model" that is a highly abstract, human-written description of the other components of the system—in our case, kernel procedures of the Windows operating system. Third, to be a practical tool in the toolbox of a driver developer, the model checker must be encapsulated in a script incorporating it in the driver development environment, then feed it with the driver's source code and report results to the user. Thus, creating a push-button experience for users requires much more than just building a good model-checking engine.

Here, we explore the various components of the SDV tool besides SLAM: driver API rules, environment models, scripts, and user interface, describing how they've evolved over the years, starting with the formation of the SDV team in Windows in 2002 and several internal and external releases of SDV.

API rules. Different classes of devices have different requirements, leading to class-specific driver APIs. Thus, networking drivers use the NDIS API, storage drivers use the StorPort and MPIO APIs, and display drivers the WDDM API. A new API called WDF was designed to provide higher-level abstractions for common device drivers. As described earlier, SLIC rules capture API-level interactions, though they are not specific to a particular device driver but to a whole class of drivers that use a common API. Such a specification means the manual effort of writing rules can be amortized by checking the rules on thousands of device drivers using the API. The SDV team has made significant investment in writing API rules and teaching others in Microsoft's Windows organization to write API rules.

We wanted to build a verifier covering all possible behaviors of the program while checking the rule, as opposed to a testing tool that checks the rule on a subset of behaviors covered by the test.

Environment models. SLAM is designed as a generic engine for checking properties of a closed C program. However, a device driver is not a closed program with a main procedure but rather a library with many entry points (registered with and called by the operating system). This problem is standard to both program analysis and model checking.

Before applying SLAM to a driver's code, we first "close" the driver program with a suitable environment consisting of a top layer called the *harness*, a `main` procedure that calls the driver's entry points, and a bottom layer of *stubs* for the Windows API functions that can be called by the device driver. Thus, the harness calls into the driver, and the driver calls the stubs.

Most API rules are local to a driver's entry points, meaning a rule can be checked independently on each entry point. However, some complex rules deal with sequences of entry points. For the rules of the first type, the body of the harness is a nondeterministic switch in which each branch calls a single and different entry point of the driver. For more complex rules, the harness contains a sequence of such nondeterministic switches.

A stub is a simplified implementation of an API function intended to approximate the input-output relation of the API function. Ideally, this relation should be an overapproximation of the API function. In many cases, a driver API function returns a scalar indicating success or failure. In these cases, the API stub usually ends with a nondeterministic switch over possible return values. In many cases, a driver API function allocates a memory object and returns its address, sometimes through an output pointer parameter. In these cases, the harness allocates a small set of such memory objects, and the stub picks up one of them and returns its address.

Scaling rules and models. Initially, we (the SDV team) wrote the API rules in SLIC based on input from driver API

experts. We tested them on drivers with injected bugs, then ran SDV with the rules on real Windows drivers. We discussed the bugs found by the rules with driver owners and API experts to refine the rules. At that time, a senior manager said, "It takes a Ph.D. to develop API rules." Since then, we've invested significant effort in creating a discipline for writing SLIC rules and spreading it among device-driver API developers and testers.

In 2007, the SDV team refined the API rules and formulated a set of guidelines for rule development and driver environment model construction. This helped us transfer rule development to two software engineers with backgrounds far removed from formal verification, enabling them to succeed and later spread this form of rule development to others. Since 2007, driver API teams have been using summer interns to develop new API rules for WDF, NDIS, StorPort, and MPIO APIs and for an API used to write file system mini-filters (such as antiviruses) and Windows services. Remarkably, all interns have written API rules that found true bugs in real drivers.

SDV today includes more than 470 API rules. The latest version SDV 2.0 (released with Windows 7 in 2009) includes more than 210 API rules for the WDM, WDF, and NDIS APIs, of which only 60 were written by formal verification experts. The remaining 150 were written or modified from earlier drafts by software engineers or interns with no experience in formal verification.

Worth noting is that the SLIC rules for WDF were developed during the design phase of WDF, whereas the WDM rules were developed long after WDM came into existence. The formalization of the WDF rules influenced WDF design; if a rule could not be expressed naturally in SLIC, the WDF designers tried to refactor the API to make it easier to verify. This experience showed that verification tools (such as SLAM) can be forward-looking design aids, in addition to being checkers for legacy APIs (such as WDM).

Scripts. SDV includes a set of scripts that perform various functions: combining rules and environment models; detecting source files of a driver and its build parameters; running the SLIC compiler on rules and the C compiler

A unique SLAM contribution is the complete automation of CEGAR for software written in expressive programming languages (such as C).

on a driver's and environment model's source code to generate an intermediate representation (IR); invoking SLAM on the generated IR; and reporting the summary of the results and error traces for bugs found by SLAM in a GUI.

The SDV team worked hard to ensure these scripts would provide a very high degree of automation for the user. The user need not specify anything other than the build scripts used to build the driver.

SDV Experience

The first version of SDV (1.3, not released externally outside Microsoft) found, on average, one real bug per driver in 30 sample drivers shipped with the Driver Development Kit (DDK) for Windows Server 2003. These sample drivers were already well tested. Eliminating defects in the WDK samples is important since code from sample drivers is often copied by third-party driver developers.

Versions 1.4 and 1.5 of SDV were applied to Windows Vista drivers. In the sample WDM drivers shipped with the Vista WDK (WDK, the renamed DDK), SDV found, on average, approximately one real bug per two drivers. These samples were mostly modifications of sample drivers from the Windows Server 2003 DDK, with fixes applied for the defects found by SDV 1.3. The newly found defects were due to improvements in the set of SDV rules and to defects introduced due to modifications in the drivers.

For Windows Server 2008, SDV version 1.6 contained new rules for WDF drivers, with which SDV found one real bug per three WDF sample drivers. The low bug count is explained by simplicity of the WDF driver model described earlier and co-development of sample drivers, together with the WDF rules.

For the Windows 7 WDK, SDV 2.0 found, on average, one new real bug per WDF sample driver and few bugs on all the WDM sample drivers. This data is explained by more focused efforts to refine WDF rules and few modifications in the WDM sample drivers. SDV 2.0 shipped with 74 WDM rules, 94 WDF rules, and 36 NDIS rules. On WDM drivers, 90% of the defects reported by SDV are true bugs, and the rest are false errors. Further, SDV reports nonresults (such as timeouts

and spaceouts) on only 3.5% of all checks. On WDF drivers, 98% of defects reported by SDV are true bugs, and non-results are reported on only 0.04% of all checks. During the development cycle of Windows 7, SDV 2.0 was applied as a quality gate to drivers written by Microsoft and sample drivers shipped with the WDK. SDV was applied later in the cycle after all other tools, yet found 270 real bugs in 140 WDM and WDF drivers. All bugs found by SDV in Microsoft drivers were fixed by Microsoft. We do not have reliable data on bugs found by SDV in third-party device drivers.

Here, we give performance statistics from a recent run of SDV on 100 drivers and 80 SLIC rules. The largest driver in the set is about 30,000 lines of code, and the total size of all drivers is 450,000 lines of code. The total runtime for the 8,000 runs (each driver-rule combination is a run) is about 30 hours on an eight-core machine. We kill a run if it exceeds 20 minutes, and SDV yields useful results (either a bug or a pass) on over 97% of the runs. We thus find SDV checks drivers with acceptable performance, yielding useful results on a large fraction of the runs.

Limitations. SLAM and SDV also involve several notable limitations. Even with CEGAR, SLAM is unable to handle very large programs (with hundreds of thousands of lines of code). However, we also found SDV is able to give useful results for control-dominated properties and programs with tens of thousands of lines of code. Though SLAM handles pointers in a sound manner, in practice, it is unable to prove properties that depend on establishing invariants of heap data structures. SLAM handles only sequential programs, though others have extended SLAM to deal with bounded context switches in concurrent programs.[40] Our experience with SDV shows that in spite of these limitations, SLAM is very successful in the domain of device-driver verification.

Related Work

SLAM builds on decades of research in formal methods. Model checking[15,16,41] has been used extensively to algorithmically check temporal logic properties of models. Early applications of model checking were in hardware[38] and protocol design.[32] In compiler and programming languages, abstract interpretation[21] provides a broad and generic framework to compute fixpoints using abstract lattices. The particular abstraction used by SLAM was called "predicate abstraction" by Graf and Saïdi.[29] Our contribution was to show how to perform predicate abstraction on C programs with such language features as pointers and procedure calls in a modular manner.[2,3] The predicate-abstraction algorithm uses an automated theorem prover. Our initial implementation of SLAM used the Simplify theorem prover.[23] Our current implementation uses the Z3 theorem prover.[22]

The Bandera project explored the idea of user-guided finite-state abstractions for Java programs[20] based on predicate abstraction and manual abstraction but without automatic refinement of abstractions. It also explored the use of program slicing for reducing the state space of models. SLAM was influenced by techniques used in Bandera to check typestate properties on all objects of a given type.

SLAM's Boolean program model checker (Bebop) computes fixpoints on the state space of the generated Boolean program that can include recursive procedures. Bebop uses the Context Free Language Reachability algorithm,[42,43] implementing it symbolically using Binary Decision Diagrams.[11] Bebop was the first symbolic model checker for pushdown systems. Since then, other symbolic checkers have been built for similar purposes,[25,36] and Boolean programs generated by SLAM have been used to study and improve their performance.

SLAM and its practical application to checking device drivers has been enthusiastically received by the research community, and several related projects have been started by research groups in universities and industry. At Microsoft, the ESP and Vault projects were started in the same group as SLAM, exploring different ways of checking API usage rules.[37] The Blast project[31] at the University of California, Berkeley, proposed a technique called "lazy abstraction" to optimize constructing and maintaining the abstractions across the iterations in the CEGAR loop. McMillan[39] proposed "interpolants" as a more systematic and general way to perform refinement; Henzinger et al.[30] found predicates generated from interpolants have nice local properties that were then used to implement local abstractions in Blast.

Other contemporary techniques for analyzing C code against temporal rules include the meta-level compilation approach of Engler et al.[24] and an extension of SPIN developed by Holzmann[33] to handle ANSI C.[33] The Cqual project uses "type qualifiers" to specify API usage rules, using type inference to check C code against the type-qualifier annotations.[26]

SLAM works by computing an overapproximation of the C program, or a "may analysis," as described by Godefroid et al.[28] The may analysis is refined using symbolic execution on traces, as inspired by the PREfix tool,[12] or a "must analysis." In the past few years, must analysis using efficient symbolic execution on a subset of paths in the program has been shown to be very effective in finding bugs.[27] The Yogi project has explored ways to combine may and must analysis in more general ways.[28] Another way to perform underapproximation or must analysis is to unroll loops a fixed number of times and perform "bounded model checking"[14] using satisfiability solvers, an idea pursued by several projects, including CBMC,[18] F-Soft,[34] and Saturn.[1]

CEGAR has been generalized to check properties of heap-manipulating programs,[10] as well as the problem of program termination.[19] The Magic model checker checks properties of concurrent programs where threads interact through message passing.[13] And Qadeer and Wu[40] used SLAM to analyze concurrent programs through an encoding that models all interleavings with two context switches as a sequential program.

Conclusion

The past decade has seen a resurgence of interest in the automated analysis of software for the dual purpose of defect detection and program verification, as well as advances in program analysis, model checking, and automated theorem proving. A unique SLAM contribution is the complete automation of CEGAR for software written in expres-

sive programming languages (such as C). We achieved this automation by combining and extending such diverse ideas as predicate abstraction, interprocedural data-flow analysis, symbolic model checking, and alias analysis. Windows device drivers provided the crucible in which SLAM was tested and refined, resulting in the SDV tool, which ships as part of the Windows Driver Kit.

Acknowledgments

For their many contributions to SLAM and SDV, directly and indirectly, we thank Nikolaj Bjørner, Ella Bounimova, Sagar Chaki, Byron Cook, Manuvir Das, Satyaki Das, Giorgio Delzanno, Leonardo de Moura, Manuel Fähndrich, Nar Ganapathy, Jon Hagen, Rahul Kumar, Shuvendu Lahiri, Jim Larus, Rustan Leino, Xavier Leroy, Juncao Li, Jakob Lichtenberg, Rupak Majumdar, Johan Marien, Con McGarvey, Todd Millstein, Arvind Murching, Mayur Naik, Aditya Nori, Bohus Ondrusek, Adrian Oney, Onur Oyzer, Edgar Pek, Andreas Podelski, Shaz Qadeer, Bob Rinne, Robby, Stefan Schwoon, Adam Shapiro, Rob Short, Fabio Somenzi, Amitabh Srivastava, Antonios Stampoulis, Donn Terry, Abdullah Ustuner, Westley Weimer, Georg Weissenbacher, Peter Wieland, and Fei Xie. C

References

1. Aiken, A., Bugrara, S., Dillig, I., Dillig, T., Hackett, B., and Hawkins, P. An overview of the Saturn project. In *Proceedings of the 2007 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering* (San Diego, June 13–14). ACM Press, New York, 2007, 43–48.
2. Ball, T., Majumdar, R., Millstein, T., and Rajamani, S.K. Automatic predicate abstraction of C programs. In *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Snowbird, UT, June 20–22). ACM Press, New York, 2001, 203–213.
3. Ball, T., Millstein, T.D., and Rajamani, S.K. Polymorphic predicate abstraction. *ACM Transactions on Programming Languages and Systems 27*, 2 (Mar. 2005), 314–343.
4. Ball, T., Podelski, A., and Rajamani, S.K. Boolean and Cartesian abstractions for model checking C programs. In *Proceedings of the Seventh International Conference on Tools and Algorithms for Construction and Analysis of Systems* (Genova, Italy, Apr. 2–6). Springer, 2001, 268–283.
5. Ball, T. and Rajamani, S.K. Bebop: A symbolic model checker for Boolean programs. In *Proceedings of the Seventh International SPIN Workshop on Model Checking and Software Verification* (Stanford, CA, Aug. 30–Sept. 1). Springer, 2000, 113–130.
6. Ball, T. and Rajamani, S.K. *Boolean Programs: A Model and Process for Software Analysis. Technical Report MSR-TR-2000-14.* Microsoft Research, Redmond, WA, Feb. 2000.
7. Ball, T. and Rajamani, S.K. Automatically validating temporal safety properties of interfaces. In *Proceedings of the Eighth International SPIN Workshop on Model Checking of Software Verification* (Toronto, May 19–20). Springer, 2001, 103–122.
8. Ball, T. and Rajamani, S.K. The SLAM project: Debugging system software via static analysis. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Portland, OR, Jan. 16–18). ACM Press, New York, Jan. 2002, 1–3.
9. Ball, T. and Rajamani, S.K. *SLIC: A Specification Language for Interface Checking. Technical Report MSR-TR-2001-21.* Microsoft Research, Redmond, WA, 2001.
10. Beyer, D., Henzinger, T.A., Théoduloz, G., and Zufferey, D. Shape refinement through explicit heap analysis. In *Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering* (Paphos, Cyprus, Mar. 20–28). Springer, 2010, 263–277.
11. Bryant, R. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers C-35*, 8 (Aug. 1986), 677–691.
12. Bush, W.R., Pincus, J.D., and Siela, D.J. A static analyzer for finding dynamic programming errors. *Software-Practice and Experience 30*, 7 (June 2000), 775–802.
13. Chaki, S., Clarke, E., Groce, A., Jha, S., and Veith, H. Modular verification of software components in C. In *Proceedings of the 25th International Conference on Software Engineering* (Portland, OR, May 3–10). IEEE Computer Society, 2003, 385–395.
14. Clarke, E., Grumberg, O., and Peled, D. *Model Checking.* MIT Press, Cambridge, MA, 1999.
15. Clarke, E.M. and Emerson, E.A. Synthesis of synchronization skeletons for branching time temporal logic. In *Proceedings of the Workshop on Logic of Programs* (Yorktown Heights, NY, May 1981). Springer, 1982, 52–71.
16. Clarke, E.M., Emerson, E.A., and Sifakis, J. Model checking: Algorithmic verification and debugging. *Commun. ACM 52*, 11 (Nov. 2009), 74–84.
17. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Counterexample-guided abstraction refinement. In *Proceedings of the 12 International Conference on Computer-Aided Verification* (Chicago, July 15–19). Springer, 2000, 154–169.
18. Clarke, E.M., Kroening, D., and Lerda, F. A tool for checking ANSI-C programs. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Barcelona, Mar. 29–Apr. 2). Springer, 2004, 168–176.
19. Cook, B., Podelski, A., and Rybalchenko, A. Abstraction refinement for termination. In *Proceedings of the 12th International Static Analysis Symposium* (London, Sept. 7–9). Springer, 2005, 87–101.
20. Corbett, J., Dwyer, M., Hatcliff, J., Pasareanu, C., Robby, Laubach, S., and Zheng, H. Bandera: Extracting finite-state models from Java source code. In *Proceedings of the 22nd International Conference on Software Engineering* (Limerick, Ireland, June 4–11). ACM Press, New York, 2000, 439–448.
21. Cousot, P. and Cousot, R. Abstract interpretation: A unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages* (Los Angeles, Jan.). ACM Press, New York, 1977, 238–252.
22. de Moura, L. and Bjørner, N. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Mar. 29–Apr. 6). Springer, 2008, 337–340.
23. Detlefs, D., Nelson, G., and Saxe, J.B. Simplify: A theorem prover for program checking. *Journal of the ACM 52*, 3 (May 2005), 365–473.
24. Engler, D., Chelf, B., Chou, A., and Hallem, S. Checking system rules using system-specific, programmer-written compiler extensions. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation* (San Diego, Oct. 23–25). Usenix Association, 2000, 1–16.
25. Esparza, J. and Schwoon, S. A BDD-based model checker for recursive programs. In *Proceedings of the 13th International Conference on Computer Aided Verification* (Paris, July 18–22). Springer, 2001, 324–336.
26. Foster, J.S., Terauchi, T., and Aiken, A. Flow-sensitive type qualifiers. In *Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Berlin, June 17–19). ACM Press, New York, 2002, 1–12.
27. Godefroid, P., Levin, M.Y., and Molnar, D.A. Automated whitebox fuzz testing. In *Proceedings of the Network and Distributed System Security Symposium* (San Diego, CA, Feb. 10–13). The Internet Society, 2008.
28. Godefroid, P., Nori, A.V., Rajamani, S.K., and Tetali, S.D. Compositional may-must program analysis: Unleashing the power of alternation. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Madrid, Jan. 17–23). ACM Press, New York, 2010, 43–56.
29. Graf, S. and Saïdi, H. Construction of abstract state graphs with PVS. In *Proceedings of the Ninth International Conference on Computer-Aided Verification* (Haifa, June 22–25). Springer, 72–83.
30. Henzinger, T.A., Jhala, R., Majumdar, R., and McMillan, K.L. Abstractions from proofs. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Venice, Jan. 14–16). ACM Press, New York, 2004, 232–244.
31. Henzinger, T.A., Jhala, R., Majumdar, R., and Sutre, G. Lazy abstraction. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium Principles of Programming Languages* (Portland, OR, Jan. 16–18). ACM Press, New York, 2002, 58–70.
32. Holzmann, G. The SPIN model checker. *IEEE Transactions on Software Engineering 23*, 5 (May 1997), 279–295.
33. Holzmann, G. Logic verification of ANSI-C code with SPIN. In *Proceedings of the Seventh International SPIN Workshop on Model Checking and Software Verification* (Stanford, CA, Aug. 30–Sept. 1). Springer, 2000, 131–147.
34. Ivancic, F., Yang, Z., Ganai, M.K., Gupta, A., and Ashar, P. Efficient SAT-based bounded model checking for software verification. *Theoretical Computer Science 404*, 3 (Sept. 2008), 256–274.
35. Kurshan, R. *Computer-aided Verification of Coordinating Processes.* Princeton University Press, Princeton, NJ, 1994.
36. La Torre, S., Parthasarathy, M., and Parlato, G. Analyzing recursive programs using a fixed-point calculus. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Dublin, June 15–21). ACM Press, New York, 2009, 211–222.
37. Larus, J.R., Ball, T., Das, M., DeLine, R., Fähndrich, M., Pincus, J., Rajamani, S.K., and Venkatapathy, R. Righting software. *IEEE Software 21*, 3 (May/June 2004), 92–100.
38. McMillan, K. *Symbolic Model Checking: An Approach to the State-Explosion Problem.* Kluwer Academic Publishers, 1993.
39. McMillan, K.L. Interpolation and SAT-based model checking. In *Proceedings of the 15th International Conference on Computer-Aided Verification* (Boulder, CO, July 8–12). Springer, 2003, 1–13.
40. Qadeer, S. and Wu, D. KISS: Keep it simple and sequential. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation* (Washington, D.C., June 9–12). ACM Press, New York, 2004, 14–24.
41. Queille, J. and Sifakis, J. Specification and verification of concurrent systems in CESAR. In *Proceedings of the Fifth International Symposium on Programming* (Torino, Italy, Apr. 6–8). Springer, 1982, 337–350.
42. Reps, T., Horwitz, S., and Sagiv, M. Precise interprocedural data flow analysis via graph reachability. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, Jan. 23–25). ACM Press, New York, 1995, 49–61.
43. Sharir, M. and Pnueli, A. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Applications*, N.D. Jones and S.S. Muchnick, Eds. Prentice-Hall, 1981, 189–233.
44. Vardi, M.Y. and Wolper, P. An automata theoretic approach to automatic program verification. In *Proceedings of the Symposium Logic in Computer Science* (Cambridge, MA, June 16–18). IEEE Computer Society Press, 1986, 332–344.

Thomas Ball (tball@microsoft.com) is a principal researcher, managing the Software Reliability Research group in Microsoft Research, Redmond, WA.

Vladimir Levin (vladlev@microsoft.com) is a principal software design engineer and the technical lead of the Static Driver Verification project in Windows in Microsoft, Redmond, WA.

Sriram Rajamani (sriram@microsoft.com) is assistant managing director of Microsoft Research India, Bangalore.

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965744

The volunteer search for Jim Gray, lost at sea in 2007, highlights the challenges of computer-aided emergency response.

BY JOSEPH M. HELLERSTEIN AND DAVID L. TENNENHOUSE
(ON BEHALF OF A LARGE TEAM OF VOLUNTEERS)

Searching for Jim Gray: A Technical Overview

ON SUNDAY JANUARY 28, 2007, noted computer scientist Jim Gray disappeared at sea in his sloop *Tenacious*. He was sailing singlehanded, with plans to scatter his mother's ashes near the Farallon Islands, some 27 miles outside San Francisco's Golden Gate. As news of Gray's disappearance spread through his social network, his friends and colleagues began discussing ways to mobilize their skills and resources to help authorities locate *Tenacious* and rescue Gray. That discussion evolved over days and weeks into an unprecedented civilian search-and-rescue (SAR[a]) exercise involving satellites, private planes, automated image analysis, ocean current simulations, and crowdsourced human computing, in collaboration with the U.S. Coast Guard. The team that emerged included computer scientists, engineers, graduate students, oceanographers, astronomers, business leaders, venture capitalists, and entrepreneurs, many of whom had never met one another before. There was ample access to funds, technol-

a SAR also refers to synthetic aperture radar, a remote-imaging technology employed in the search for *Tenacious*; using it here, we refer exclusively to search-and-rescue.

» key insights

- **Loosely coupled teams quickly evolved software polytechtures with varying interfaces, decoupling data acquisition from analysis to enable use of expertise at a distance.**
- **The U.S. Coast Guard developed software to aid search and rescue and is an interesting potential research partner for computer scientists.**
- **New open-source tools and research could help with group coordination, crowdsourced image acquisition, high-volume image processing, ocean drift modeling, and analysis of open-water satellite imagery.**

ogy, organizational skills and know-how, and a willingness to work round the clock.

Even with these advantages, the odds of finding *Tenacious* were never good. On February 16, 2007, in consultation with the Coast Guard and Gray's family, the team agreed to call off the search. *Tenacious* remains lost to this day, despite a subsequent extensive underwater search of the San Francisco coastline.[4]

Gray was famous for many things, including his determination to work with practitioners to transform the practical challenges they faced into scientific questions that could be formalized and addressed by the research community. As the search for *Tenacious* wound down, a number of us felt that even though the effort was not successful on its own terms, it offered a Jim-Gray-like opportunity to convert the particulars of the experience into higher-level technical observations of more general interest. One goal was to encourage efforts to "democratize" the ability of families and friends to use technology to assist SAR, so people whose social network is not as well-connected as Gray's could undertake analogous efforts. In addition, we hoped to review the techniques we used and ask how to improve them further to make the next search effort more effective. To that end, in May 2008, the day after a public tribute to Gray at the University of California, Berkeley, we convened a meeting of search participants, including the Coast Guard. This was the first opportunity for the virtual organization that had searched for *Tenacious* to meet face-to-face and compare stories and perspectives.

One sober conclusion the group quickly reached was that its specific lessons on maritime SAR could have only modest impact, as we detail here. However, we still felt it would be constructive to cull lessons learned and identify technical challenges. First, maritime search is not a solved problem, and even though the number of lives to be saved is small, each life is precious. Second, history shows that technologies developed in one application setting often have greater impact in others. We were hopeful that lessons learned searching for Gray could inform efforts launched during larger life-threatening scenarios, including civilian-driven efforts toward disaster response and SAR during natural disasters and military conflict. Moreover, as part of the meeting, we also brainstormed about the challenges of safety and prevention.

This article aims to distill some of that discussion within computer science, which is increasingly interested in disaster response (such as following the 2007 Kenyan election crisis[1] and 2010 Haiti earthquake[2]). We document the emergent structure of the team and its communication, the "polytechture" of the systems built during the search, and some of the related challenges; a longer version of this article[3] includes additional figures, discussion, and technical challenges.

Background

The amateur effort to find *Tenacious* and its skipper began with optimism but little context as to the task at hand. We had no awareness of SAR practice and technology, and only a vague sense of the special resources Gray's friends could bring to bear on a problem. With the benefit of hindsight, we provide a backdrop for our discussion of computer science challenges in SAR, reflecting first on the unique character of the search for *Tenacious*, then on the basics of maritime SAR as practiced today.

***Tenacious* SAR.** The search for *Tenacious* was in some ways unique and in others a typical volunteer SAR. The uniqueness had its roots in Gray's persona. In addition to being a singular scientist and engineer, he was distinctly social, cultivating friendships and collaborations across industries and sciences. The social network he built over decades brought enormous advantages to many aspects of the search, in ways that would be very difficult to replicate. First, the team that assembled to find *Tenacious* included leaders in such diverse areas as computing, astronomy, oceanography, and business management. Second, due to Gray's many contacts in the business and scientific worlds, funds and resources were essentially unlimited, including planes, pilots, satellite imagery, and control of well-provisioned computing resources. Finally, the story of famous-scientist-gone-missing attracted significant media interest, providing public awareness that attracted help with manual image analysis and information on sightings of debris and wreckage.

On the other hand, a number of general features the team would wrestle with seem relatively universal to volunteer SAR efforts. First, the search got off to a slow start, as volunteers emerged and organized to take concrete action. By the time all the expertise was in place, the odds of finding a survivor or even a boat were significantly diminished. Second, almost no one involved in the volunteer search had any SAR experience. Finally, at every stage of the search, the supposition was that it would last only a day or two more. As a result, there were disincentives to invest time in improving existing practices and tools and positive incentives for decentralized and lightweight development of custom-crafted tools and practices.

If there are lessons to be learned, they revolve around questions of both the uniqueness of the case and its universal properties. The first category motivated efforts to democratize techniques used to search for *Tenacious*, some of which didn't have to be as complex or expensive as they were in this instance. The second category motivated efforts to address common technological problems arising in any volunteer emergency-response situation.

Maritime SAR. Given our experience, maritime SAR is the focus of our discussion here. As it happens, maritime SAR in the U.S. is better understood and more professionally conducted than land-based SAR. Maritime SAR is the responsibility of a single federal agency: the Coast Guard, a branch of the U.S. Department of Homeland Security. By contrast, land-based SAR is managed in an ad hoc manner by local law-enforcement authorities. Our experience with the Coast Guard was altogether positive; not only were its members eminently good at their jobs, they were technically sophisticated and encouraging of our (often naïve) ideas, providing advice and coordination despite their own limited time and resources. In the U.S. at least, maritime settings are a good incubator

for development of SAR technology, and the Coast Guard is a promising research partner. As of the time of writing, its funding is modest, so synergies and advocacy from well-funded computer-science projects would likely be welcome.

In hindsight, the clearest lessons for the volunteer search team were that the ocean is enormous, and the Coast Guard has a sophisticated and effective maritime SAR program. The meeting in Berkeley opened with a briefing from Arthur Allen, an oceanographer at the Coast Guard Headquarters Office of Search and Rescue, which oversees all Coast Guard searches, with an area of responsibility covering most of the Pacific, half of the Atlantic, and half of the Arctic Oceans. Here, we review some of the main points Allen raised at the meeting.

SAR technology is needed only when people get into trouble. From a public-policy perspective, it is cheaper and more effective to invest in preventing people from getting into trouble than in ways of saving them later; further discussion of boating safety can be found at http://www.uscgboating.org. We cannot overemphasize the importance of safety and prevention in saving lives; the longer version of this article[3] includes more on voluntary tracking technologies and possible extensions.

Even with excellent public safety, SAR efforts are needed to handle the steady stream of low-probability events triggered by people getting into trouble. When notification of trouble is quick, the planning and search phases become trivial, and the SAR activity can jump straight to rescue recovery. SAR is more difficult when notification is delayed, as it was with Gray. This leads to an iterative process of planning and search. Initial planning is intended to be quick, often consisting simply of the decision to deploy planes for a visual sweep of the area where a boat is expected to be. When an initial "alpha" search is not successful, the planning phase becomes more deliberate. The second, or "bravo," search is planned via software using statistical methods to model probabilities of a boat's location. The Coast Guard developed a software package for this process called SAROPS,[5] which treats the boat-location task as a probabilistic planning problem it addresses with Bayesian machine-learning techniques. The software accounts for prior information about weather and ocean conditions and the properties of the missing vessel, as well as the negative information from the alpha search. It uses a Monte Carlo particle-filtering approach to infer a distribution of boat locations, making suggestions for optimal search patterns. SAROPS is an ongoing effort updated with models of various vessels in different states, including broken mast, rudder missing, and keel missing. The statistical training experiments to parameterize these models are expensive exercises that place vessels underway to track their movement. The Coast Guard continues to conduct these experiments on various parameters as funds and time permit. No equivalent software package or methodology is currently available for land-based SAR.

Gray was famous for many things, including his determination to work with practitioners to transform the practical challenges they faced into scientific questions that could be formalized and addressed by the research community.

Allen shared the Coast Guard's SAR statistics, 2003–2006, which are included in the longer version of this article.[3] They show that most cases occur close to shore, with many involving land-based vehicles going into the ocean. The opportunities for technologists to assist with maritime SAR are modest. In Allen's U.S. statistics, fewer than 1,000 lives were confirmed lost in boating accidents each year, and only 200 to 300 deaths occur after the Coast Guard had been notified and thus might have been avoided through rescue. A further 600 people per year remain unaccounted for, and, while it is unknown how many of them remained alive post-notification, some fraction of them are believe to have committed suicide. Relative to other opportunities to save lives through technology, the margin for improvement in maritime SAR is relatively small. This reality frames the rest of our discussion, focusing on learning lessons from our experience that apply to SAR and hopefully other important settings as well.

Communication and Coordination

As in many situations involving groups of people sharing a common goal, communication and coordination were major aspects of the volunteer search for Gray. Organizing these "back-office" tasks were ad hoc and

evolving, and, in retrospect, interesting patterns emerged around themes related to social computing, including organizational development, brokering of volunteers and know-how, and communicating with the media and general public. Many could be improved through better software.

Experience. The volunteer effort began via overlapping email threads among Gray's colleagues and friends in the hours and days following his disappearance. Various people exchanged ideas about getting access to satellite imagery, hiring planes, and putting up missing-person posters. Many involved reaching out in a coordinated and thoughtful manner to third parties, but it was unclear who heard what information and who might be contacting which third parties. To solve that problem a blog called "*Tenacious* Search" was set up to allow a broadcast style of communication among the first group of participants. Initially, authorship rights on the blog were left wide open. This simple "blog-as-bulletin-board" worked well for a day or two for coordinating those involved in the search, loosely documenting our questions, efforts, skills, and interests in a way that helped define the group's effort and organization.

Within a few days the story of Gray's disappearance was widely known, however, and the blog transitioned from in-group communication medium to widely read publishing venue for status reports on the search effort, serving this role for the remainder of the volunteer search. This function was quickly taken seriously, so authorship on the blog was closed to additional members, and a separate "Friends of Jim" mailing list was set up for internal team communications. This transition led to an increased sense of organizational and social structure within the core group of volunteers.

Over the next few days, various individuals stepped into unofficial central roles for reasons of expedience or unique skills or both. The blog administrator evolved into a general "communications coordinator," handling messages sent to a public email box for tips, brokering skill-matching for volunteers, and serving as a point of contact with outside parties. Another volunteer emerged as "aircraft coordinator," managing efforts to find, pilot, and route private planes and boats to search for Gray. A third volunteer assumed the role of "analysis coordinator," organizing various teams on image analysis and ocean-drift modeling at a various organizations in the U.S. A fourth was chosen by Gray's family to serve as "media coordinator," the sole contact for press and public relations. These coordinator roles were identified in retrospect, and the role names were coined for this article to clarify the discussion. Individuals with management experience in the business world provided guidance along the way, but much of the organizational development happened in an organic "bottom-up" mode.

Interesting patterns emerged around themes related to social computing, including organizational development, brokering of volunteers and know-how, and communicating with the media and general public.

On the communications front, an important role that quickly emerged was the brokering of tasks between skilled or well-resourced volunteers and people who could take advantage of those assets. This began in an ad hoc broadcast mode on the blog and email lists, but, as the search progressed, offers of help came from unexpected sources, and coordination and task brokering became more complex. Volunteers with science and military backgrounds emerged with offers of specific technical expertise and suggestions for acquiring and analyzing particular satellite imagery. Others offered to search in private planes and boats, sometimes at serious risk to their own lives, and so were discouraged by the team and the Coast Guard. Yet others offered to post "Missing Sailor" posters in marinas, also requiring coordination. Even psychic assistance was offered. Each offer took time from the communications coordinator to diplomatically pursue and route or deflect. As subteams emerged within the organization, this responsibility became easier; the communications coordinator could skim an inbound message and route it to one of the other volunteer coordinators for follow-up.

Similar information-brokering challenges arose in handling thousands of messages from the general public, after being encouraged by the media to keep their eyes open for boats and debris, reporting to a public email address. The utility of many of these messages was ambiguous,

Some of the remote imagery sources considered during the search for Jim Gray.

RADARSAT-1	A commercial earth-observing satellite (EOS) from Canada, whose products are distributed by MDA Geospatial Services. NASA has access to RADARSAT-1 data, in exchange for having provided a rocket to launch the satellite; http://en.wikipedia.org/wiki/RADARSAT-1
Ikonos	A commercial EOS operated by GeoEye (U.S.); http://en.wikipedia.org/wiki/IKONOS
QuickBird	A commercial EOS owned and operated by Digital Globe (U.S.) in use at the time by Google Earth and Microsoft Virtual Earth; http://en.wikipedia.org/wiki/QuickBird
ER-2	A high-altitude aircraft operated by NASA similar to the U.S. Air Force U2-S reconnaissance platform; http://www.nasa.gov/centers/dryden/research/AirSci/ER-2/index.html
SPOT-5	A commercial EOS operated by SPOT Image (France); http://en.wikipedia.org/wiki/SPOT_(satellites)
Envisat	A commercial EOS launched by the European Space Agency. Data products are distributed by the SARCOM consortium, created and led by SPOT Image; http://en.wikipedia.org/wiki/Envisat

and, given the sense of urgency, it was often difficult to decide whether to bring them to the attention of busy people: the Coast Guard, the police, Gray's family, and technical experts in image analysis and oceanography. In some cases, tipsters got in contact repeatedly, and it became necessary to assemble conversations over several days to establish a particular tipster's credibility. This became burdensome as email volume grew.

Discussion. On reflection, the organization's evolution was one of the most interesting aspects of its development. Leadership roles emerged fairly organically, and subgroups formed with little discussion or contention over process or outcome. Some people had certain baseline competencies; for example, the aircraft coordinator was a recreational pilot, and the analysis coordinator had both management experience and contacts with image-processing experts in industry and government. In general, though, leadership developed by individuals stepping up to take responsibility and others stepping back to let them do their jobs, then jumping in to help as needed. The grace with which this happened was a bit surprising, given the kind of ambitious people who had surrounded Gray, and the fact that the organization evolved largely through email. The evolution of the team seems worthy of a case study in ad hoc organizational development during crisis.

It became clear that better software is needed to facilitate group communication and coordination during crises. By the end of the search for *Tenacious*—February 16, 2007—various standard communication methods were in use, including point-to-point email and telephony, broadcast via blogs and Web pages, and multicast via conference calls, wikis, and mailing lists. This mix of technologies was natural and expedient in the moment but meant communication and coordination were a challenge. It was difficult to work with the information being exchanged, represented in natural-language text and stored in multiple separate repositories. As a matter of expedience in the first week, the communications coordinator relied on mental models of basic information, like who knew what information and who was working on what tasks. Emphasizing mental note taking made sense in the short term but limited the coordinator's ability to share responsibility with others as the "crisis watch" extended from hours to days to weeks.

Various aspects of this problem are addressable through well-known information-management techniques. But in using current communication software and online services, it remains difficult to manage an evolving discussion that includes individuals, restricted groups, and public announcements, especially in a quickly changing "crisis mode" of operation. Identifying people and their relationships is challenging across multiple communication tools and recipient endpoints. Standard search and visualization metaphors—folders, tags, threads—are not well-matched to group coordination.

Brokering volunteers and tasks introduces further challenges, some discussed in more detail in the longer version of this article.[3] In any software approach to addressing them, one constraint is critical: In an emergency, people do not reach for new software tools, so it is important to attack the challenges in a way that augments popular tools, rather than seeking to replace or recreate them.

Imagery Acquisition

When the volunteer search began, our hope was to use our special skills and resources to augment the Coast Guard with satellite imagery and private planes. However, as we learned, real-time search for boats at sea is not as simple as getting a satellite feed from a mapping service or borrowing a private jet.

Experience. The day after *Tenacious* went missing, Gray's friends and colleagues began trying to access satellite imagery and planes. One of the first connections was to colleagues in earth science with expertise in remote sensing. In an email message in the first few days concerning the difficulty of using satellite imagery to find *Tenacious*, one earth scientist said, "The problem is that the kind of sensors that can see a 40ft (12m) boat have a correspondingly narrow field of view, i.e., they can't see too far either side of straight down... So if they don't just happen to be overhead when you need them, you may have a long wait before they show up again. ...[A]t this resolution, it's strictly target-of-opportunity."

Undeterred, the team pursued multiple avenues to acquire remote imagery through connections at NASA and other government agencies, as well as at various commercial satellite-imagery providers, while the satellite-data teams at both Google and Microsoft directed us to their commercial pro-

vider, Digital Globe. The table here outlines the data sources considered during the search.

As we discovered, distribution of satellite data is governed by national and international law. We attempted from the start to get data from the SPOT-5 satellite but were halted by the U.S. State Department, which invoked the International Charter on Space and Major Disasters to claim exclusive access to the data over the study area, retroactive to the day before our request. We also learned, when getting data from Digital Globe's QuickBird satellite, that full-resolution imagery is available only after a government-mandated 24-hour delay; before that time, Digital Globe could provide only reduced-resolution images.

The first data acquired from the QuickBird satellite was focused well south of San Francisco, near Catalina Island, and the odds of *Tenacious* being found in that region were short. On the other hand, it seemed important to begin experimenting with real data to see how effectively the team could process it, and this early learning proved critical to getting the various pieces of the image-processing pipeline in place and tested. As the search progressed, Digital Globe was able to acquire imagery solidly within the primary search area, and the image captures provided to the team were some of the biggest data products Digital Globe had ever generated: more than 87 gigapixels. Even so, the areas covered by the satellite captures were dwarfed by the airborne search conducted by the Coast Guard immediately after Gray went missing (see Figure 5 of the longer version of this article[3]).

We were able to establish contacts at NASA regarding planned flights of its ER-2 "flying laboratory" aircraft over the California coast. The ER-2 is typically booked on scientific missions and requires resources—fuel, airport time, staffing, wear-and-tear—to launch under any circumstances. As it happened, the ER-2 was scheduled for training flights in the area where *Tenacious* disappeared. Our contacts were able to arrange flight plans to pass over specific areas of interest and record various forms of digital imagery due to a combination of fortunate circumstance and a well-connected social network. Unfortunately, a camera failure early in the ER-2 flight limited data collection.

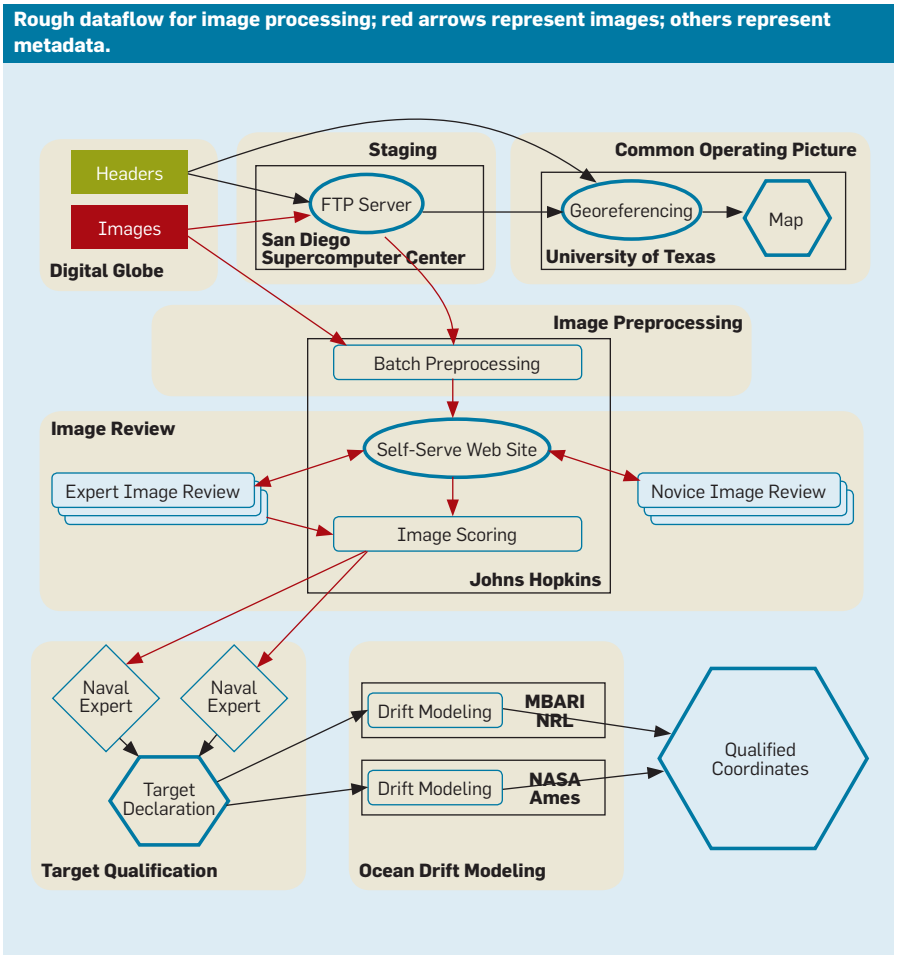
In addition to these relatively rare imaging resources, we chartered private planes to fly over the ocean, enabling volunteer spotters to look for *Tenacious* with their naked eyes and record digital imagery. This effort ended up being more limited than we expected. One cannot simply charter or borrow a private jet and fly it out over the ocean. Light planes are not designed or allowed to fly far offshore. Few people maintain planes equipped for deep-sea search, and flights over deep sea can be undertaken only by pilots with appropriate maritime survival training and certification. Finally, aircraft of any size require a flight plan to be filed and approved with a U.S. Flight Service Station in order to cross the U.S. Air Defense Identification Zone beginning a few miles offshore. As a result of these limitations and many days of bad weather, we were able to arrange only a small number of private overflights, with all but one close to shore.

Another source of imagery considered was land-based video cameras that could perhaps have more accurately established a time of departure for *Tenacious*, beyond what we knew from Gray's mobile phone calls to family members on his way out. The Coast Guard operates a camera on the San Francisco Bay that is sometimes pointed out toward the Golden Gate and the ocean, but much of the imagery captured for that day was in a state of "white-out," rather than useful imagery, perhaps due to foggy weather.

Discussion. The search effort was predicated on quick access to satellite imagery and was surprisingly successful, with more than 87 gigapixels of satellite imagery acquired from Digital Globe alone within about four days of capture. Yet in retrospect we would have wanted much more data, with fewer delays. The longer version of this article[3] reviews some of the limitations we encountered, as well as ideas for improving the ability to acquire imagery in life-threatening emergencies.

Policy concerns naturally come up

Rough dataflow for image processing; red arrows represent images; others represent metadata.



when discussing large volumes of remote imagery, and various members of the amateur team voiced concern about personal privacy during the process. Although popular media-sharing Web sites provide widespread access to crowdsourced and aggregated imagery, they have largely confined themselves to benign settings (such as tourism and ornithology), whereas maritime SAR applications (such as monitoring marinas and shipping lanes) seem closer to pure surveillance. The potential for infringing on privacy raises understandable concern, and the policy issues are not simple. Perhaps our main observation on this front was the need for a contextual treatment of policy, balancing general-case social concerns against specific circumstances for using the data, in our case, trying to rescue a friend. On the other hand, while the search for *Tenacious* and its lone sailor was uniquely urgent for us, similar life-and-death scenarios occur on a national scale with some frequency. So, we would encourage research into technical solutions that can aggressively harvest and process imagery while provably respecting policies that limit image release based on context.

From Imagery to Coordinates

Here, we discuss the processing pipeline(s) and coordination mechanisms used to reduce the raw image data to qualified search coordinates—the locations to which planes were dispatched for a closer look. This aspect of the search was largely data-driven, involving significant technical expertise and much more structured and tool-intensive processes than those described earlier. On the other hand, since time was short and the relevant expertise so specialized, it also led to simple interfaces between teams and their software. The resulting amalgam of software was not the result of a specific architecture, in the usual sense of the word (archi- "chief" + techton "builder"). A more apt term for the software and workflow described here might be a polytechture, the kind of system that emerges from the design efforts of many independent actors.

Overview. The figure here outlines the ultimate critical-path data and control flow that emerged, depicting the ad hoc pipeline developed for Digital Globe's satellite imagery. In the paragraphs that follow, we also discuss the Mechanical Turk pipeline developed early on and used to process NASA ER-2 overflight imagery but that was replaced by the ad hoc pipeline.

Before exploring the details, it would be instructive to work "upstream" through the pipeline, from final qualified targets back to initial imagery. The objective of the image-processing effort was to identify one or more sets of qualified search coordinates to which aircraft could be dispatched (lower right of the figure). To do so, it was not sufficient to simply identify the coordinates of qualified targets on the imagery; rather, we had to apply a mapping function to the coordinates to compensate for drift of the target from the time of image capture to flight time. This mapping function was provided by two independent "drift teams" of volunteer oceanographers, one based at the Monterey Bay Aquarium Institute and Naval Research Lab, another at NASA Ames ("Ocean Drift Modeling" in the figure).

The careful qualification of target coordinates was particularly important. It was quickly realized that many of the potential search coordinates would be far out at sea and, as mentioned earlier, require specialized aircraft and crews. Furthermore, flying low-altitude search patterns offshore in single-engine aircraft implied a degree of risk to the search team. Thus, it was incumbent on the analysis team to weigh this risk before declaring a target to be qualified. A key step in the process was a review of targets by naval experts prior to their final qualification ("Target Qualification" in the figure).

Prior to target qualification, an enormous set of images had to be reviewed and winnowed down to a small set of candidates that appeared to contain boats. To our surprise and disappointment, there were no computer-vision algorithms at hand well suited to this task, so it was done manually. At first, image-analysis tasking was managed using Amazon's Mechanical Turk infrastructure to coordinate volunteers from around the world. Subsequently, a distributed team of volunteers with expertise in image analysis used a collection of ad hoc tools to coordinate and perform the review function ("Image Review" in the figure).

Shifting to the start of the pipeline, each image data set required a degree of preprocessing prior to human analysis of the imagery, a step performed by members of Johns Hopkins's Department of Physics and Astronomy in collaboration with experts at CalTech and the University of Hawaii. At the same time, a separate team at the University of Texas's Center for Space Research georeferenced the image-file headers onto a map included in a Web interface for tracking the progress of image analysis ("Image Preprocessing," "Common Operating Picture," and "Staging" in the figure).

The eventual workflow was a distributed, multiparty process. Its components were designed and built individually, "bottom-up," by independent volunteer teams at various institutions. The teams also had to quickly craft interfaces to stitch together the end-to-end workflow with minimal friction. An interesting and diverse set of design styles emerged, depending on a variety of factors. In the following sections, we cover these components in greater detail, this time from start to finish:

Preprocessing. Once the image providers had data and the clearance to send it, they typically sent notification of availability via email to the image-analysis coordinator, together with an ftp address and the header file describing the collected imagery ("the collection").

Upon notification, the preprocessing team at Johns Hopkins began copying the data to its cluster. Meanwhile, the common storage repository at the San Diego Supercomputer Center began ftp-ing the data to ensure its availability, with a copy of the header passed to a separate geo-coordination team at the University of Texas that mapped the location covered by the collection, adding it to a Web site. That site provided the overall shared picture of imagery collected and analyses completed and was used by many groups within the search team to track progress and solicit further collections.

Analysis tasking and result processing. Two approaches to the parallel processing of the tiled images were used during the course of the search.

In each, image tiles (or smaller sub-tiles) had to be farmed out to human analysts and the results of their analysis collated and further filtered to avoid a deluge of false positives.

The initial approach was to use Amazon's Mechanical Turk service to solicit and task a large pool of anonymous reviewers whose credentials and expertise were not known to us.

Mechanical Turk is a "crowdsourcing marketplace" for coordinating the efforts of humans performing simple tasks from their own computers. Given that the connectivity and display quality available to them was unknown, the Mechanical Turk was configured to supply users with work items called Human Interface Tasks (HITs), each consisting of a few 300×300-pixel image sub-tiles. Using a template image we provided of what we were looking for, the volunteers were asked to score each sub-tile for evidence of similar features and provide comments on artifacts of interest. This was an exceedingly slow process due to the number of HITs required to process a collection.

In addition to handling the partitioning of the imagery across volunteers, Mechanical Turk bookkeeping was used to ensure that each sub-tile was redundantly viewed by multiple volunteers prior to declaring the pipeline "complete." Upon completion, and at checkpoints along the way, the system also generated reports aggregating the results received concerning each sub-tile.

False positives were a significant concern, even in the early stages of processing. So a virtual team of volunteers who identified themselves as having some familiarity with image analysis (typically astronomical or medical imagery rather than satellite imagery) was assembled to perform this filtering. In order to distribute the high-scoring sub-tiles among them, the image-analysis team configured an iterative application of Mechanical Turk accessible only to the sub-team, with the high-scoring sub-titles from the first pipeline fed into it. The coordinator then used the reports generated by this second pipeline to drive the target-qualification process. This design pattern of an "expertise hierarchy" seems likely to have application in other crowdsourcing settings.

A significant cluster of our image reviewers were co-located at the Johns Hopkins astronomy research center. These volunteers, with ample expertise, bandwidth, high-quality displays, and a sense of personal urgency, realized they could process the imagery much faster than novices scheduled by Mechanical Turk. This led to two modifications in the granularity of tasking: larger sub-tiles and a Web-based visual interface to coordinate downloading them to client-specific tools.

They were accustomed to looking for anomalies in astronomical imagery and were typically able to rapidly display, scan, and discard sub-tiles that were three-to-four times larger than those presented to amateurs. This ability yielded an individual processing rate of approximately one (larger) subtile every four seconds, including tiles requiring detailed examination and entry of commentary, as compared to the 20–30-second turnaround for each Mechanical Turk HIT. The overall improvement in productivity over Mechanical Turk was considerably better than these numbers indicate, because the analysts' experience reduced the overhead of redundant analysis, and their physical proximity facilitated communication and cross-training.

A further improvement was that the 256 sub-tiles within each full-size tile were packaged into a single zip file. Volunteers could then use their favorite image-browsing tools to page from one sub-tile to the next with a single mouse click. To automate tasking and results collection, this team used scripting tools to create a Web-based visual interface through which it (and similarly equipped volunteers worldwide) could visually identify individual tiles requiring work, download them, and then submit their reports.

In this interface, tiles were superimposed on a low-resolution graphic of the collection that was, in turn, geo-referenced and superimposed on a map. This allowed the volunteers to prioritize their time by working on the most promising tiles first (such as those not heavily obscured by cloud cover).

The self-tasking capability afforded by the visual interface also supported collaboration and coordination among the co-located expert analysts who worked in "shifts" and had subteam leaders who would gather and score the most promising targets. Though scoring of extremely promising targets was performed immediately, the periodic and collective reviews that took place at the end of each shift promoted discussion among the analysts, allowing them to learn from one another and adjust their individual standards of reporting.

In summary, we started with a system centered on crowdsourced amateur analysts and converged on a solution in which individuals with some expertise, though not in this domain, were able to operate at a very quick pace, greatly outperforming the crowdsourced alternative. This operating point, in and of itself, was an interesting result.

Target qualification. The analysis coordinator examined reports from the analysis pipelines to identify targets for submission to the qualification step. With Mechanical Turk, this involved a few hours sifting through the output of the second Mechanical Turk stage. Once the expert pipeline was in place, the coordinator needed to examine only a few filtered and scored targets per shift.

Promising targets were then submitted to a panel of two reviewers, each with expertise in identifying engineered artifacts in marine imagery. The analysis coordinator isolated these reviewers from one another, in part to avoid cross-contamination, but also from having to individually carry the weight of a potentially risky decision to initiate a search mission while avoiding overly biasing them in a negative direction. Having discussed their findings with each reviewer, the coordinator would then make the final decision to designate a target as qualified and thus worthy of search.

Given the dangers of deep-sea flights, this review step included an intentional bias by imposing less rigorous constraints on targets that had likely drifted close to shore than on those farther out at sea.

Drift modeling. Relatively early in the analysis process, a volunteer with marine expertise recognized that, should a target be qualified, it would be necessary to estimate its move-

ment since the time of image capture. A drift-modeling team was formed, ultimately consisting of two sub-teams of oceanographers with access to two alternative drift models. As image processing proceeded, these sub-teams worked in the background to parameterize their models with weather and ocean-surface data during the course of the search. Thus, once targets were identified, the sub-teams could quickly estimate likely drift patterns.

The drift models utilized a particle-filtering approach of virtual buoys that could be released at an arbitrary time and location, and for which the model would then produce a projected track and likely endpoint at a specified end time. In practice, one must release a string of adjacent virtual buoys to account for the uncertainty in the initial location and the models' sensitivity to local effects that can have fairly large influence on buoy dispersion. The availability of two independent models, with multiple virtual buoys per model, greatly increased our confidence in the prediction of regions to search.

Worth noting is that, although these drift models were developed by leading scientists in the field, the results often involved significant uncertainty. This was particularly true in the early part of the search, when drift modeling was used to provide a "search box" for Gray's boat and had to account for many scenarios, including whether the boat was under sail or with engines running. These scenarios reflected very large uncertainty and led to large search boxes. By the time the image processing and weather allowed for target qualification, the plausible scenario was reduced to a boat adrift from a relatively recent starting point. Our colleagues in oceanography and the Coast Guard said the problem of ocean-drift modeling merits more research and funding; it would also seem to be a good area for collaboration with computer science.

The drift-modeling team developed its own wiki-based workflow interface. The analysis coordinator was given a Web site where he could enter a request to release virtual "drifters" near a particular geolocation at a particular time. Requests were processed by the two trajectory-modeling teams,

A more apt term for the software and workflow described here might be a polytechture, the kind of system that emerges from the design efforts of many independent actors.

and the resulting analysis, including maps of likely drift patterns, were posted back to the coordinator via the drift team's Web site. Geolocations in latitude/longitude are difficult to transcribe accurately over the phone, so using the site helped ensure correct inputs to the modeling process.

Analysis results. The goal of the analysis team was to identify qualified search coordinates. During the search, it identified numerous targets, but only two were qualified: One was in ER-2 flyover imagery near Monterey, originally flagged by Mechanical Turk volunteers; the other was in Digital Globe imagery near the Farallon Islands, identified by a member of the more experienced image-processing team.[3] Though the low number might suggest our filtering of targets was overly aggressive, we have no reason to believe potential targets were missed. Our conclusion is simply that the ocean surface is not only very large but also very empty.

Once qualified, these two targets were then drift-modeled to identify coordinates for search boxes. For the first target, the drift models indicated it should have washed ashore in Monterey Bay. Because this was a region close to shore, it was relatively easy to send a private plane to the region, and we did. The second target was initially not far from the Farallon Islands, with both models predicting it would have drifted into a reasonably bounded search box within a modest distance from the initial location. Given our knowledge of Gray's intended course for the day, this was a very promising target, so we arranged a private offshore SAR flight. Though we did not find *Tenacious*, we did observe a few fishing vessels of *Tenacious*'s approximate size in the area. It is possible that the target we identified was one of these vessels. Though the goal of the search was not met, this particular identification provided some validation of the targeting process.

Discussion. The image-processing effort was the most structured and technical aspect of the volunteer search. In trying to cull its lessons, we highlight three rough topics: polytechtural design style, networked approaches to search, and civilian computer-vision research targeted at

disaster-response applications. For more on the organizational issues that arose in this more structured aspect of the search, see the longer version of this article.[3]

Polytechture. The software-development-and-deployment process that emerged was based on groups of experts working independently. Some of the more sophisticated software depended on preexisting expertise and components (such as parallelized image-processing pipelines and sophisticated drift-modeling software). In contrast, some software was ginned up for the occasion, building on now-standard Web tools like wikis, scripting languages, and public geocoding interfaces; it is encouraging to see how much was enabled through these lightweight tools.

Redundancy was an important theme in the process. Redundant ftp sites ensured availability; redundant drift modeling teams increased confidence in predictions; and redundant target qualification by experts provided both increased confidence and limits on "responsibility bias."

Perhaps the most interesting aspect of this loosely coupled software-development process was the variety of interfaces that emerged to stitch together the independent components: the cascaded Mechanical Turk interface for hierarchical expertise in image analysis; the ftp/email scheme for data transfer and staging; the Web-based "common operating picture" for geolocation and coarse-grain task tracking; the self-service "checkin/checkout" interface for expert image analysis; the decoupling of image file access from image browsing software; and the transactional workflow interface for drift modeling. Variations in these interfaces seemed to emerge from both the tasks at hand and the styles of the people involved.

The Web's evolution over the past decade enabled this polytechtural design. Perhaps most remarkable were the interactions between public data and global communication. The manufacturer's specifications for *Tenacious* were found on the Web, aerial images of *Tenacious* in its berth in San Francisco were found in publicly available sources, including Google Earth and Microsoft Virtual Earth, and a former owner of *Tenacious* discovered the *Tenacious* Search blog in the early days of the search and provided additional photos of *Tenacious* under sail. These details were helpful for parameterizing drift models and providing "template" pictures of what analysts should look for in their imagery. Despite its inefficiencies, the use of Mechanical Turk by volunteers to bootstrap the image-analysis process was remarkable, particularly in terms of having many people redundantly performing data analysis. Beyond the Turk pipeline, an interesting and important data-cleaning anecdote occurred while building the search template for *Tenacious*. Initially, one of Gray's relatives identified *Tenacious* in a Virtual Earth image by locating its slip in a San Francisco marina. In subsequent discussion, an analyst noticed that the boat in that image did not match *Tenacious*'s online specifications, and, following some reflection, the family member confirmed that Gray had swapped boat slips some years earlier and that the online image predated the swap. Few if any of these activities would have been possible 10 years before, not because of the march of technology per se but because of the enormous volume and variety of information now placed online and the growing subset of the population habituated to using it.

The volunteer search team's experience reinforces the need for technical advances in social computing.

Networked search. It is worthwhile reflecting on the relative efficacy of the component-based polytechtural design approach, compared to more traditional and deliberate strategies. The amateur effort was forced to rely on loosely coupled resources and management, operating asynchronously at a distance. In contrast, the Coast Guard operates in a much more prepared and tightly coupled manner, performing nearly all search steps at once, in real time; once a planning phase maps out the maximum radius a boat can travel, trained officers fly planes in carefully plotted flight patterns over the relevant area, using real-time imaging equipment and their naked eyes to search for targets. In contrast, a network-centric approach to SAR might offer certain advantages in scaling and evolution, since it does not rely on tightly integrated and relatively scarce human and equipment resources. This suggests a hybrid

methodology in which the relevant components of the search process are decoupled in a manner akin to our volunteer search, but more patiently architected, evolved, and integrated. For example, Coast Guard imagery experts need not be available to board search planes nationwide; instead, a remote image-analysis team could examine streaming (and archived) footage from multiple planes in different locales. Weather hazards and other issues suggest removing people boarding planes entirely; imagery could be acquired via satellites and unmanned aerial vehicles, which are constantly improving. Furthermore, a component-based approach takes advantage of the independent evolution of technologies and the ability to quickly train domain experts on each component. Image-analysis tools can improve separately from imaging equipment, which can evolve separately from devices flying the equipment. The networking of components and expertise is becoming relatively common in military settings and public-sector medical imaging. It would be useful to explore these ideas further for civilian settings like SAR, especially in light of their potential application to adjacent topics like disaster response.

Automated image analysis. The volunteer search team included experts in image processing in astronomy, as well as in computer vision. The consensus early on was that off-the-shelf image-recognition software wouldn't be accurate enough for the urgent task of identifying boats in satellite imagery of open ocean. During the course of the search a number of machine-vision experts examined the available data sets, concluding they were not of sufficient quality for automated processing, though it may have been because we lacked access to the "raw bits" obtained by satellite-based sensors. Though some experts attempted a simple form of automated screening by looking for clusters of adjacent pixels that stood out from the background, even these efforts were relatively unsuccessful.

It would be good to know if the problem of finding small boats in satellite imagery of the ocean is inherently difficult or simply requires more focused attention from computer-vision researchers. The problem of using remote imagery for SAR operations is a topic for which computer vision would seem to have a lot to offer, especially at sea, where obstructions are few.

Reflection

Having described the amateur SAR processes cobbled together to find *Tenacious*, we return to some of the issues we outlined initially when we met in Berkeley in 2008.

On the computational front, there are encouraging signs that SAR can be "democratized" to the point where a similar search could be conducted without extraordinary access to expertise and resources. The price of computer hardware has continued to shrink, and cloud services are commoditizing access to large computational clusters; it is now affordable to get quick access to enormous computing resources without social connections or up-front costs. In contrast, custom software pipelines for tasks like image processing, drift modeling, and command-and-control coordination are not widely available. This software vacuum is not an inherent problem but is an area where small teams of open-source developers and software researchers could have significant impact. The key barrier to SAR democratization may be access to data. Not clear is whether data providers (such as those in satellite imagery and in plane leasing) would be able to support large-scale, near-real-time feeds of public-safety-related imagery. Also not clear, from a policy perspective, is whether such a service is an agreed-upon social good. This topic deserves more public discussion and technical investigation. Sometimes the best way to democratize access to resources is to build disruptive low-fidelity prototypes; perhaps then this discussion can be accelerated through low-fidelity open-source prototypes that make the best of publicly available data (such as by aggregating multiple volunteer Webcams[3]).

The volunteer search team's experience reinforces the need for technical advances in social computing. In the end, the team exploited technology for many uses, not just the high-profile task of locating *Tenacious* in images from space. Modern networked technologies enabled a group of acquaintances and strangers to quickly self-organize, coordinate, build complex working systems, and attack problems in a data-driven manner. Still, the process of coordinating diverse volunteer skills in an emerging crisis was quite difficult, and there is significant room for improvement over standard email and blogging tools. A major challenge is to deliver solutions that exploit the software that people already use in their daily lives.

The efforts documented here are not the whole story of the search for *Tenacious* and its skipper; in addition to incredible work by the Coast Guard, there were other, quieter efforts among Gray's colleagues and family outside the public eye. Though we were frustrated achieving our primary goal, the work done in the volunteer effort was remarkable in many ways, and the tools and systems developed so quickly by an amateur team worked well in many cases. This was due in part to the incredible show of heart and hard work from the volunteers, for which many people will always be grateful. It is also due to the quickly maturing convergence of people, communication, computation, and sensing on the Internet. Jim Gray was a shrewd observer of technology trends, along with what they suggest about the next important steps in research. We hope the search for *Tenacious* sheds some light on those directions as well. C

References

1. Goldstein, J. and Rotich, J. *Digitally Networked Technology in Kenya's 2007–2008 Post-Election Crisis. Technical Report 2008–2009.* Berkman Center for Internet and Society at Harvard University, Cambridge, MA, Sept. 2008.
2. Heinzelman, J. and Waters, C. *Crowdsourcing Crisis Information in Disaster-Affected Haiti. Technical Report, Special Report 252.* United States Institute of Peace, Washington, D.C., Oct. 2010.
3. Hellerstein, J.M. and Tennenhouse, D.L. *Searching for Jim Gray: A Technical Overview, Technical Report UCB/EECS-2010-142.* EECS Department, University of California, Berkeley, Dec. 2010.
4. Saade, E. Search survey for S/V *Tenacious*: Gulf of Farallones and approaches to San Francisco Bay. *ACM SIGMOD Record 37*, 2 (June 2008), 70–77.
5. U.S. Coast Guard. *Search and Rescue Optimal Planning System (SAROPS) 2009*; http://www.uscg.mil/acquisition/international/sarops.asp

Joseph M. Hellerstein (hellerstein@berkeley.edu) is a professor in the EECS Computer Science Division of the University of California, Berkeley.

David L. Tennenhouse (dtennenhouse@nvpllc.com) is a partner in New Venture Partners, a venture-capital firm with offices in California, New Jersey, and the U.K., and former head of research at Intel.

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965745

A private overlay may ease concerns over surveillance tools supported by cellular networks.

BY STEPHEN B. WICKER

Cellular Telephony and the Question of Privacy

THE EVIL INCIDENT *to invasion of the privacy of the telephone is far greater than that involved in tampering with the mails. Whenever a telephone line is tapped, the privacy of the persons at both ends of the line is invaded, and all conversations between them upon any subject, and although proper, confidential, and privileged, may be overheard. Moreover, the tapping of one man's telephone line involves the tapping of the telephone of every other person whom he may call, or who may call him. As a means of espionage, writs of assistance and general warrants are but puny instruments of tyranny and oppression when compared with wiretapping.*

Justice Louis Brandeis, Dissenting Opinion
Olmstead v. United States, 277 U.S. 438 (1928)

Justice Brandeis wrote this warning when all telephones were wired and dedicated solely to speech communication. Since then we have witnessed the development of cellular technology and the convergence of a wide variety of functions onto the cellular platform. The combination of mobility and data services has led cellular technology to play an increasingly important role in economic and social networks, from forming the basis for new markets to facilitating political action across the globe. It is thus critical to recognize that cellular telephony is a surveillance technology that generates a vast store of personal information, information that has become a focus for law enforcement and marketing. The subsequent use of the collected data, both overt and covert, affects the use of cellular technology, as well as the individuals who use it and the society in which it has become ubiquitous.

In this article, I review how the courts have attempted to balance the needs of law enforcement and marketers against the privacy rights of individuals. The social science literature on the impact of surveillance on the individual and on society is surveyed and then applied to the specific case of cellular telephony. I conclude with a closer look at the mechanics of cellular data collection and a demonstra-

» key insights

- **The consolidation of all major forms of modern electronic communication onto the cellular platform and the ubiquity and power of the cellular platform have led to major changes in personal and social dynamics, political action, and economics. It is thus vitally important to recognize that cellular telephony is a surveillance technology.**
- **Professionals interested in the design and deployment of cellular technology will receive an overview of the current legal status of cellular databases, as well as the impact of the use of this data on the individual and society.**
- **A "private overlay" will allow cellular subscribers to enjoy the same user experience without providing private information.**



ILLUSTRATION BY ALEX WILLIAMSON

tion that a cellular network need not be a surveillance network; relatively simple public-key technology can be used to create a private overlay, allowing subscribers to make the most of cellular technology without the fear of creating a data record that can be exploited by others.

Telephony and the Bill of Rights

During the U.S.'s colonial period, British troops used *writs of assistance* as the basis for general searches for contraband in the homes of the colonists.[8] In an effort to prevent such searches in the new republic, the Fourth Amendment was included in the Bill of Rights. The Fourth Amendment protects against "unreasonable searches and seizures," and states that no warrant shall issue "but upon probable cause." The amendment's language says nothing, however, about telephones or electronic communication. The means by which legal protection against telephonic surveillance evolved through judicial interpretation of the Fourth Amendment is summarized here.

Content. The first significant Supreme Court case to address wiretapping was *Olmstead v. The United States* (1928). In a 5-4 decision, the Court determined that the police use of a wiretap was not search and seizure. Writing for the majority, Chief Justice Taft expressed an extremely literal interpretation of "search and seizure":

> *The [Fourth] Amendment does not forbid what was done here. There was no searching. There was no seizure. The evidence was secured by the use of the sense of hearing and that only. There was no entry of the houses or offices of the defendants.*
>
> Chief Justice William Howard Taft
> *Olmstead v. United States*,
> 277 U.S. 438 (1928)

The first of the two holdings of the *Olmstead* decision—the interception of a conversation is not seizure—was reversed in *Berger v. New York* (1967). Acting under a New York law of the time, police planted listening devices in the office of an attorney named Ralph Berger. Berger was subsequently indicted, tried, and convicted for conspiracy to bribe a public official. In its opinion, the Supreme Court focused on the extremely broad authority granted by the statute: Law enforcement authorities were only required to identify the individual and the phone number to be tapped in order to obtain authorization for a wiretap. Likening this type of warrant to the general warrants used by the British in the American colonies, the Court overturned the New York statute. In doing so, the Court held that conversations were indeed protected by the Fourth Amendment, and that the intercep-

tion of a conversation was a seizure.

The second of the *Olmstead* holdings—where there is no physical trespass, there can be no search—fell that same year. In *Katz v. United States* (1967), the Court considered the case of Charles Katz, who had used a pay phone in Los Angeles to place illegal bets in Miami and Boston. Without obtaining a warrant, FBI agents placed listening devices outside of the phone booth and recorded Katz' end of several conversations. The transcripts of these conversations were introduced during Katz' trial, and presumably played a role in his conviction. In response to his appeal, the Supreme Court ruled that tapping phone calls placed from a phone booth required a warrant. The majority opinion explicitly overturned *Olmstead*, holding that the Fourth Amendment "protects people, not places;" trespass was no longer necessary for the Fourth Amendment to be implicated.

Justice Harlan's concurring opinion introduced a two-part test for determining whether the Fourth Amendment should be applied in a given situation:

- The person must have exhibited "an actual (subjective) expectation of privacy;"
- This expectation is one that "society is prepared to recognize as reasonable."

Thus by 1967 *Olmstead* was completely reversed, and the Court was applying Fourth Amendment protection to the content of telephone calls. However, the *context* of telephone and other electronic communication did not receive the same level of protection.

Context. The distinction between the content and context of electronic communication is best understood through the analogy of postal mail. The content information is the letter itself—the written or typed communication generated by one party for the purpose of communicating with another party. As with the content of a telephone call, letters are protected by a series of rather strict regulations.[a] The context information consists of the information on the outside of the envelope, information used by the communication system to establish communication between the two parties. In the case of the postal system, this consists primarily of the mailing and return addresses, but may also include postmarks or other information that accumulates in transit. In the case of a cellular telephone call, context data includes the number the caller dials, the number from which the caller dials, the location of the caller, the time of the call, and its duration.

The surveillance architecture adopted for cellular networks generates a pool of data that feeds into law enforcement's and marketers' desire for personal information.

Courts and legislatures have been far less protective of context information than content. The basic rationale is that the user understands context information is needed to complete the communication process, and that in using the technology, context information is freely given to the network. It follows that, according to the courts, there is no reasonable expectation of privacy in this information, and the Fourth Amendment is not implicated.

The key precedent is *United States v. Miller* (1976), a case with far reaching implications for the public use of a wide variety of communication networks. The case involved a modern-day bootlegger named Mitch Miller; prohibition was not the issue, the focus was instead on the more mundane matter of taxation. While putting out a fire at Miller's warehouse, firefighters and police discovered 175 gallons of whiskey that did not have the requisite tax stamps. Investigators obtained, without a warrant, copies of Miller's deposit slips and checks. The cancelled checks showed that Miller had purchased material for the construction of a still. Miller was subsequently convicted of possessing an unregistered still.

Miller appealed, claiming that his Fourth Amendment rights had been violated; the investigators should have obtained a warrant before acquiring his bank records. The Supreme Court disagreed. Writing for the Court, Justice Powell stated that:

There is no legitimate "expectation of privacy" in the contents of the original checks and deposit slips, since the checks are not confidential communications, but negotiable instruments to be used in commercial transactions, and ***all the documents obtained contain only in-***

a See Ex Parte Jackson, 96 U.S. (6 Otto) 727, 733 (1877); *Walter v. United States*, 447 U.S. 649, 651 (1980).

formation voluntarily conveyed to the banks and exposed to their employees in the ordinary course of business (emphasis added).

Justice Lewis Powell
United States v. Miller,
425 U.S. 435 (1976)

The *Miller* ruling was applied to electronic communication a few years later in the case of *Smith v. Maryland* (1979). In this case, Michael Lee Smith burglarized a woman's home and then made harassing telephone calls to her after the fact. In response to a request from investigators, the telephone company installed a *pen register* at the central office that served Smith's home telephone line. A pen register is a device that records all of the numbers dialed from a given telephone line. In this particular case, the pen register captured the victim's phone number being dialed on Smith's telephone line; as a result, a warrant for a search of Smith's home was obtained, evidence was found, and Smith was subsequently convicted of robbery. Smith appealed, claiming that the use of the pen register violated his Fourth Amendment rights. The Supreme Court disagreed. On the basis of the *Katz* reasonable expectation test and the results of the *Miller* case, Justice Blackmun wrote that:

First, it is doubtful that telephone users in general have any expectation of privacy regarding the numbers they dial, ***since they typically know that they must convey phone numbers to the telephone company and that the company has facilities for recording this information*** *and does in fact record it for various legitimate business purposes* (emphasis added).

Justice Harry Blackmun
Smith v. Maryland,
442 U.S. 735 (1979)

By 1979, the Court had clearly distinguished privacy rights regarding the content of telephone calls from the rights accorded to their context. This distinction was embedded in the Electronic Communication Privacy Act of 1986 (ECPA[12]), which includes three titles that provide varying levels of protection for various types of electronic communication:

- Title I: Electronic Communications in Transit;
- Title II: Stored Electronic Communication; and
- Title III: Pen Register/Trap and Trace Devices.[b]



ILLUSTRATION BY ALEX WILLIAMSON

Title I covers the content of electronic communication, and generally requires a warrant for the disclosure of the content. Title II, sometimes referred to as the Stored Communications Act (SCA), covers stored wire and electronic communications, as well as transactional records. Title III, sometimes referred to as the Pen Register Act, covers pen registers and related devices.

There has been a great deal of court time spent debating which of the three titles applies to the information collected by a cellular network. This is an important issue, as it determines the legal burdens that law enforcement must overcome to obtain the data. Title II has been found to cover *historical* cell site data.[c] Historical cell site data is a list of the cell sites visited by a subscriber up until the point in time that the request by law enforcement is made. According to Title II, law enforcement agencies can obtain this information by providing "specific and articulable facts" showing that the information is "relevant and material to an ongoing investigation," a procedural hurdle that is substantially lower than the "probable cause" requirement for a warrant.[d]

Prospective or real-time cell site data is forward looking. A request for prospective data is a request that the service provider provide a continuous update of the cell sites with which the subscriber has made contact. The legal status of prospective data depends in part on whether or not a cellular telephone is considered a tracking device.[e] Several courts[f] have ruled that a cellphone is not a tracking device and that Title III of the ECPA is the ruling authority. In these cases the registration messages emitted have been likened to the numbers dialed by the user. The legal protection under Title III is minimal, requiring only that an attorney for the government certify that the information to be obtained is relevant to an ongoing criminal investigation.[12]

Other courts,[g] however, have come to the opposite conclusion. In 2005 Judge Orenstein of the Eastern District of New York denied a law enforcement request for prospective cell site data.

b A trap and trace device is similar to a pen register, but instead of capturing numbers dialed from a given number, it captures the numbers of parties that dial to a given number.

c See In re Applications, 509 F. Supp. 2d 76 (D. Mass. 2007); In re Application, 2007 WL 3036849 (S.D. Tex. Oct. 17, 2007).

d The details of the requirements for a warrant can be found in Rule 41 of the Federal Rules of Criminal Procedure.

e See In re Application for Pen Register and Trap/Trace Device with Cell Site Location Authority, H-05-557M S.D. Tex., Oct. 14, 2005: [a] Rule 41 probable cause warrant was (and is) the standard procedure for authorizing the installation and use of mobile tracking devices. See *United States v. Karo*, (1984).

f See, for example, In re Application for an Order Authorizing the Extension and Use of a Pen Register Device, 2007 WL 397129 (E.D. Cal. Feb. 1, 2007); In re Application of the United States, 411 F. Supp. 2d 678 (W.D. La. 2006); In re Application of the United States for an Order for Prospective Cell Site Location Info., 460 F. Supp. 2d 448 (S.D.N.Y. 2006) (S.D.N.Y. II); In re Application of the United States of America, 433 F.Supp.2d 804 (S.D. Tex. 2006)

g See, for example, re Application of United States of America for an Order Authorizing the Disclosure of Prospective Cell Site Info., 2006 WL 2871743 (E.D. Wis. Oct. 6, 2006); In re Application of the United States of America, 441 F. Supp. 2d 816 (S.D. Tex. 2006); In re Application for an Order Authorizing the Installation and Use of a Pen Register and Directing the Disclosure of Telecomm. Records, 439 F. Supp. 2d 456 (D. Md. 2006).

Judge Orenstein found[h] that a cellphone was in fact a tracking device, and that a showing of probable cause was necessary to obtain prospective cell site data. On Sept. 7, 2010 the United States Court of Appeals for the Third Circuit upheld a lower court's opinion that a cellular telephone was in fact a tracking device, and further ruled that it is within a magistrate judge's discretion to require a showing of probable cause before granting a request for historical cell site data.[i]

CALEA and the USA PATRIOT Act. Clearly the information made available by the cellular architecture has motivated law enforcement to pursue it. And having gotten used to this massive source of personal information, law enforcement would like to keep the data conduits open. The development and commercialization of new telephone technologies in the 1980s and 1990s caused concern that less surveillance-friendly architectures were becoming the norm. This prompted law enforcement to ask Congress for legislation that would require service providers to provide a common means for surveillance regardless of the technology in use. The Director of the FBI made the point quite clearly in testimony before Congress:

The purpose of this legislation, quite simply, is to maintain technological capabilities commensurate with existing statutory authority; that is, to prevent advanced telecommunications technology from repealing, de facto, statutory authority now existing and conferred to us by the Congress.

Former FBI Director Louis Freeh[18]

The result of this effort—the Communications Assistance for Law Enforcement Act (CALEA[4])—was passed on the last night of the 1994 congressional session. CALEA requires that service providers "facilitat[e] authorized communications interceptions and access to call-identifying information unobtrusively and with a minimum of interference with any subscriber's telecommunications service."[j]

Perhaps the most significant impact of CALEA on cellular systems will be through its amended provisions affecting voice-over-IP (VoIP). Under CALEA, VoIP service providers cannot release IP calls to travel freely between subscriber terminal adapters; instead, the service provider must anchor most calls, creating a fixed point that must be traversed by call packets in both directions.[k] Upon the presentation of an appropriate warrant, a duplicate call stream is generated at this fixed point and passed to a law enforcement agency. Such restrictions will almost certainly apply to 4G cellular platforms, which will implement all-IP solutions for voice and data.[l]

Several of the provisions of the USA PATRIOT Act[m] also have current and future implications for cellular systems. The PATRIOT Act amended much of the legislation discussed earlier,[n] the following provides a brief summary of a few key elements.

- Section 204 amended Title II of the ECPA so that stored voicemail can be obtained by the government through a search warrant rather than through the more stringent process of obtaining a wiretap order.[o]
- Section 216 expanded the pen register and trap and trace provisions of the ECPA to explicitly cover the context of Internet traffic. The URLs visited from a cellular platform, for example, thus receive the low level of protection provided by Title III of the ECPA.
- Section 217 permits government interception of the "communications of a computer trespasser" if the owner or operator of a "protected computer" authorizes the interception.

The last of the provisions, commonly referred to as the "computer trespasser" provision, has caused concern as it appears to allow interception of all traffic through intermediate routers and switches if the owners of the equipment authorize the interception. This could, for example, include all traffic through a gateway GPRS support node—the interface between 3G cellular networks and the Internet. Given that the service providers have been granted immunity from lawsuits filed in response to their cooperation with intelligence agencies,[27] this provision was particularly troubling to some privacy advocates.[p]

It should be noted that some researchers have argued that the PATRIOT Act has simply clarified existing policy. Orin Kerr, for example, has provided a detailed argument that "none of the changes altered the basic statutory structure of the Electronic Communications Privacy Act of 1986."[26]

The Right to Market. Thus far, I have focused on the laws and regulations that limit law enforcement's access to the data collected by cellular service providers. But what of the service providers themselves? A quick tour through some recent case law is interesting in that it shows how the carriers view their right to use this information, and the commercial value that they place on it. In what follows there will be two basic questions: Are the carriers limited in how they may use the data for their own marketing? Are they limited in their ability to sell the data to third parties?

On January 3, 1996 Congress passed the Telecommunications Act of 1996, the first major restructuring of telecom law since 1934. Section 222 of the Act states that "[e]very telecommunications carrier has a duty to protect the confidentiality of proprietary information of, and relating

h 384 F. Supp.2d 562 (E.D.N.Y. 2005)

i See *The Matter Of The Application Of The United States Of America For An Order Directing A Provider Of Electronic Communication Service To Disclose Records To The Government*, 3d. Cir., 08-4227.

j 47 U.S.C. Section 1002(a)

k The fixed point often takes the form of a Session Border Controller (SBC). See, for example, The Benefits of Router-Integrated Session Border Control, White paper, Juniper Networks, http://www.juniper.net/us/en/local/pdf/whitepapers/2000311-en.pdf and http://tools.ietf.org/html/draft-ietf-sipping-sbc-funcs-00.

l For a discussion of potential vulnerabilities of CALEA monitoring systems, see Pfitzmann et al.[35] and Sherr et al.[41]

m Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism Act of 2001, signed into law Oct. 26, 2001.

n A detailed discussion can be found at http://epic.org/privacy/terrorism/usapatriot/#history. Many of the provisions discussed here had associated sunset clauses, but as recently as Mar. 1, 2010, Congress has continued to provide extensions to these clauses.

o For a comparison of the two procedures, see, for example, Susan Friewald:[19] "Because of the particular dangers of abusing electronic surveillance, the Court required that agents who wanted to conduct it had to surmount several procedural hurdles significantly more demanding than the probable cause warrant needed to search a home."

p See, for example, http://epic.org/privacy/terrorism/usapatriot/.

to, other telecommunication carriers, equipment manufacturers, and customers."[44] With regard to customers, section 222 defined "customer proprietary network information" (CPNI) to be "information that relates to the quantity, technical configuration, type, destination, location, and amount of use of a telecommunications service subscribed to by any customer of a telecommunications carrier, and that is made available to the carrier by the customer solely by virtue of the carrier-customer relationship." Note that Congress was somewhat prescient in its inclusion of "location."

In the 1998 order passed by the FCC to implement section 222, the FCC imposed an "opt-in" requirement on any carrier that wanted to use a customer's data to market additional services to that customer. The carriers had to obtain a customer's affirmative, explicit consent before using or sharing that customer's information outside of the existing relationship with the carrier.[14] The carriers sued the FCC in the 10th Circuit Court of Appeals (*U.S. West, Inc. v. FCC*), claiming that the opt-in rule violated their First and Fifth Amendment rights. With regard to the First Amendment, the carriers argued that the FCC's rules were an unconstitutional restriction on the carriers' "rights to speak with their customers." The carriers' Fifth Amendment argument relied on the Takings Clause; the last phrase in the Fifth Amendment, the Takings Clause states that "private property [shall not] be taken for public use, without just compensation." The carriers argued that "CPNI represents valuable property that belongs to the carriers and the regulations greatly diminish its value."[47]

In a 2-1 decision, the Circuit Court agreed with the carriers' First Amendment argument. While acknowledging that the speech involved was commercial and that such speech receives less protection than, for example, political speech, the Court held the FCC's rule was "more extensive than is necessary to serve the government's interest." Writing for the Court, Judge Tacha stated that "Even assuming that telecommunications customers value the privacy of CPNI, the FCC record does not adequately show that an opt-out strategy would not sufficiently protect customer privacy."

In dynamic political situations, many users will be aware of the potential for surveillance, and will thus put self-imposed limitations on their use of cellular technology.

Judge Tacha did not address the Fifth Amendment argument, but Judge Briscoe, writing in dissent, made his opinion clear, stating that "I view U.S. West's petition for review as little more than a run-of-the-mill attack on an agency order 'clothed by ingenious argument in the garb' of First and Fifth Amendment issues."

In response to the Tenth Circuit's decision, the FCC modified its rules in 2002, allowing for an opt-out rule for sharing of customer information between a carrier and its affiliates for marketing purposes.[15] The 2002 rule also addressed the sharing of information with "independent contractors" for marketing communications-related services. An opt-out rule was deemed acceptable here as well, but recognizing the additional privacy risk, the FCC required that the carriers establish confidentiality agreements with the contractors to further protect consumer privacy.

In 2005, the Electronic Privacy Information Center (EPIC) requested that these third-party rules be modified. Pointing to the use of "pretexting"—a practice in which third parties pretend to have the authority to receive the data and then use it for their own marketing, tracking, or other purposes—EPIC called for stricter rules that would protect the safety of the subscriber.[q] In 2007, the FCC passed yet another set of rules, this time requiring that the carriers "obtain opt-in consent from a customer before disclosing that customer's [information] to a carrier's joint venture partner or independent contractor for the purpose of marketing communications-related services to that customer."[16]

The carriers sued, once again asserting their First Amendment rights. In *National Cable & Telecommunication Assoc. v. F.C.C.* (2009), the U.S. Court of Appeals for the District of Columbia Circuit conducted a meticulous analysis in which the judges considered whether the government had met its constitutional burden in regulating what all agreed was commercial speech. In the end, the Court upheld

q In 2006 Congress passed the Telephone Records and Privacy Protection Act of 2006, making pretexting illegal.

the FCC's rules, asserting that they were "proportionate to the interests sought to be advanced."

Which brings us up to date: an opt-out rule governs the carriers' use of CPNI in their own marketing, while an opt-in rule covers the transfer of this data to third parties for their own marketing purposes.

Concluding thoughts on the law. In summary, the surveillance architecture adopted for cellular networks generates a pool of data that feeds into law enforcement's and marketers' desire for personal information. The result has been a long-running legal battle in which the privacy rights of individuals are continuously traded off against legal and economic imperatives.

The Impact of Cellular Surveillance

The social science literature on surveillance and privacy covers a great deal of ground, so I will begin with a few basic assumptions that will narrow the field a bit. We first assume that the primary impact of surveillance is a reduction in privacy. The next step—a definition for privacy—has proven in the past to be a notoriously difficult problem. Attempts at definitions are usually followed by a flurry of articles pointing out why the definition doesn't work in one or more contexts.[r] An all-encompassing definition is not necessary for our purposes, however, as we are focusing on the impact of surveillance on the use of the cellular platform. We need only note that a common element of most privacy theories is the metaphor of a zone of seclusion, a zone in which the agent can control access to various types of personal information.[33] The value of such a zone lies in part in the agent's perception of solitude and safety. The agent feels free to exercise various thoughts and behaviors without threat of censure, and is thus able to develop a sense of self-realization. Self-realization is a core personal and social value—it has been cited as the basis for valuing free speech,[37] thus enmeshing privacy in a web of values that animate democratic systems of government. Privacy is thus connected to personal as well as societal development and well-being.

An overlapping yet distinct issue related to the cellular platform is the potential for manipulation through the use of personal information. As we will see, the availability of personal information increases the efficacy of advertising and other attempts to drive the agent to particular thoughts or actions. The agent's autonomy is thus at risk, implicating another of the values important to democratic government.[6,11]

From the standpoint of the cellular platform, then, there are two issues to be addressed: the relatively passive infringement on the zone of seclusion through eavesdropping and data collection, and the more active infringement through manipulation based on collected data. The passive infringers generally consist of service providers and law enforcement agencies, while the more active take the form of marketers, a group including service providers as well as third parties that have purchased the collected data.

Passive surveillance. Passive privacy infringement has its impact through the cellular user community's awareness of the potential for surveillance. The omnipresent potential for surveillance affects several aspects of the use of the cellular platform, including social networking, family interaction, and political expression. We will consider the latter as an exemplary case, but it should be borne in mind that this is but one dimension of a multidimensional problem.

The cellular platform has become increasingly important as a means for conveying political speech and organizing political behavior. The copiers and FAX machines that enabled the movements that brought down the Soviet empire[s] have been replaced by the cellphone and its immediately available, highly portable texting and video capabilities. Some of the more salient examples of the political use of the cellular platform have involved the coordination of mass action against political corruption, such as the 2001 protest against Philippine President Joseph Estrada and the Ukranian "Orange Revolution" of 2004.

A Kenyan example typifies both the use of the platform as a political tool and the potential consequences of surveillance. In January 2008, it was reported that incumbent presidential candidate Mwai Kibaki had rigged the Kenyan presidential election. A texting campaign to promote demonstrations began almost immediately, with the discourse quickly devolving into racial hatred.[21] Instead of shutting down the SMS system, the Kenyan authorities sent messages of peace and calm to the nine million Safaricom subscribers. After the violence subsided, cellular service providers gave the Kenyan government a list of some 1,700 individuals who had allegedly used texting to promote mob violence.[36] The Kenyan Parliament is debating a law that places limits on the contents of text messages.

Cellular networks have thus become a key platform for political speech. The impact of surveillance on such use can be developed through analogy to Jeremy Bentham's Panopticon.[2] The Panopticon was a proposed prison in which the cells were arranged radially about a central tower. The cells were backlit so that a guard in the tower could always see the prisoners, but the prisoners could never see the guards. Bentham characterized the Panopticon as providing a "new mode of obtaining power of mind over mind, in a quantity hitherto without example."

The analogy is obvious—we know that wiretapping or location data collection through use of the cellular platform is possible, we just do not know whether or when it is happening. It follows that in dynamic political situations, many users will be aware of the potential for surveillance, and will thus put self-imposed limitations on their use of cellular technology. Cellular networks are thus a distributed form of Panopticon.[45]

The self-imposition of discipline is a key element in this analysis. In *Discipline and Punish*, Michel Foucault characterized the impact of the Panopticon's pervasive and undetectable surveillance as assuring "the automatic functioning of power."[17] Foucault argued that this led to an internalization of discipline that resulted in "docile

r A sense of the back and forth can be obtained by starting at the beginning of Schoeman's excellent anthology[38] and reading straight through.

s See, for example, Endre Dányi's Xerox Project: Photocopy Machines as a Metaphor for an 'Open Society.' *The Information Society 22*, 2 (Apr. 2006), 111–115.

bodies," bodies that were ideal for the regimented classrooms, factories, and military of the modern state. Docility can take many forms: Dawn Schrader, for example, has noted the impact of surveillance/observation on knowledge acquisition patterns; the individual under surveillance is intellectually docile, less likely to experiment or to engage in what she calls "epistemic stretch."[39] Surveillance can literally make us dumber over time. The impact of the perception of surveillance on cellular users is thus to limit experimentation by the users, who subsequently channel speech into "safe" and innocuous pathways. It follows that given the growing importance of the cellular platform as a means for political speech, the surveillance capabilities inherent in the design of cellular networks are a problem with deep political ramifications.

Active surveillance creates another, overlapping, set of problems for the individual and society. The first lies in the use of the data to sort individuals into categories that may limit their options in various ways. In the second, the information flows themselves are manipulative. We begin with the problem of sorting, and then move on to the latter form of manipulation.

In *The Panoptic Sort*, Oscar Gandy investigated the means by which panoptic data is used to classify and sort individuals.[20] Law enforcement, for example, uses data to "profile" and thereby sort people into those who are suspicious and those who appear relatively harmless. Credit agencies use personal data to perform a finer sort, allocating individuals into varying levels of credit worthiness. Direct marketers use a similar approach to determine who is most likely to buy a given range of products. Gandy notes that the latter creates an insidious form of discrimination, as individuals are relegated to different information streams based on the likelihood they will buy a given item or service, and individual perspectives and life opportunities are correspondingly limited.

In the cellular context, such sorting is performed by both the service providers and third-party marketers. As we have seen, exemplars from both groups have fought against FCC restrictions on the use of CPNI for selective marketing of communication and other services.



There is an extensive literature on how individual information flows can be manipulative. For example, in his "Postscript on the Societies of Control," Gilles Deleuze introduces the concept of "modulation" as an adaptive control mechanism in which an information stream from the individual is used to fine-tune the information provided to the individual, driving the individual to the desired state of behavior or belief.[9]

The general idea here is that information about an individual is used to frame a decision problem in such a manner that the individual is guided to make the choice desired by the framer. This has become an important concept in economics and game theory; Tversky and Kahneman, for example, have shown that the rational actor's perception of a decision problem is substantially dependent on the how the problem is presented—what Tversky and Kahneman refer to as the "framing" of the problem.[46] Framing is so important to decision making that individuals have been shown to come to differing conclusions depending on how the relevant information has been presented.

Framing plays an important role in advertising. In *Decoding Advertisements*,[48] Williamson uses the psychoanalytic methodologies of Lacan and Althusser to describe how targeted advertisements invite the individual into a conceptual framework, creating a sense of identity in which the individual will naturally buy the proffered product or service. Personal information is used in this process to fine-tune the frame, enhancing the sense in which the advertisement "names" the individual reader or viewer and thus draws the consumer in and drives him or her to the desired behavior.

The ability of the marketer to fine-tune efforts is greatly enhanced when the customer's response to advertising can be directly observed, as is the case with the cellular platform. This is made possible through real-time interactive technologies that are embedded in cellphones, such as Web browsers with Internet connectivity. A simple example (an example to which the author is highly susceptible) involves an email message describing a newly released book that is available at a notable Web retailer. The advertiser will know when the email went out, when the link was followed to the Web site, and whether or not a purchase was made. Cell-based social networking applications such as Foursquare and Loopt take the process a step further by using subscriber location information as the basis for delivering location-based advertising. For example, a user may be informed that she is close to a restaurant that happens to serve her favorite food. She may even be offered a discount, further adding to the attraction. The efficacy of the advertising can then be measured by determining whether the user actually enters the restaurant.[28]

The problematic nature of such examples is not always clear, as some would argue that they are pleased to receive the advertisements and to be informed, for example, of the availability of their favorite food. So what is the problem? Primarily, it lies in transparency—the user may not understand the nature of location data collection, or the process that led to one restaurant or service being proffered instead of another. There has been a pre-selection process that has taken place outside of the cellu-

ILLUSTRATION BY ALEX WILLIAMSON

lar user's field of vision and cognizance. The opportunity to explore and learn on one's own has been correspondingly limited and channeled, affecting both self-realization and autonomy.[11] The "tightness" of this Deleuzean feedback loop—its bandwidth and precision—is particularly troubling.

Cellular Architecture, Cellular Databases

What it is about the cellular network that makes it so surveillance friendly, and a potential threat to the individual user and to society? The answer lies in a series of design choices, choices made in an attempt to solve the problem of establishing and maintaining contact with a mobile user. The details have filled many books (see, for example, Etemad,[13] Holma and Toskala,[22] Kaarenenetal et al.,[24] and Mouly and Pautet.[30]), but we need only trace the path of a call that is incoming to a cellular user to see how personal data is being collected and put to use.

The coverage area of a cellular network is partitioned into relatively small areas called cells, with each cell receiving a subset of the radio resources of the overall network. Two cells may be assigned identical spectral resources—a process called frequency reuse—if the cells are far enough apart to prevent their radio transmissions from interfering with each other. A cell tower sits at the center of each cell, establishing connections between mobile users and the wired cellular infrastructure. Location areas are defined to consist of one or a small number of cells. As we will see, the location area is the finest level of granularity used by the network in trying to complete a call to a cellular platform.

We now consider an incoming call. To complete an incoming call to a cellular phone, the network routes the call to a mobile switching center (MSC[t]) that is near the phone. Through a process called paging, the MSC then causes the called cellular phone to ring. When the cellular user answers his or her phone, the MSC completes the call and communication can commence.

t As space is limited and such details are not important to the theme of this article, I will not attempt to track vocabulary distinctions between second-, third-, and fourth-generation cellular systems.

It remains possible, however, to secure cellular networks against surveillance.

In order to perform this routing and paging process, the network must keep track of the location of the cellular telephone. This is done through the registration process. All cellular telephones that are powered on periodically transmit registration messages that are received by one or more nearby cell towers and then processed by the network. The resulting location information thus acquired is stored with varying levels of granularity in several databases. The databases of interest to us here are the Home Location Register (HLR) and the Visitor Location Register (VLR). The HLR is a centralized database that contains a variety of subscriber information, including a relatively coarse estimate of the subscriber's current location. HLRs are generally quite large; there need be only one per cellular network. VLRs, generally associated with local switches, contain local registration data, including the identity of the cell site through which registration messages are received. There is typically one VLR per mobile switching center (MSC) or equivalent.

The VLR stores the identification number for the cell site through which the registration message was received. The identity of the MSC associated with the VLR is forwarded to the Home Location Register (HLR) that maintains the records for the registering platform.

We can now track the progress of an incoming call in more detail. Calls from outside the cellular network will generally enter the network through a gateway MSC. The gateway MSC will use the called number to identify and query the appropriate HLR to determine how to route the call. The call is then forwarded to the MSC associated with the last registration message, which in turn queries the VLR to determine in which location area to attempt to contact the subscriber. The base station controller associated with the location area then causes a paging message to be sent to the called cellular telephone, causing it to ring. If the subscriber answers the call, the MSC connects a pair of voice channels (to and from the cellular platform), and completes call setup.

The HLR and VLRs (or equivalents) are thus the sources of the historic and prospective cell site data discussed earlier in the survey of telephone privacy law.

The question of whether a cellular telephone is a tracking device has often hinged on the resolution of the cell site data. If the data consists solely of the cell site ID, then the precision of the location information is clearly a function of the size of the cell. Cell sizes vary significantly, but the following can be used as a rough rule of thumb:[u]

Urban:	1 mile radius
Suburban:	2 mile radius
Rural:	>4 mile radius

It follows that through registration messages alone, a subscriber's location is recorded to the level of a metropolitan area at a minimum, and sometimes to the level of a neighborhood.

So far I have focused on voice calls. With regard to data "calls," it should be noted that 3G cellular separates the core network into circuit-switched and packet-switched domains, while 4G is purely packet-switched. Data calls are set up in packet-switched domains through the support of a serving and a gateway General Packet Radio Service (GPRS) support node. The HLR and VLR play registration, roaming, and mobility management roles for data calls that are similar to those provided in voice calls, so I will not go into further details here except to note that location data is accumulated in a similar manner.

In summary, the functionality of a cellular network is based on the network's ability to track the cellular subscriber. It was designed to collect and store location information, inadvertently creating an attractive information source for law enforcement and marketing professionals, as described previously. Next, we will see this need not be the case.

A Private Overlay

So long as the cellular concept requires that a piece of equipment be located within a particular cell, there will be a requirement in cellular systems that an MSC be able to locate user equipment at the level of one or a small number of cell sites. It is important to note, however, that it is the equipment that needs to be located and not a specific, named subscriber. In this section we will consider the possibility of creating a private overlay for cellular systems that protects user privacy by strictly separating equipment identity from user identity. The proposed overlay requires the addition of a Public Key Infrastructure (PKI).[10] The PKI provides the network and all subscribers with a public encryption key and a private decryption key. With this addition, a private overlay to the existing cellular infrastructure can be established as described below.

The scenario assumed here is that of a cellular telephone with standard capabilities to which has been added the ability to operate in a private mode, a private mode in which the service provider is unable to associate location data for the phone with a specific user. The private mode is predicated on a private registration process, which is enabled by having the network transmit once a day (or at some suitable interval) an identical certification message to each authorized subscriber. The certification message that is sent to each subscriber is encrypted using that subscriber's public encryption key.

When the user enables the private cellular mode, the cellular platform sends a *Privacy Enabling Registration* (PER) message to the network. The PER, consisting of the certification message and a *Random Equipment Tag* (RET), is encrypted using the network's public encryption key. The certification message acts as a zero-knowledge proof, showing the network that the PER was sent by a valid user, but without actually identifying the user (we will address the problem of cloning in a moment). The RET is a random number that will be entered into the VLR and the HLR and treated as if it were a phone number. The VLR and the HLR will thus collect all of the information needed to establish and maintain phone calls to the cellular platform, but will not associate this information with a particular individual or phone number. So long as the user chooses to remain in private cellular mode, subsequent registration messages will include the RET as opposed to the user's telephone number.

Call setup, mobility management, and roaming will all be handled exactly as before, with the difference that the HLR and VLR location information is associated with the RET, as opposed to a phone number. Data calls can be kept private by associating the RET with a temporary IP address.[v]

Incoming calls require that calling parties know the RET. In order for the RET to be associated with the correct HLR, it will also be necessary that the calling party identify the service provider that serves the called party. The user in private cellular mode must thus distribute, using public key encryption, his or her RET and the identity of the service provider to those parties from whom he or she would be willing to receive a call.

Calls can be placed from the cellular platform in private mode using the private context developed for incoming calls, or it may prove desirable to register outgoing calls on a call-by-call basis using distinct random strings. This would reduce the amount of information associated with a single random string, thus reducing the ability of the service provider to associate the private context with a specific user.

We now must confront the problems of cloning and billing. Both can be addressed by building a Trusted Platform Module (TPM)[1] into the cellular platform. The TPM (or an equivalent device) can be programmed to keep the certification message in a cryptographically secure vault, and thus unavailable to anyone wishing to transfer it to another platform. When the network receives a PER message, it can thus be assured that the transmitting phone actually received the certification message from the network. Remote attestation can be used to ensure that the software controlling the TPM has not been altered.

The problem of billing has to be clearly addressed, for the service provider faces the uncomfortable task of providing service to an unknown party. The solution lies, once again, in

u Jeff Pool, Innopath, private correspondence. These areas are further reduced if the cell has multiple sectors.

v One version of the GPRS standard allowed for an anonymous Packet Data Protocol (PDP) context. This context associated a PDP address at the SGSN with a temporary logical link identifier—the IMSI was not associated with the PDP address, and the context was thus anonymous. The details were described in early versions of section 9.2.2.3 of ETSI GSM 03.60, but were later removed from the standard.

the TPM. The number of private call minutes available to the platform can be controlled through software in the platform, with the software certified by remote attestation. If need be, the private call minutes can be prepaid.

The potential for considering the private mode as a prepaid service may have a significant advantage with respect to CALEA, as CALEA does not currently cover prepaid cellular telephones. In the U.S. and many other countries, one may buy and use a prepaid cellular telephone without associating one's name with the phone.[w] The proposed privacy overlay would thus provide postpaid cellular telephone users with the privacy benefits of prepaid cellular.[x]

Other problems remain to be addressed, of course. For example, Cortes, Pregibon, and Volinsky have shown that it is possible to identify fraudulent users of a cellular system by using call data to construct dynamic graphs, and then performing a comparative analysis of subgraphs that form "communities of interest."[7] A similar comparative analysis can be used for deanonymizing users of the proposed system unless the random tag is changed fairly frequently.

Conclusion

We have seen that cellular telephony is a surveillance technology. Cellular networks were designed, however unintentionally, to collect personal data, thus creating an extremely attractive source of information for law enforcement agencies and marketers. The impact of this surveillance on the users and uses of the cellular platform is becoming increasingly important as the platform plays a prominent role in social, economic, and political contexts. It remains possible, however, to secure cellular networks against surveillance. The private cellular overlay proposed here would serve this purpose while potentially putting the subscriber in control of his or her personal information. Legal issues remain and legislation may be necessary before a private cellular system can be made available to the public, but a public discussion as to whether we want a technology as important as cellular to be open to covert surveillance would be a good and highly democratic idea.

Acknowledgments

This work was funded in part by the National Science Foundation TRUST Science and Technology Center and the NSF Trustworthy Computing Program. The author gratefully acknowledges the technical and editorial assistance of Sarah Hale, Lee Humphries, and Jeff Pool. He also extends thanks to the anonymous reviewers for their extensive and insightful comments.

w According to the UPI, many of the cell phones used to coordinate action in the Philippine uprisings against former President Estrada were unregistered, prepaid phones. See http://www.upiasia.com/Politics/2008/01/21/texting_as_an_activist_tool/6075/.

x On May 26, 2010, Senators Charles Schumer (D-NY) and John Cornyn (R-TX) introduced a bill—S.3427: The Pre-Paid Mobile Device Identification Act—that would require that a consumer provide his or her name, address, and date of birth prior to the purchase of a pre-paid mobile device or SIM card. As of May 2010, the bill had been read twice and referred to the Committee on Commerce, Science, and Transportation.

References

1. TPM Main, Part 1 Design Principles, Specification Version 1.2, Level 2 Revision 103. Tech. rep., Trusted Computing Group (July 9 2007).
2. Bentham, J. *The Panopticon; or The Inspection House.* London, 1787. Miran Božovi (Ed.). Verso, London, UK, 1995.
3. *Berger v. New York*, 388 U.S. 41 (1967).
4. Communications Assistance for Law Enforcement Act (CALEA, 47 U.S.C. xx10011010).
5. Clarke, R.A. Information technology and dataveillance. *Commun. ACM 31*, 5 (May 1988), 498–512.
6. Cohen, J. E. Examined lives: Informational privacy and the subject as object. *Stanford Law Review* (2000).
7. Cortes, C., Pregibon, D., and Volinsky, C. Communities of interest. In *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis* (2001), 105-114.
8. Cuddihy, W.J. *The Fourth Amendment: Origins and Original Meaning*, 602–1791. Oxford University Press, 2009. (See also the Ph.D. thesis with the same title, Claremont Graduate School, 1990).
9. Deleuze, G. Postscript on the societies of control. October 59 (1992), 3–7. (Winter).
10. Diffie, W., and Hellman, M. New directions in cryptography. *IEEE Transactions on Information Theory 22*, 6 (1976), 644–654.
11. Dworkin, G. *The Theory and Practice of Autonomy.* University Press, Cambridge, 1988.
12. Electronic Communications Privacy Act.
13. Etemad, K. *CDMA 2000 Evolution: System Concepts and Design Principles.* Wiley, NY, 2004.
14. Implementation of the Telecommunications Act of 1996: Telecommunications Carriers Use of Customer Proprietary Network Information and Other Customer Information (1998).
15. Implementation of the Telecommunications Act of 1996: Telecommunications Carriers Use of Customer Proprietary Network Information and Other Customer Information, 17 F.C.C.R. 14860 (2002).
16. Implementation of the Telecommunications Act of 1996: Telecommunications Carriers Use of Customer Proprietary Network Information and Other Customer Information.
17. Foucault, M. *Discipline and Punish.* Vintage, 1995, (*Surveiller et punir: Naissance de la Prison*, 1975).
18. Freeh, L.J. Digital telephony and law enforcement access to advanced telecommunications technologies and services. Joint Hearings on H.R. 4922 and S. 2375, 103d Cong. 7, 1994.
19. Freiwald, S. First principles of communication privacy. *Stanford Technology Law Review 3* (2007).
20. Gandy, O.H. *The Panoptic Sort: A Political Economy of Personal Information.* Westview Publishers, 1993.
21. Goldstein, J., and Rotich, J. Digitally networked technology in kenya's 2007–2008 post-election crisis. Tech. Rep. 2008–09, Harvard University, Berkman Center for Internet & Society, Sept. 2008.
22. Holma, H., and Toskala, A. *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications*, 3rd Ed. Wiley, NY, 2004.
23. IMT-2000. International mobile telecommunications-2000 standard.
24. Kaaranen, H., Ahtiainen, A., Laitinen, L., Naghian, S. and Niemi, V. *UMTS Networks, 2nd Ed.* Wiley and Sons, Hoboken, NJ 2005.
25. *Katz v. United States*, 389 U.S. 347 (1967).
26. Kerr, O.S. Internet surveillance law after the USA Patriot Act: The big brother that isn't. *Northwestern University Law Review 97*, 2 (2002–2003), 607–611.
27. Lichtblau, E. Telecoms win dismissal of wiretap suits. *New York Times* (June 3 2009).
28. Loopt2010. Loopt strengthens its location-based advertising offerings, sets sights on hyperlocal marketing. *Mobile Marketing Watch* (Feb. 17, 2010).
29. *United States v. Miller*, 425 U.S. 435 (1976).
30. Mouly, M., and Pautet, M.-B. The GSM System for Mobile Communications. Self-published, 1992.
31. *Nardone v. United States*, 302 U.S. 379 (1937).
32. Networks, J. The benefits of router-integrated session border control. Tech. rep., Juniper Networks, 2009.
33. Nissenbaum, H. *Privacy in Context: Technology, Policy, and the Integrity of Social Life.* Stanford University Press, Palo Alto, CA, 2010.
34. *Olmstead v. United States*, 277 U.S. 438 (1928).
35. Pfitzmann, A., Pfitzmann, B., and Waidner, M. ISDN-MIXes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems* (1991). Springer-Verlag, 451–463.
36. Querengesser, T. Kenya: Hate speech SMS offenders already tracked (Mar. 2008).
37. Redish, M. *Freedom of Expression: A Critical Analysis.* Michie Co, Charlottesville, NC, 984.
38. Schoeman, F.D., Ed. *Philosophical Dimensions of Privacy: An Anthology.* Cambridge University Press, 1984.
39. Schrader, D.E. Intellectual safety, moral atmosphere, and epistemology in college classrooms. *Journal of Adult Development 11*, 2 (Apr. 2004).
40. Semayne's Case. Coke's Rep. 91a, 77 Eng. Rep. 194 (K.B. 1604).
41. Sherr, M., Cronin, E., Clark, S., and Blaze, M. Signaling vulnerabilities in wiretapping systems. *IEEE Security & Privacy 3*, 6 (2005), 13-25.
42. *Smith v. Maryland*, 442 U.S. 735 (1979).
43. Solove, D.J., and Schwartz, P.M. *Privacy, Information, and Technology; 2nd Ed.* Aspen Publishers, Inc., 2008.
44. Telecommunications Act of 1996.
45. Toeniskoetter, S.B. Preventing a modern panopticon: Law enforcement acquisition of real-time cellular tracking data. Rich. *J.L. & Tech. 13*, 4 (2007), 1–49.
46. Tversky, A., and Kahneman, D. The framing of decisions and the psychology of choice. *Science 211*, 4481 (Jan. 30 1981), 453-458.
47. *U.S. West, Inc. v. FCC*, 182 F.3d 1224 (10th Cir. 1999).
48. Williamson, J. *Decoding Advertisements: Ideology and Meaning in Advertising.* Marion Boyars Publishers Ltd, 1978.

Stephen B. Wicker (wicker@ece.cornell.edu) is a professor in the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY.

© 2011 ACM 0001-0782/11/07 $10.00

research highlights

P. 100
Technical Perspective
FAWN: A Fast Array of Wimpy Nodes
By Luiz André Barroso

P. 101
FAWN: A Fast Array of Wimpy Nodes
By David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan

P. 110
Technical Perspective
Is Scale Your Enemy, Or Is Scale Your Friend?
By John Ousterhout

P. 111
Debugging in the (Very) Large: Ten Years of Implementation and Experience
By Kinshuman Kinshumann, Kirk Glerum, Steve Greenberg, Gabriel Aul, Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt

DOI:10.1145/1965724.1965746

Technical Perspective
FAWN: A Fast Array of Wimpy Nodes

By Luiz André Barroso

INNOVATION IN COMPUTING systems thrives at the beginning and the end of technology cycles. When facing the limits of an existing technology or contemplating the applications of a brand new one, system designers are at their creative best. The past decade has been rich on both fronts, particularly for computer architects. CMOS technology scaling is no longer yielding the energy savings it used to provide across generations, resulting in severe thermal constraints leading to increased attention to so called "wimpy processors." These processors achieve high performance and energy efficiency by using a larger number of low-to-modest-speed CPU cores. Also in the past decade, the consumer electronics industry's investment in non-volatile storage technologies has resulted in NAND FLASH devices that are becoming competitive for general-purpose computing usage as they fit nicely within the huge cost/performance gap between DRAM and magnetic disks. FLASH-based storage devices are over 100 times faster than disks, although at over 10 times the cost per byte stored.

The emergence of wimpy processors and FLASH met a promising deployment scenario in the field of large-scale data centers for Internet services. These warehouse-scale computing (WSC) systems tend to run workloads that are rich in request-level parallelism—a match for the increased parallelism of wimpy CPUs—and are very data intensive—a match for the high input-output rates that are possible with FLASH technology. The energy efficiency potential of both these technologies could help lower the substantial energy-related costs of WSCs.

Given all this potential, how can we explain the rather slow pace of adoption of these technologies in commercial WSCs? At first glance, wimpy processors and FLASH seem compelling enough to fit within existing data center hardware and software architectures without the need for substantial redesign of major infrastructure components, thus facilitating rapid adoption. In reality, there are obstacles to extracting the maximum value from them. Hölzle[1] summarized some of the challenges facing wimpy cores in commercial deployments, including parallelization overheads (Amdahl's Law) and programmer productivity concerns. FLASH adoption has also suffered due to software related issues. FLASH will not fully replace disks for most workloads due to its higher costs, therefore storage system software must be adapted to use both FLASH and disk drives effectively.

FAWN combines wimpy cores and FLASH to create an efficient, high-throughput, key-value storage system.

The lesson here is that to extract the most value from compelling new technology one often needs to consider the system more broadly, and rethink how applications and infrastructure components might be changed in light of new hardware component characteristics. This is precisely what the authors of the following article on FAWN have done.

FAWN presents a new storage hardware architecture that takes advantage of wimpy cores and FLASH devices, but does so alongside a new datastore software system infrastructure (FAWN-DS) that is specifically targeted to the new hardware component characteristics. The system is not a generic distributed storage system, but one that is specialized for workloads that require high rates of key-value lookup queries. By co-designing the hardware and software, and by targeting the system for a particular (but compelling) use case, the authors present a solution that has greater potential to realize the full value of new energy-efficient components. Their approach, which includes building and experimenting with actual software and hardware artifacts, is a model worthy of being followed by future systems research projects. C

Reference

1. Hölzle, U. Brawny cores still beat wimpy cores, most of the time. *IEEE Micro* (Aug/Sept. 2010).

Luiz André Barroso (luiz@google.com) is a Distinguished Engineer at Google.

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965747

FAWN: A Fast Array of Wimpy Nodes

By David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan

Abstract

This paper presents a fast array of wimpy nodes—FAWN—an approach for achieving low-power data-intensive datacenter computing. FAWN couples low-power processors to small amounts of local flash storage, balancing computation and I/O capabilities. FAWN optimizes for per node energy efficiency to enable efficient, massively parallel access to data.

The key contributions of this paper are the principles of the FAWN approach and the design and implementation of FAWN-KV—a consistent, replicated, highly available, and high-performance key-value storage system built on a FAWN prototype. Our design centers around purely log-structured datastores that provide the basis for high performance on flash storage, as well as for replication and consistency obtained using chain replication on a consistent hashing ring. Our evaluation demonstrates that FAWN clusters can handle roughly 350 key-value *queries per Joule* of energy—two orders of magnitude more than a disk-based system.

1. INTRODUCTION

Large-scale data-intensive applications, such as high-performance key-value storage systems, are growing in both size and importance; they now are critical parts of major Internet services such as Amazon (Dynamo[7]), LinkedIn (Voldemort), and Facebook (`memcached`).

The workloads these systems support share several characteristics: They are I/O, not computation, intensive, requiring random access over large datasets; they are massively parallel, with thousands of concurrent, mostly independent operations; their high load requires large clusters to support them; and the size of objects stored is typically small, for example, 1KB values for thumbnail images, hundreds of bytes for wall posts, and twitter messages.

The clusters that serve these workloads must provide both high performance and low-cost operation. Unfortunately, small-object random-access workloads are particularly ill served by conventional disk-based or memory-based clusters. The poor seek performance of disks makes disk-based systems inefficient in terms of both system performance and performance per Watt. High-performance DRAM-based clusters, storing terabytes or petabytes of data, are expensive and power-hungry: Two high-speed DRAM DIMMs can consume as much energy as a 1TB disk.

The power draw of these clusters is becoming an increasing fraction of their cost—up to 50% of the 3 year total cost of owning a computer. The density of the datacenters that house them is in turn limited by their ability to supply and cool 10–20 kW of power per rack and up to 10–20 MW per datacenter.[12] Future datacenters may require as much as 200 MW,[12] and datacenters are being constructed today with dedicated electrical substations to feed them.

These challenges necessitate the question: Can we build a cost-effective cluster for data-intensive workloads that uses less than a tenth of the power required by a conventional architecture, but that still meets the same capacity, availability, throughput, and latency requirements?

The FAWN approach is designed to address this question. FAWN couples low-power, efficient CPUs with flash storage to provide efficient, fast, and cost-effective access to large, random-access data. Flash is faster than disk, cheaper than DRAM, and consumes less power than either. Thus, it is a particularly suitable choice for FAWN and its workloads. FAWN represents a class of systems that targets both system balance and per node energy efficiency: The 2008-era FAWN prototypes used in this work used embedded CPUs and CompactFlash, while today a FAWN node might be composed of laptop processors and higher-speed SSDs. Relative to today's highest-end computers, a contemporary FAWN system might use dual or quad-core 1.6 GHz CPUs with 1–4GB of DRAM.

To show that it is *practical* to use these constrained nodes as the core of a large system, we designed and built the FAWN-KV cluster-based key-value store, which provides storage functionality similar to that used in several large enterprises.[7] FAWN-KV is designed to exploit the advantages and avoid the limitations of wimpy nodes with flash memory for storage.

The key design choice in FAWN-KV is the use of a *log-structured per node datastore* called FAWN-DS that provides high-performance reads and writes using flash memory. This append-only data log provides the basis for replication and strong consistency using *chain replication*[21] between nodes. Data is distributed across nodes using consistent hashing, with data split into contiguous ranges on disk such that all replication and node insertion operations involve only a fully in-order traversal of the subset of data that must be copied to a new node. Together with the log structure, these properties combine to provide fast failover and fast node insertion, and they minimize the time the affected datastore's key range is locked during such operations.

The original version of this paper was published in *Proceedings of the 22nd ACM Symposium of Operating Systems Principles,* October 2009.

We have built a prototype 21-node FAWN cluster using 500 MHz embedded CPUs. Each node can serve up to 1300 256 byte queries/s, exploiting nearly all of the raw I/O capability of their attached flash devices, and consumes under 5 W when network and support hardware is taken into account. The FAWN cluster achieves 330 queries/J—two orders of magnitude better than traditional disk-based clusters.

2. WHY FAWN?

The FAWN approach to building *well-matched* cluster systems has the potential to achieve high performance and be fundamentally more energy-efficient than conventional architectures for serving massive-scale I/O and data-intensive workloads. We measure system performance in queries per second and measure energy efficiency in queries per Joule (equivalently, queries per second per Watt). FAWN is inspired by several fundamental trends:

Increasing CPU-I/O gap: Over the past several decades, the gap between CPU performance and I/O bandwidth has continually grown. For data-intensive computing workloads, storage, network, and memory bandwidth bottlenecks often cause low CPU utilization.

FAWN approach: To efficiently run I/O-bound data-intensive, computationally simple applications, FAWN uses wimpy processors selected to reduce I/O-induced idle cycles while maintaining high performance. The reduced processor speed then benefits from a second trend.

CPU power consumption grows super-linearly with speed: Higher frequencies require more energy, and techniques to mask the CPU-memory bottleneck come at the cost of energy efficiency. Branch prediction, speculative execution, out-of-order execution and large on-chip caches all require additional die area; modern processors dedicate as much as half their die to L2/3 caches.[9] These techniques do not increase the speed of basic computations, but do increase power consumption, making faster CPUs less energy efficient.

FAWN approach: A FAWN cluster's slower CPUs dedicate proportionally more transistors to basic operations. These CPUs execute significantly more *instructions per Joule* than their faster counterparts: Multi-GHz superscalar quad-core processors can execute approximately 100 million instructions/J, assuming all cores are active and avoid stalls or mispredictions. Lower-frequency in-order CPUs, in contrast, can provide over 1 billion instructions/J—an order of magnitude more efficient while running at 1/3 the frequency.

Worse yet, running fast processors below their full capacity draws a disproportionate amount of power.

Dynamic power scaling on traditional systems is surprisingly inefficient: A primary energy-saving benefit of dynamic voltage and frequency scaling (DVFS) was its ability to reduce voltage as it reduced frequency, but modern CPUs already operate near minimum voltage at the highest frequencies.

Even if processor energy was completely proportional to load, non-CPU components such as memory, motherboards, and power supplies have begun to dominate energy consumption,[2] requiring that all components be scaled back with demand. As a result, a computer may consume over 50% of its peak power when running at only 20% of its capacity.[20] Despite improved power scaling technology, systems remain most energy efficient when operating at peak utilization.

A promising path to energy proportionality is turning machines off entirely.[6] Unfortunately, these techniques do not apply well to FAWN-KV's target workloads: Key-value systems must often meet service-level agreements for query throughput and latency of hundreds of milliseconds; the inter-arrival time and latency bounds of the requests prevent shutting machines down (and taking many seconds to wake them up again) during low load.[2]

Finally, energy proportionality alone is not a panacea: Systems should be both proportional *and* efficient at 100% load. FAWN specifically addresses efficiency, and cluster techniques that improve proportionality should apply universally.

3. DESIGN AND IMPLEMENTATION

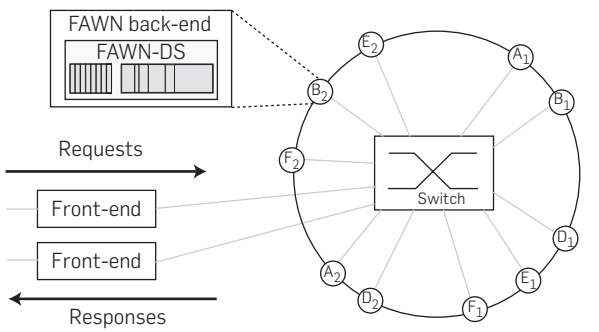
We describe the design and implementation of the system components from the bottom up: a brief overview of flash storage (Section 3.2), the per node FAWN-DS datastore (Section 3.3), and the FAWN-KV cluster key-value lookup system (Section 3.4), including replication and consistency.

3.1. Design overview

Figure 1 gives an overview of the entire FAWN system. Client requests enter the system at one of several *front ends*. The front-end nodes forward the request to the *back-end* FAWN-KV node responsible for serving that particular key. The back-end node serves the request from its FAWN-DS datastore and returns the result to the front end (which in turn replies to the client). Writes proceed similarly.

The large number of back-end FAWN-KV storage nodes is organized into a ring using consistent hashing. As in systems such as Chord,[18] keys are mapped to the node that follows the key in the ring (its *successor*). To balance load and reduce failover times, each *physical node* joins the ring as a small number (V) of *virtual nodes*, each virtual node representing a *virtual ID* ("*VID*") in the ring space. Each physical node is thus responsible for V different (noncontiguous) key ranges. The data associated with each virtual ID is stored on flash using FAWN-DS.

Figure 1. FAWN-KV architecture.



3.2. Understanding flash storage

Flash provides a non-volatile memory store with several significant benefits over typical magnetic hard disks for random-access, read-intensive workloads—but it also introduces several challenges. Three characteristics of flash underlie the design of the FAWN-KV system described in this section:

1. **Fast random reads:** ($\ll$1 ms) up to 175 times faster than random reads on magnetic disk.[17]
2. **Efficient I/O:** Many flash devices consume less than 1 W even under heavy load, whereas mechanical disks can consume over 10 W at load.
3. **Slow random writes:** Small writes on flash are expensive. Updating a single page requires first erasing an entire erase block (128–256KB) of pages and then writing the modified block in its entirety. Updating a single byte of data is therefore as expensive as writing an entire block of pages.[16]

Modern devices improve random write performance using write buffering and preemptive block erasure. These techniques improve performance for short bursts of writes, but sustained random writes still underperform.[17]

These performance problems motivate log-structured techniques for flash filesystems and data structures.[10, 15, 16] These same considerations inform the design of FAWN's node storage management system, described next.

3.3. The FAWN datastore

FAWN-DS is a log-structured key-value store. Each store contains values for the key range associated with one virtual ID. It acts to clients like a disk-based hash table that supports `Store`, `Lookup`, and `Delete`.

FAWN-DS is designed to perform well on flash storage and to operate within the constrained DRAM available on wimpy nodes: All writes to the datastore are sequential, and reads require a single random access. To provide this property, FAWN-DS maintains an in-DRAM hash table (Hash Index) that maps keys to an offset in the append-only Data Log on flash (Figure 2a). This log-structured design is similar to several append-only filesystems such as the Google File System (GFS) and Venti, which avoid random seeks on magnetic disks for writes.

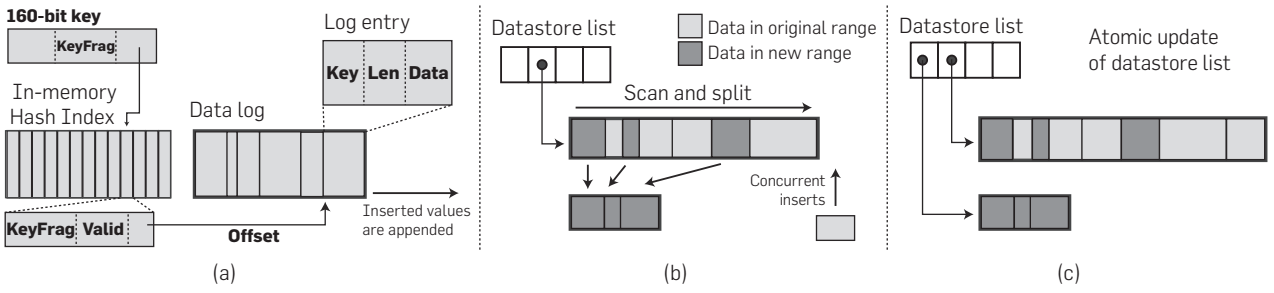
Mapping a key to a value: FAWN-DS uses an in-memory (DRAM) Hash Index to map 160 bit keys to a value stored in the Data Log. It stores only a fragment of the actual key in memory to find a location in the log; it then reads the full key (and the value) from the log and verifies that the key it read was, in fact, the correct key. This design trades a small and configurable chance of requiring two reads from flash (we set it to roughly 1 in 32,768 accesses) for drastically reduced memory requirements (only 6 bytes of DRAM per key-value pair).

FAWN-DS's Lookup procedure extracts two fields from the 160 bit key: The i low order bits of the key (the *index bits*) and the next 15 low order bits (the *key fragment*). FAWN-DS uses the index bits to select a bucket from the Hash Index, which contains 2^i hash buckets. Each bucket is 6 bytes: a 15 bit key fragment, a valid bit, and a 4 byte pointer to the location in the Data Log where the full entry is stored.

Lookup proceeds, then, by locating a bucket using the index bits and comparing the key against the key fragment. If the fragments do not match, FAWN-DS uses hash chaining to continue searching the hash table. Once it finds a matching key fragment, FAWN-DS reads the record off of the flash. If the stored full key in the on-flash record matches the desired lookup key, the operation is complete. Otherwise, FAWN-DS resumes its hash chaining search of the in-memory hash table and searches additional records. With the 15-bit key fragment, only 1 in 32,768 retrievals from the flash will be incorrect and require fetching an additional record.

The constants involved (15 bits of key fragment, 4 bytes of log pointer) target the prototype FAWN nodes described in Section 4. A typical object is between 256 bytes and 1KB, and the nodes have 256MB of DRAM and approximately 4GB of flash storage. Because each physical node is responsible for V key ranges (each with its own datastore file), it can address 4GB $*$ V bytes of data. Expanding the in-memory storage to 7 bytes per entry would permit FAWN-DS to address 1TB of data per key range. While some additional optimizations are possible, such as rounding the size of objects stored in flash or reducing the number of bits used for the key fragment (and thus incurring, e.g., a 1-in-1000 chance of having to do two reads from flash), the current design works well for the key-value workloads we study.

Figure 2. (a) FAWN-DS appends writes to the end of the Data Log. (b) Split requires a sequential scan of the data region, transferring out-of-range entries to the new store. (c) After scan completes, the datastore list is atomically updated to add the new store. Compaction of the original store cleans up out-of-range entries.



Reconstruction: The Data Log contains all the information necessary to reconstruct the Hash Index from scratch. As an optimization, FAWN-DS periodically checkpoints the index by writing the Hash Index and a pointer to the last log entry to flash. After a failure, FAWN-DS uses the checkpoint as a starting point to reconstruct the in-memory Hash Index.

Virtual IDs and semi-random writes: A physical node has a separate FAWN-DS datastore file for each of its virtual IDs, and FAWN-DS appends new or updated data items to the appropriate datastore. Sequentially appending to a small number of files is termed *semi-random writes*. With many flash devices, these semi-random writes are nearly as fast as a single sequential append.[15] We take advantage of this property to retain fast write performance while allowing key ranges to be stored in independent files to speed the maintenance operations described in the following.

3.3.1. Basic functions: Store, lookup, delete

`Store` appends an entry to the log, updates the corresponding hash table entry to point to the offset of the newly appended entry within the Data Log, and sets the valid bit to true. If the key written already existed, the old value is now *orphaned* (no hash entry points to it) for later garbage collection.

`Lookup` retrieves the hash entry containing the offset, indexes into the Data Log, and returns the data blob.

`Delete` invalidates the hash entry corresponding to the key and writes a *Delete entry* to the end of the data file. The delete entry is necessary for fault tolerance—the invalidated hash table entry is not immediately committed to nonvolatile storage to avoid random writes, so a failure following a delete requires a log to ensure that recovery will delete the entry upon reconstruction. Because of its log structure, FAWN-DS deletes are similar to store operations with 0 byte values. Deletes do not immediately reclaim space and require compaction to perform garbage collection. This design defers the cost of a random write to a later sequential write operation.

3.3.2. Maintenance: Split, merge, compact

Inserting a new virtual node into the ring causes one key range to split into two, with the new virtual node gaining responsibility for the first part of it. Nodes handling these *VIDs* must therefore `Split` their datastore into two datastores, one for each key range. When a virtual node departs the system, two adjacent key ranges must similarly `Merge` into a single datastore. In addition, a virtual node must periodically `Compact` its datastores to clean up stale or orphaned entries created by `Split`, `Store`, and `Delete`.

These maintenance functions are designed to work well on flash, requiring only scans of one datastore and sequential writes into another.

`Split` parses the Data Log sequentially, writing each entry in a new datastore if its key falls in the new datastore's range.

`Merge` writes every log entry from one datastore into the other datastore; because the key ranges are independent, it does so as an append. Split and Merge propagate delete entries into the new datastore.

`Compact` cleans up entries in a datastore, similar to garbage collection in a log-structured filesystem. It skips entries that fall outside of the datastore's key range, which may be leftover after a split. It also skips orphaned entries that no in-memory hash table entry points to, and then skips any delete entries corresponding to those entries. It writes all other valid entries into the output datastore.

3.3.3. Concurrent maintenance and operation

All FAWN-DS maintenance functions allow concurrent reads and writes to the datastore. `Stores` and `Deletes` only modify hash table entries and write to the end of the log.

Maintenance operations (`Split`, `Merge`, and `Compact`) sequentially parse the Data Log, which may be growing due to deletes and stores. Because the log is append only, a log entry once parsed will never be changed. These operations each create one new output datastore logfile. The maintenance operations run until they reach the end of the log, and then briefly lock the datastore, ensure that all values flushed to the old log have been processed, update the FAWN-DS datastore list to point to the newly created log, and release the lock (Figure 2c).

3.4. The FAWN key-value system

In FAWN-KV, client applications send requests to front ends using a standard put/get interface. Front ends send the request to the back-end node that owns the key space for the request. The back-end node satisfies the request using its FAWN-DS and replies to the front ends.

3.4.1. Consistent hashing: Key ranges to nodes

A typical FAWN cluster will have several front ends and many back ends. FAWN-KV organizes the back-end *VIDs* into a storage ring-structure using consistent hashing.[18] Front ends maintain the entire node membership list and directly forward queries to the back-end node that contains a particular data item.

Each front-end node manages the *VID* membership list and queries for a large contiguous chunk of the key space. A front end receiving queries for keys outside of its range forwards the queries to the appropriate front-end node. This design either requires clients to be roughly aware of the front-end mapping or doubles the traffic that front ends must handle, but it permits front ends to cache values without a cache consistency protocol.

The key space is allocated to front ends by a single management node; we envision this node being replicated using a small Paxos cluster,[13] but we have not (yet) implemented this. There would be 80 or more back-end nodes per front-end node with our current hardware prototypes, so the amount of information this management node maintains is small and changes infrequently—a list of 125 front ends would suffice for a 10,000 node FAWN cluster. When a back-end node joins, it obtains the list of front-end IDs. It uses this list to determine which front ends to contact to join the ring, one *VID* at a time. We chose this design so that the system would be robust to front-end node failures: The back-end node identifier (and thus, what keys

it is responsible for) is a deterministic function of the back-end node ID. If a front-end node fails, data does not move between back-end nodes, though virtual nodes may have to attach to a new front end.

FAWN-KV uses a 160 bit circular ID space for *VIDs* and keys. Virtual IDs are hashed identifiers derived from the node's address. Each *VID owns* the items for which it is the item's successor in the ring space (the node immediately clockwise in the ring). As an example, consider the cluster depicted in Figure 3 with five physical nodes, each of which has two *VIDs*. The physical node *A* appears as *VIDs A*1 and *A*2, each with its own 160 bit identifiers. *VID A*1 owns key range *R*1, *VID B*1 owns range *R*2, and so on.

3.4.2. Replication and consistency

FAWN-KV offers a configurable replication factor for fault tolerance. Items are stored at their successor in the ring space and at the $R-1$ following virtual IDs. FAWN-KV uses chain replication[21] to provide strong consistency on a per key basis. Updates are sent to the head of the chain, passed along to each member of the chain via a TCP connection between the nodes, and queries are sent to the tail of the chain. By mapping chain replication to the consistent hashing ring, each virtual ID in FAWN-KV is part of *R* different chains: it is the "tail" for one chain, a "mid" node in $R-2$ chains, and the "head" for one. Figure 4 depicts a ring with six physical nodes, where each has two virtual IDs ($V=2$), using a replication factor of 3. In this figure, node *C*l is the tail for range *R*l, mid for range *R*2, and tail for range *R*3.

Figure 3. Consistent hashing with five physical nodes and two virtual IDs each.

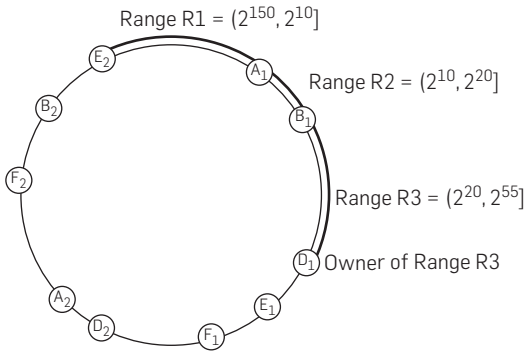


Figure 4. Overlapping chains in the ring—each node in the ring is part of *R* = 3 chains.

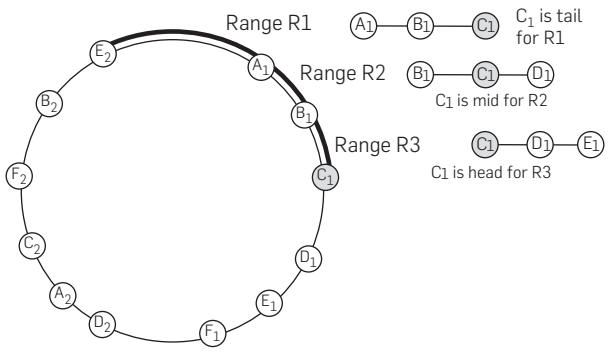


Figure 5. Life cycle of a put with chain replication—puts go to the head and are propagated through the chain. Gets go directly to the tail.

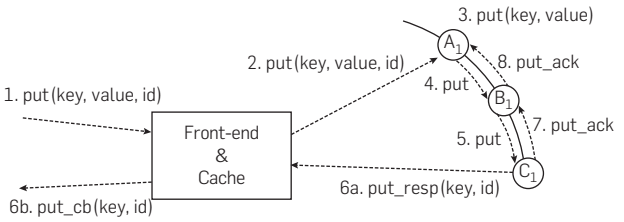


Figure 5 shows a put request for an item in range *R*1. The front end sends the put to the key's successor, *VID A*1, which is the head of the replica chain for this range. After storing the value in its datastore, *A*1 forwards this request to *B*1, which stores the value and forwards the request to the tail, *C*1. After storing the value, *C*l sends the put response back to the front end and sends an acknowledgment back up the chain indicating that the response was handled properly.

For reliability, nodes buffer put requests until they receive the acknowledgment. Because puts are written to an append-only log in FAWN-DS and are sent in-order along the chain, this operation is simple: nodes maintain a pointer to the last unacknowledged put in their datastore and increment it when they receive an acknowledgment. By using a log-structured datastore, chain replication in FAWN-KV reduces to simply streaming the datastore from node to node.

Get requests proceed as in chain replication—the front end directly routes gets to the tail of the chain for range *R*1, node *C*l, which responds to requests. Any update seen by the tail has therefore also been applied by other replicas in the chain.

4. EVALUATION

We begin by characterizing the baseline I/O performance of a node. We then show that FAWN-DS's performance is similar to the node's baseline I/O capability. To illustrate the advantages of FAWN-DS's design, we compare its performance to an implementation using the general-purpose BerkeleyDB, which is not optimized for flash writes. We then study a prototype FAWN-KV system running on a 21-node cluster, evaluating its energy efficiency in queries per second per Watt.

Evaluation hardware: Our FAWN cluster has 21 back-end nodes built from commodity PCEngine Alix 3c2 devices, commonly used for thin clients, kiosks, network firewalls, wireless routers, and other embedded applications. These devices have a single-core 500 MHz AMD Geode LX processor, 256MB DDR SDRAM operating at 400 MHz, and 100 Mbit/s Ethernet. Each node contains one 4GB Sandisk Extreme IV CompactFlash device. A node consumes 3 W when idle and a maximum of 6 W when using 100% CPU, network, and flash. The nodes are connected to each other

and to a 27W Intel Atom-based front-end node using two 16-port Netgear GS116 GigE Ethernet switches.

Evaluation workload: We show query performance for 256 byte and 1KB values. We select these sizes as proxies for small text posts, user reviews or status messages, image thumbnails, and so on. They represent a quite challenging regime for conventional disk-bound systems and stress the limited memory and CPU of our wimpy nodes.

4.1. Individual node performance

We benchmark the I/O capability of the FAWN nodes using iozone and Flexible I/O tester. The flash is formatted with the ext2 filesystem. These tests read and write 1KB entries, the lowest record size available in iozone. The filesystem I/O performance using a 3.5GB file is shown in Table 1.

4.1.1. FAWN-DS single node local benchmarks

Lookup speed: This test shows the query throughput achieved by a local client issuing queries for randomly distributed, existing keys on a single node. We report the average of three runs (the standard deviations were below 5%). Table 2 shows FAWN-DS 1KB and 256 byte random read queries/s as a function of the DS size. If the datastore fits in the buffer cache, the node locally retrieves 50,000–85,000 queries/s. As the datastore exceeds the 256MB of RAM available on the nodes, a larger fraction of requests go to flash.

FAWN-DS imposes modest overhead from hash lookups, data copies, and key comparisons; and it must read slightly more data than the iozone tests (each stored entry has a header). The query throughput, however, remains high: Tests reading a 3.5GB datastore using 1KB values achieved 1,150 queries/s compared to 1,424 queries/s from the filesystem. Using 256 byte entries achieved 1,298 queries/s from a 3.5GB datastore. By comparison, the raw filesystem achieved 1,454 random 256 byte reads/s using Flexible I/O.

Bulk `store` speed: The log structure of FAWN-DS ensures that data insertion is entirely sequential. Inserting 2 million entries of 1KB each (2GB total) into a *single* FAWN-DS log proceeds at 23.2MB/s (nearly 24,000 entries/s), which is 96% of the raw speed that the flash can be written through the filesystem.

Table 1. Baseline CompactFlash statistics for 1KB entries. QPS = Queries/second.

Seq. Read	*Rand Read*	*Seq. Write*	*Rand. Write*
28.5MB/s	1424 QPS	24MB/s	110 QPS

Table 2. Local random read speed of FAWN-DS.

DS Size	*1KB Rand Read* (in queries/s)	*256 bytes Rand Read* (in queries/s)
10KB	72352	85012
125MB	51968	65412
250MB	6824	5902
500MB	2016	2449
1GB	1595	1964
2GB	1446	1613
3.5GB	1150	1298

`Put` speed: Each FAWN-KV node has $R * V$ FAWN-DS files: Each virtual ID adds one primary data range, plus an additional $R - 1$ replicated ranges. A node receiving puts for different ranges will concurrently append to a small number of files ("semi-random writes"). Good semi-random write performance is central to FAWN-DS's per range data layout that enables single-pass maintenance operations. Our recent work confirms that modern flash devices can provide good semi-random write performance.[1]

4.1.2. Comparison with BerkeleyDB

To understand the benefit of FAWN-DS's log structure, we compare with a general purpose disk-based database that is *not* optimized for flash. BerkeleyDB provides a simple put/get interface, can be used without heavy-weight transactions or rollback, and performs well vs. other memory or disk-based databases. We configured BerkeleyDB using both its default settings and using the reference guide suggestions for flash-based operation.[3] The best performance we achieved required 6 hours to insert 7 million, 200 byte entries to create a 1.5GB B-Tree database. This corresponds to an insert rate of 0.07MB/s.

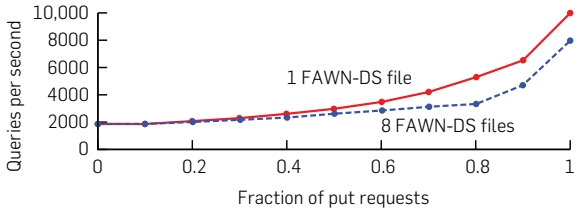
The problem was, of course, small writes: When the BDB store was larger than the available RAM on the nodes (<256MB), BDB had to flush pages to disk, causing many writes that were much smaller than the size of an erase block.

That comparing FAWN-DS and BDB seems unfair is exactly the point: Even a well-understood, high-performance database will perform poorly when its write pattern has not been specifically optimized to flash characteristics. We evaluated BDB on top of NILFS2, a log-structured Linux filesystem for block devices, to understand whether log-structured writing could turn the random writes into sequential writes. Unfortunately, this combination was not suitable because of the amount of metadata created for small writes for use in filesystem checkpointing and rollback, features not needed for FAWN-KV—writing 200MB worth of 256 bytes key-value pairs generated 3.5GB of metadata. Other existing Linux log-structured flash filesystems, such as JFFS2, are designed to work on raw flash, but modern SSDs, compact flash, and SD cards all include a Flash Translation Layer that hides the raw flash chips. While future improvements to filesystems can speed up naive DB performance on flash, the pure log structure of FAWN-DS remains necessary even if we could use a more conventional back end: It provides the basis for replication and consistency across an array of nodes.

4.1.3. Read-intensive vs. write-intensive workloads

Most read-intensive workloads have some writes. For example, Facebook's memcached workloads have a 1:6 ratio of application-level puts to gets.[11] We therefore measured the aggregate query rate as the fraction of puts ranging from 0

Figure 6. FAWN supports both read- and write-intensive workloads. Small writes are cheaper than random reads due to the FAWN-DS log structure.



(all gets) to 1 (all puts) on a single node (Figure 6).

FAWN-DS can handle more puts per second than gets because of its log structure. Even though semi-random write performance across eight files on our CompactFlash devices is worse than purely sequential writes, it still achieves higher throughput than pure random reads.

When the put-ratio is low, the query rate is limited by the get requests. As the ratio of puts to gets increases, the faster puts significantly increase the aggregate query rate. On the other hand, a pure write workload that updates a small subset of keys would require frequent cleaning. In our current environment and implementation, both read and write rates slow to about 700–1000 queries/s during compaction, bottlenecked by increased thread switching and system call overheads of the cleaning thread. Last, because deletes are effectively 0 byte value puts, delete-heavy workloads are similar to insert workloads that update a small set of keys frequently. In the next section, we mostly evaluate read-intensive workloads because it represents the target workloads for which FAWN-KV is designed.

4.2. FAWN-KV system benchmarks

System throughput: To measure query throughput, we populated the KV cluster with 20GB of values and then measured the maximum rate at which the front end received query responses for random keys. Figure 7 shows that the cluster sustained roughly 36,000 256 byte gets per second (1,700 per second per node) and 24,000 1KB gets per second (1,100 per second per node). A single node serving a 512MB datastore over the network could sustain roughly 1,850 256 byte gets per second per node, while Table 2 shows that it could serve the queries locally at 2,450 256 byte queries per second per node. Thus, a single node serves roughly 70% of the sustained rate that a single FAWN-DS could handle with

Figure 7. Query throughput on 21-node FAWN-KV system for 1KB and 256 bytes entry sizes.

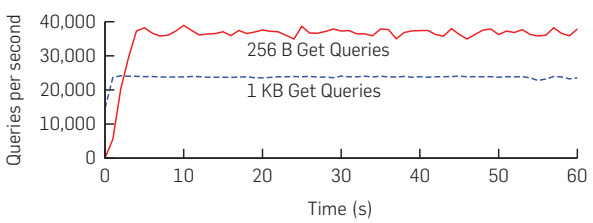
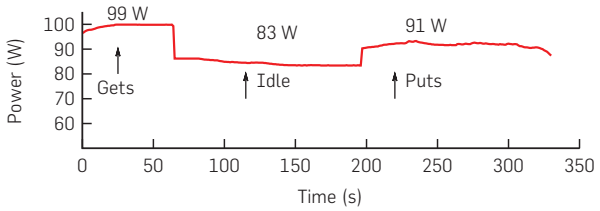


Figure 8. Power consumption of 21-node FAWN-KV system for 256 bytes values during Puts/Gets.



local queries. The primary reasons for the difference are the addition of network overhead, request marshaling and unmarshaling, and load imbalance—with random key distribution, some back-end nodes receive more queries than others, slightly reducing system performance.

System power consumption: Using a WattsUp power meter that logs power draw each second, we measured the power consumption of our 21-node FAWN-KV cluster and two network switches. Figure 8 shows that, when idle, the cluster uses about 83 W, or 3 W/node and 10 W/switch. During gets, power consumption increases to 99 W, and during insertions, power consumption is 91 W. Peak get performance reaches about 36,000 256 bytes queries/s for the cluster serving the 20GB dataset, so this system, excluding the front end, provides 364 queries/J.

The front end connects to the back-end nodes through a 1 Gbit/s uplink on the switch, so the cluster requires about one low-power front end for every 80 nodes—enough front ends to handle the aggregate query traffic from all the back ends (80 nodes * 1500 queries/s/node * 1KB/query = 937 Mbit/s). Our prototype front end uses 27 W, which adds nearly 0.5 W/node amortized over 80 nodes, providing 330 queries/J for the entire system. A high-speed (4 ms seek time, 10 W) magnetic disk by itself provides less than 25 queries/J—two orders of magnitude fewer than our existing FAWN prototype.

Network switches currently account for 20% of the power used by the entire system. Moving to FAWN requires roughly one 8-to-1 aggregation switch to make a group of FAWN nodes look like an equivalent-bandwidth server; we account for this in our evaluation by including the power of the switch when evaluating FAWN-KV. As designs such as FAWN reduce the power drawn by servers, the importance of creating scalable, energy-efficient datacenter networks will grow.

5. ALTERNATIVE ARCHITECTURES

When is the FAWN approach likely to beat traditional architectures? We examine this question by comparing the 3 year total cost of ownership (TCO) for six systems: Three "traditional" servers using magnetic disks, flash SSDs, and DRAM; and three hypothetical FAWN-like systems using the same storage technologies. We define the 3 year TCO as the sum of the capital cost and the 3 year power cost at 10 cents/kWh.

Because the FAWN systems we have built use several-year-old technology, we study a theoretical 2009 FAWN node

using a low-power CPU that consumes 10W–20 W and costs ~$150 in volume. We in turn give the benefit of the doubt to the server systems we compare against—we assume a 2 TB disk exists that serves 300 queries/s at 10 W.

Our results indicate that both FAWN and traditional systems have their place—but for the small random-access workloads we study, traditional systems are surprisingly absent from much of the solution space, in favor of FAWN nodes using either disks, flash, or DRAM.

Key to the analysis is a question: *Why does a cluster need nodes?* The answer is, of course, for both storage space and query rate. Storing a *DS* gigabyte dataset with query rate *QR* requires *N* nodes:

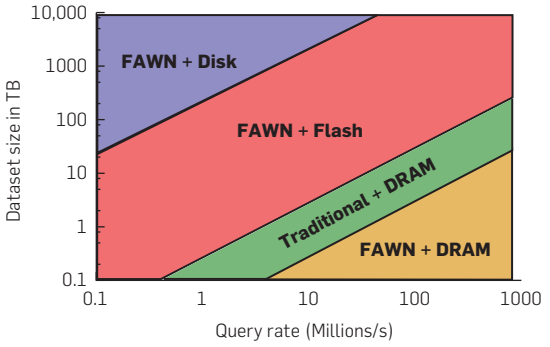
$$N = max\left(\frac{DS}{\frac{gb}{node}}, \frac{QR}{\frac{qr}{node}}\right)$$

With large datasets with low query rates, the number of nodes required is dominated by the storage capacity per node: Thus, the important metric is the total cost per GB for an individual node. Conversely, for small datasets with high query rates, the per node query capacity dictates the number of nodes: the dominant metric is queries per second per dollar. Between these extremes, systems must provide the best trade-off between per node storage capacity, query rate, and power cost.

Table 3 shows these cost and speculative performance statistics for several candidate systems circa 2009; while the numbers are outdated, the trends likely still apply. The "traditional" nodes use 200 W servers that cost $1,000 each. *Traditional+Disk* pairs a single server with five 2 TB high-speed (10,000 RPM) disks capable of 300 queries/s, each disk consuming 10 W. *Traditional+SSD* uses two PCI-E Fusion-IO 80GB flash SSDs, each also consuming about 10 W (Cost: $3 K). *Traditional+DRAM* uses 8GB server-quality DRAM modules, each consuming 10 W. *FAWN+Disk* nodes use one 2 TB 7200 RPM disk: FAWN nodes have fewer connectors available on the board. *FAWN+SSD* uses one 32GB Intel SATA flash SSD capable of 35,000 random reads/s,[17] consuming 2 W ($400). *FAWN+DRAM* uses a single 2GB, slower DRAM module, also consuming 2 W.

Figure 9 shows which base system has the lowest cost for a particular dataset size and query rate, with dataset sizes between 100GB and 10PB and query rates between 100 K and 1 billion/s.

Figure 9. Solution space for lowest 3 year TCO as a function of dataset size and query rate.



Large datasets, low query rates: *FAWN+Disk* has the lowest total cost per GB. While not shown on our graph, a traditional system wins for exabyte-sized workloads if it can be configured with sufficient disks per node (over 50), though packing 50 disks per machine poses reliability challenges.

Small datasets, high query rates: *FAWN+DRAM* costs the fewest dollars per queries per second, keeping in mind that we do *not* examine workloads that fit entirely in L2 cache on a traditional node. This somewhat counterintuitive result is similar to that made by the intelligent RAM project, which coupled processors and DRAM to achieve similar benefits[4] by avoiding the memory wall. We assume the FAWN nodes can only accept 2GB of DRAM per node, so for larger datasets, a traditional DRAM system provides a high query rate and requires fewer nodes to store the same amount of data (64GB vs. 2GB/node).

Middle range: *FAWN+SSDs* provide the best balance of storage capacity, query rate, and total cost. If SSD cost per GB improves relative to magnetic disks, this combination is likely to continue expanding into the range served by *FAWN+Disk*; if the SSD cost per performance ratio improves relative to DRAM, so will it reach into DRAM territory. It is therefore conceivable that *FAWN+SSD* could become the dominant architecture for many random-access workloads.

Are traditional systems obsolete? We emphasize that this analysis applies only to small, random-access workloads.

Table 3. Traditional and FAWN node statistics.

System	Cost	W	QPS	Queries/Joule	GB/Watt	TCO/GB	TCO/QPS
Traditionals							
5–2TB Disks	$2K	250	1500	6	40	0.26	1.77
160GB PCIe SSD	$8K	220	200K	909	0.72	53	0.04
64GB DRAM	$3K	280	1M	3.5K	0.23	59	0.004
FAWNs							
2TB Disk	$350	20	250	12.5	100	0.20	1.61
32GB SSD	$500	15	35K	2.3K	2.1	16.9	0.015
2GB DRAM	$250	15	100K	6.6K	0.13	134	0.003

Sequential-read workloads are similar, but the constants depend strongly on the per byte processing required. Traditional cluster architectures retain a place for CPU-bound workloads, but we do note that architectures such as IBM's BlueGene successfully apply large numbers of low-power, efficient processors to many supercomputing applications—but they augment their wimpy processors with custom floating point units to do so.

Our definition of "total cost of ownership" ignores several notable costs: In comparison to traditional architectures, FAWN should reduce power and cooling infrastructure but may increase network-related hardware and power costs due to the need for more switches. Our current hardware prototype improves work done per volume, thus reducing costs associated with datacenter rack or floor space. Finally, our analysis assumes that cluster software developers can engineer away the human costs of management—an optimistic assumption for all architectures. We similarly ignore issues such as ease of programming, though we selected an x86-based wimpy platform for ease of development.

6. RELATED WORK

Several projects are using low-power processors for datacenter workloads to reduce energy consumption.[5, 8, 14, 19] These systems leverage low-cost, low-power commodity components for datacenter systems, similarly arguing that this approach can achieve the highest work per dollar and per Joule. More recently, ultra-low power server systems have become commercially available, with companies such as SeaMicro, Marvell, Calxeda, and ZT Systems producing low-power datacenter computing systems based on Intel Atom and ARM platforms.

FAWN builds upon these observations by demonstrating the importance of re-architecting the software layers in obtaining the potential energy efficiency such hardware can provide.

7. CONCLUSION

The FAWN approach uses nodes that target the "sweet spot" of per node energy efficiency, typically operating at about half the frequency of the fastest available CPUs. Our experience in designing systems using this approach, often coupled with fast flash memory, has shown that it has substantial potential to improve energy efficiency, but that these improvements may come at the cost of re-architecting software or algorithms to operate with less memory, slower CPUs, or the quirks of flash memory: The FAWN-KV key-value system presented here is one such example. By successfully adapting the software to this efficient hardware, our then four-year-old FAWN nodes delivered over an order of magnitude more queries per Joule than conventional disk-based systems.

Our ongoing experience with newer FAWN-style systems shows that its energy efficiency benefits remain achievable, but that further systems challenges—such as high kernel I/O overhead—begin to come into play. In this light, we view our experience with FAWN as a potential harbinger of the systems challenges that are likely to arise for future many-core energy-efficient systems.

Acknowledgments

This work was supported in part by gifts from Network Appliance, Google, and Intel Corporation, and by grant CCF-0964474 from the National Science Foundation, as well as graduate fellowships from NSF, IBM, and APC. We extend our thanks to our OSDI and SOSP reviewers, Vyas Sekar, Mehul Shah, and to Lorenzo Alvisi for shepherding the work for SOSP. Iulian Moraru provided feedback and performance-tuning assistance. C

References

1. Andersen, D.G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., Vasudevan, V. FAWN: A fast array of wimpy nodes. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)* (Big Sky, MT, October 2009).
2. Barroso, L.A., Hölzle, U. The case for energy-proportional computing. *Computer 40*, 12 (2007), 33–37.
3. Memory-only or Flash configurations. http://www.oracle.com/technology/documentation/ berkeley-db/db/ref/program/ram.html
4. Bowman, W., Cardwell, N., Kozyrakis, C., Romer, C., Wang, H. Evaluation of existing architectures in IRAM systems. In *Workshop on Mixing Logic and DRAM, 24th International Symposium on Computer Architecture* (Denver, CO, June 1997).
5. Caulfield, A.M., Grupp, L.M., Swanson, S. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. In *14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)* (San Diego, CA, March 2009).
6. Chase, J.S., Anderson, D., Thakar, P., Vahdat, A., Doyle, R. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)* (Banff, AB, Canada, October 2001).
7. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)* (Stevenson, WA, Oct. 2007).
8. Hamilton, J. Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for Internet scale services, http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_CEHS.pdf (2009).
9. Penryn Press Release. http://www.intel.com/pressroom/archive/releases/20070328fact.htm
10. The Journaling Flash File System. http://sources.redhat.com/jffs2/
11. Johnson, B. Facebook, personal communication (November 2008).
12. Katz, R.H. Tech titans building boom. *IEEE Spectrum* (February 2009). http://spectrum.ieee.org/green-tech/buildings/tech-titans-building-boom
13. Lamport, L. The part-time parliament. *ACM Trans. Comput. Syst., 16*, 2, (1998), 133–169.
14. Lim, K., Ranganathan, P., Chang, J., Patel, C., Mudge, T., Reinhardt, S. Understanding and designing new server architectures for emerging warehouse-computing environments. In *International Symposium on Computer Architecture (ISCA)* (Beijing, China, June 2008).
15. Nath, S., Gibbons, P.B. Online maintenance of very large random samples on flash storage. In *Proceedings of VLDB* (Auckland, New Zealand, August 2008).
16. Nath, S., Kansal, A. FlashDB: Dynamic self-tuning database for NAND flash. In *Proceedings of ACM/IEEE International Conference on Information Processing in Sensor Networks* (Cambridge, MA, April 2007).
17. Polte, M., Simsa, J., Gibson, G. Enabling enterprise solid state disks performance. In *Proceedings of the Workshop on Integrating Solid-State Memory into the Storage Hierarchy* (Washington, DC, March 2009).
18. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for Internet applications. August. 2001. http://portal.acm.org/citation.cfm?id=383071
19. Szalay, A., Bell, G., Terzis, A., White, A., Vandenberg, J. Low power Amdahl blades for data intensive computing, 2009. http://portal.acm.org/citation.cfm?id=1740407&dl=ACM
20. Tolia, N., Wang, Z., Marwah, M., Bash, C., Ranganathan, P., Zhu, X. Delivering energy proportionality with non energy-proportional systems—optimizing the ensemble. In *Proceedings of HotPower* (Palo Alto, CA, December 2008).
21. van Renesse, R. Schneider, F.B. Chain replication for supporting high throughput and availability. In *Proceedings of the 6th USENIX OSDI* (San Francisco, CA, December 2004).

David G. Andersen, **Jason Franklin**, **Amar Phanishayee**, **Lawrence Tan**, and **Vijay Vasudevan**, Carnegie Mellon University

Michael Kaminsky, Intel Labs

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965748

Technical Perspective
Is Scale Your Enemy, Or Is Scale Your Friend?

By John Ousterhout

ALTHOUGH THE NOMINAL topic of the following paper is managing crash reports from an installed software base, the paper's greatest contributions are its insights about managing large-scale systems. Kinshumann et al. describe how the Windows error reporting process became almost unmanageable as the scale of Windows deployment increased. They then show how an automated reporting and management system (Windows Error Reporting, or WER) not only eliminated the existing problems, but capitalized on the scale of the system to provide features that would not be possible at smaller scale. WER turned scale from enemy to friend.

Scale has been the single most important force driving changes in system software over the last decade, and this trend will probably continue for the next decade. The impact of scale is most obvious in the Web arena, where a single large application today can harness 1,000–10,000 times as many servers as the largest pre-Web applications of 10–20 years ago and supports 1,000 times as many users. However, scale also impacts developers outside the Web; in this paper, scale comes from the large installed base of Windows and the correspondingly large number of error reports emanating from the installed base.

Scale creates numerous problems for system developers and managers. Manual techniques that are sufficient at small scale become unworkable at large scale. Rare corner cases that are unnoticeable at small scale become common occurrences that impact overall system behavior at large scale. It would be easy to conclude that scale offers nothing to developers except an unending parade of problems to overcome.

Microsoft, like most companies, originally used an error reporting process with a significant manual component, but it gradually broke down as the scale of Windows deployment increased. As the number of Windows installation skyrocketed, so did the rate of error reports. In addition, the size and complexity of the Windows system increased, making it more difficult to track down problems. For example, a buggy third-party device driver could cause crashes that were difficult to distinguish from problems in the main kernel.

In reading this paper and observing other large-scale systems, I have noticed four common steps by which scale can be converted from enemy to friend. The first and most important step is automation: humans must be removed from the most important and common processes. In any system of sufficiently large scale, automation is not only necessary, but it is cheap: it's much less expensive to build tools than to manage a large system manually. WER automated the process of detecting errors, collecting information about them, and reporting that information back to Microsoft.

The second step in capitalizing on scale is to maintain records; this is usually easy once the processes have been automated. In the case of WER the data consists of information about each error, such as a stack trace. The authors developed mechanisms for categorizing errors into buckets, such that all the errors in a bucket probably share the same root cause. Accurate and complete data enables the third and fourth steps.

In any system of sufficiently large scale, automation is not only necessary, but it is cheap.

The third step is to use the data to make better decisions. At this point the scale of the system becomes an asset: the more data, the better. For example, WER analyzes error statistics to discover correlations with particular system configurations (a particular error might occur only when a particular device driver is present). WER also identifies the buckets with the most reports so they can be addressed first.

The fourth and final step is that processes change in fundamental ways to capitalize on the level of automation and data analysis. For example, WER allows a bug fix to be associated with a particular error bucket; when the same error is reported in the future, WER can offer the fix to the user at the time the error happens. This allows fixes to be disseminated much more rapidly, which is crucial in situations such as virus attacks.

Other systems besides WER are also taking advantage of scale. For example, Web search indexes initially kept independent caches of index data in the main memory of each server. As the number of servers increased they discovered that the sum total of all the caches was greater than the total amount of index data; by reorganizing their servers to eliminate duplication they were able to keep the entire index in DRAM. This enabled higher performance and new features. Another example is that many large-scale Web sites use an incremental release process to test new features on a small subset of users before exposing them to the full user base.

I hope you enjoy reading this paper, as I did, and that it will stimulate you to think about scale as an opportunity, not an obstacle. C

John Ousterhout (http://www.stanford.edu/~ouster) is Professor (Research) of CS at Stanford University.

© 2011 ACM 0001-0782/11/07 $10.00

DOI:10.1145/1965724.1965749

Debugging in the (Very) Large: Ten Years of Implementation and Experience

By Kinshuman Kinshumann, Kirk Glerum, Steve Greenberg, Gabriel Aul, Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt

Abstract

Windows Error Reporting (WER) is a distributed system that automates the processing of error reports coming from an installed base of a billion machines. WER has collected billions of error reports in 10 years of operation. It collects error data automatically and classifies errors into buckets, which are used to prioritize developer effort and report fixes to users. WER uses a progressive approach to data collection, which minimizes overhead for most reports yet allows developers to collect detailed information when needed. WER takes advantage of its scale to use error statistics as a tool in debugging; this allows developers to isolate bugs that cannot be found at smaller scale. WER has been designed for efficient operation at large scale: one pair of database servers records all the errors that occur on all Windows computers worldwide.

1. INTRODUCTION

Debugging a single program run by a single user on a single computer is a well-understood problem. It may be arduous, but follows general principles: a user reports an error, the programmer attaches a debugger to the running process or a core dump and examines program state to deduce where algorithms or state deviated from desired behavior. When tracking particularly onerous bugs the programmer can resort to restarting and stepping through execution with the user's data or providing the user with a version of the program instrumented to provide additional diagnostic information. Once the bug has been isolated, the programmer fixes the code and provides an updated program.[a]

Debugging in the large is harder. As the number of deployed Microsoft Windows and Microsoft Office systems scaled to tens of millions in the late 1990s, our programming teams struggled to scale with the volume and complexity of errors. Strategies that worked in the small, like asking programmers to triage individual error reports, failed. With hundreds of components, it became much harder to isolate the root causes of errors. Worse still, prioritizing error reports from millions of users became arbitrary and ad hoc.

In 1999, we realized we could completely change our model for debugging in the large, by combining two tools then under development into a new service called Windows Error Reporting (WER). The Windows team devised a tool to automatically diagnose a core dump from a system crash to determine the most likely cause of the crash and identify any known resolutions. Separately, the Office team devised a tool to automatically collect a stack trace with a small of subset of heap memory on an application failure and upload this *minidump* to servers at Microsoft. WER combines these tools to form a new system which automatically generates error reports from application and operating systems failures, reports them to Microsoft, and automatically diagnoses them to point users at possible resolutions and to aid programmers in debugging.

Beyond mere debugging from error reports, WER enables a new form of *statistics-based debugging*. WER gathers all error reports to a central database. In the large, programmers can mine the error report database to prioritize work, spot trends, and test hypotheses. Programmers use data from WER to prioritize debugging so that they fix the bugs that affect the most users, not just the bugs hit by the loudest customers. WER data also aids in correlating failures to co-located components. For example, WER can identify that a collection of seemingly unrelated crashes all contain the same likely culprit—say a device driver—even though its code was not running at the time of failure.

Three principles account for the use of WER by every Microsoft product team and by over 700 third-party companies to find thousands of bugs: *automated error diagnosis* and *progressive data collection*, which enable error processing at global scales, and *statistics-based debugging*, which harnesses that scale to help programmers more effectively improve system quality.

WER is not the first system to automate the collection of memory dumps. Postmortem debugging has existed since the dawn of digital computing. In 1951, The Whirlwind I system[2] dumped the contents of tube memory to a CRT in octal when a program crashed. An automated camera took a snapshot of the CRT on microfilm, delivered for debugging the following morning. Later systems dumped core to disk; used partial core dumps, which excluded shared code, to minimize the dump size[5]; and eventually used

A previous version of this paper appeared in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles* (SOSP '09).

[a] We use the following definitions: *error* (noun): a single event in which program behavior differs from that intended by the programmer; *bug* (noun): a root cause, in program code, that results in one or more errors.

telecommunication networks to deliver core dumps to the computer manufacturer.[4]

WER is the first system to provide automatic error diagnosis, the first to use progressive data collection to reduce overheads, and the first to automatically direct users to available fixes based on automated error diagnosis. WER remains unique in four aspects:

1. WER is the largest automated error-reporting system in existence. Approximately one billion computers run WER client code: every Windows system since Windows XP.
2. WER automates the collection of additional client-side data for hard-to-debug problems. When initial error reports provide insufficient data to debug a problem, programmers can request that WER collect more data in future error reports including: broader memory dumps, environment data, log files, and program settings.
3. WER automatically directs users to solutions for corrected errors. For example, 47% of kernel crash reports result in a direction to an appropriate software update or work around.
4. WER is general purpose. It is used for operating systems and applications, by Microsoft and non-Microsoft programmers. WER collects error reports for crashes, non-fatal assertion failures, hangs, setup failures, abnormal executions, and hardware failures.

2. PROBLEM, SCALE, AND STRATEGY

The goal of WER is to allow us to diagnose and correct every software error on every Windows system. We realized early on that scale presented both the primary obstacle and the primary solution to address the goals of WER. If we could remove humans from the critical path and scale the error reporting mechanism to admit millions of error reports, then we could use the law of large numbers to our advantage. For example, we did not need to collect all error reports, just a statistically significant sample. And we did not need to collect complete diagnostic samples for all occurrences of an error with the same root cause, just enough samples to diagnose the problem and suggest correlation. Moreover, once we had enough data to allow us to fix the most frequently occurring errors, then their occurrence would decrease, bringing the remaining errors to the forefront. Finally, even if we made some mistakes, such as incorrectly diagnosing two errors as having the same root cause, once we fixed the first then the occurrences of the second would reappear and dominate future samples.

Realizing the value of scale, five strategies emerged as necessary components to achieving sufficient scale to produce an effective system: automatic bucketing of error reports, collecting data progressively, minimizing human interaction, preserving user privacy, and directing users to solutions.

2.1. Automatic bucketing

WER automatically aggregates error reports likely originating from the same bug into a collection called a *bucket*.[b] If not, WER data naively collected with no filtering or organization, would absolutely overwhelm programmers. The ideal bucketing algorithm would map all error reports caused by the one bug into one unique bucket with no other bugs in that bucket. Because we know of no such algorithm, WER instead employs a set of bucketing heuristics in two phases. First, errors are *labeled*, assigned to a first bucket based on immediate evidence available at the client with the goal that each bucket contains error reports from just one bug. Second, errors are *classified* at the WER service; they are consolidated to new buckets as additional data is analyzed with the goal of minimizing programmer effort by placing error reports from just one bug into just one final bucket.

Bucketing enables automatic diagnosis and progressive data collection. Good bucketing relieves programmers and the system of the burden of processing redundant error reports, helps prioritize programmer effort by bucket prevalence, and can be used to link users to updates when the bugs has been fixed. In WER, bucketing is progressive. As additional data related to an error report is collected, such as symbolic information to translate from an offset in a module to a named function, the report is associated with a new bucket. Although the design of optimal bucketing algorithms remains an open problem, the bucketing algorithms used by WER are in practice quite effective.

2.2. Progressive data collection

WER uses a progressive data collection strategy to reduce the cost of error reporting so that the system can scale to high volume while providing sufficient detail for debugging. Most error reports consist of no more than a simple bucket identifier, which just increments its count. If additional data is needed, WER will next collect a *minidump* (an abbreviated stack and memory dump) and the configuration of the faulting system into a compressed cabinet archive file (the *CAB file*). If data beyond the minidump is required to diagnose the error, WER can progress to collecting full memory dumps, memory dumps from related programs, related files, or additional data queried from the reporting computer. Progressive data collection reduces the scale of incoming data enough that one pair of SQL servers can record every error on every Windows system worldwide. Progressive data collection also reduces the cost to users in time and bandwidth of reporting errors, thus encouraging user participation.

2.3. Minimizing human interaction

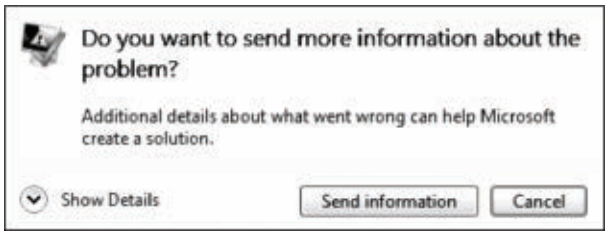
WER removes users from all but the authorization step of error reporting and removes programmers from initial error diagnosis. User interaction is reduced in most cases to a yes/no authorization (see Figure 1). Users may permanently opt in or out of future authorization requests. WER servers analyze each error report automatically to direct users to existing fixes, or, as needed, ask the client to collect additional data. Programmers are notified only after WER determines that a sufficient number of error reports have been collected for an unresolved bug.

2.4. Preserving user privacy

We take considerable care to avoid knowingly collecting personal identifying information (PII). This encourages user participation and reduces regulatory burden. For example,

[b] *bucket* (noun): a collection of error reports likely caused by the same bug; *bucket* (verb): to triage error reports into buckets.

Figure 1. Typical WER authorization dialog.



although WER collects hardware configuration information, client code zeros serial numbers, and other known unique identifiers to avoid transmitting data that might identify the sending computer. WER operates on an *informed consent* policy with users. Errors are reported only with user consent. All consent requests default to negative, thus requiring that the user opt-in before transmission. WER reporting can be disabled on a per-error, per-program, or per-computer basis by individual users or by administrators. Because WER does not have sufficient metadata to locate and filter possible PII from collected stack or heap data, we minimize the collection of heap data. Microsoft also enforces data-access policies that restrict the use of WER data strictly to debugging and improving program quality.

2.5. Providing solutions to users

Many errors have known corrections. For example, users running out-of-date software should install the latest service pack. The WER service maintains a mapping from buckets to *solutions*. A solution is the URL of a web page describing steps a user should take to prevent reoccurrence of the error. Solution URLs can link the user to a page hosting a patch for a specific problem, to an update site where users can get the latest version, or to documentation describing workarounds. Individual solutions can be applied to one or more buckets with a simple regular expression matching mechanism. For example, all users who hit *any* problem with the original release of Word 2003 are directed to a web page hosting the latest Office 2003 service pack.

3. BUCKETING ALGORITHMS

The most important element of WER is its mechanism for automatically assigning error reports to buckets. Conceptually WER bucketing heuristics can be divided along two axes. The first axis describes where the bucketing code runs: heuristics performed on client computers attempt to minimize the load on the WER servers and heuristics performed on servers attempt to minimize the load on programmers. The second axis describes the effect of the heuristic on the number of final buckets presented to programmers from a set of incoming error reports: *expanding* heuristics increase the number of buckets so that no two bugs are assigned to the same bucket; *condensing* heuristics decrease the number of buckets so that no two buckets contain error reports from the same bug. Working in concert, expanding and condensing heuristics should move WER toward the desired goal of a one-to-one mapping between bugs and buckets.

3.1. Client-side bucketing

When an error report is first generated, the client-side bucketing heuristics attempt to produce a unique bucket label using only local information; ideally a label likely to align with other reports caused by the same bug. The client-side heuristics are important because in most cases, the only data communicated to the WER servers will be a bucket label. An initial label contains the faulting program, module, and offset of the program counter within the module. Additional heuristics apply under special conditions, such as when an error is caused by a hung application. Programs can also apply custom client-side bucketing heuristics through the WER APIs.

Most client-side heuristics are expanding heuristics, intended to spread separate bugs into distinct buckets. For example, the `hang_wait_chain` heuristic starts from the program's user-input thread and walks the chain of threads waiting on synchronization objects held by other threads to find the source of the hang. The few client-side condensing heuristics were derived empirically for common cases where a single bug produces many buckets. For example, the `unloaded_module` heuristic condenses all errors where a module has been unloaded prematurely due to an application reference counting bug.

3.2. Server-side bucketing

Errors collected by WER clients are sent to the WER service. The heuristics for server-side bucketing attempt to classify error reports to maximize programmer effectiveness. While the current server-side code base includes over 500 heuristics, the most important heuristics execute in an algorithm that analyzes the memory dump to determine which thread context and stack frame most likely caused the error. The algorithm finds all thread context records in the memory dump. It assigns each stack frame a priority from 0 to 5 based on its increasing likelihood of being a root cause. The frame with the highest priority is selected. Priority 1 is used for core OS components, like the kernel, priority 2 for core device drivers, priority 3 for other OS code like the shell, and priority 4 for most other code. Priority 5, the highest priority, is reserved for frames known to trigger an error, such as a caller of `assert`. Priority 0, the lowest priority, is reserved for functions known never to be the root cause of an error, such as `memcpy`, `memset`, and `strcpy`.

WER contains a number of server-side heuristics to filter out error reports unlikely to be debuggable, such as applications executing corrupt binaries. Kernel dumps are placed into special buckets if they contain evidence of out-of-date device drivers, drivers known to corrupt the kernel heap, or hardware known to cause memory or computation errors.

4. STATISTICS-BASED DEBUGGING

Perhaps the most important feature enabled by WER is statistics-based debugging. With data from a sufficient percentage of all errors that occur on Windows systems worldwide, programmers can mine the WER database to prioritize debugging effort, find hidden causes, test root cause hypotheses, measure deployment of solutions, and monitor for regressions. The amount of data in the WER database is enormous, yielding opportunity for creative and useful queries.

Programmers sort their buckets and prioritize debugging effort on the buckets with largest volumes of error reports, thus helping the most users per unit of work. Often, programmers will aggregate error counts by function and then work through the buckets for the function in order of decreasing bucket count. This strategy tends to be effective as errors at different locations in the same function often have the same root cause.

The WER database can help find root causes which are not immediately obvious from memory dumps. For example, in one instance we received a large number of error reports with invalid pointer usage in the Windows event tracing infrastructure. An analysis of the error reports revealed that 96% of the faulting computers were running a specific third-party device driver. With well below 96% market share (based on all other error reports), we approached the vendor who found a memory corruption bug in their code. By comparing expected versus occurring frequency distributions, we similarly have found hidden causes from specific combinations of third-party drivers and from buggy hardware. A similar strategy is "stack sampling" in which error reports for similar buckets are sampled to determine which functions, other than the first target, occur frequently on the thread stacks.

WER can help test programmer hypotheses about the root causes of errors. The basic strategy is to construct a test function that can evaluate a hypothesis on a memory dump, and then apply it to thousands of memory dumps in the WER database to verify that the hypothesis is not violated. For example, a Windows programmer debugging an error related to a shared lock in the Windows I/O subsystem constructed a query to extract the current holder of the lock from a memory dump and then ran the expression across 10,000 memory dumps to see how many reports had the same lock holder. One outcome of the analysis was a bug fix; another was the creation of a new server-side heuristic.

The WER database can measure how widely a software update has been deployed. Deployment can be measured by absence, measuring the decrease in error reports fixed by the software update. Deployment can also be measured by an increased presence of the new program or module version in error reports for other issues.

The WER database can be used to monitor for regressions. Similar to the strategies for measuring deployment, we look at error report volumes over time to determine if a software fix had the desired effect of reducing errors. We also look at error report volumes around major software releases to quickly identify and resolve new errors that may appear with the new release.

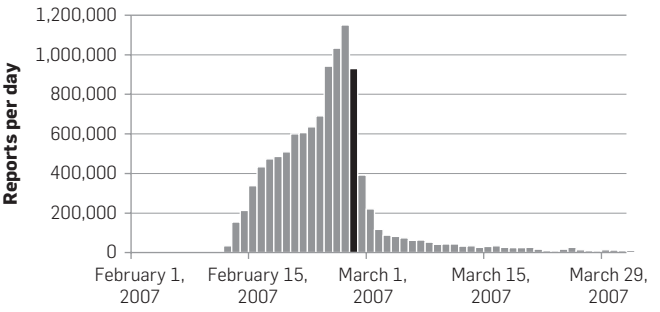
5. EVALUATION AND IMPACT

5.1. Scalability

WER collected its first million error reports within 8 months of its deployment in 1999. Since then, WER has collected billions more. The WER service employs approximately 60 servers provisioned to process well over 100 million error reports per day. From January 2003 to January 2009, the number of error reports processed by WER grew by a factor of 30.

The WER service is over provisioned to accommodate globally correlated events. For example, in February 2007, users of Windows Vista were attacked by the Renos Malware. If installed on a client, Renos caused the Windows GUI shell, `explorer.exe`, to crash when it tried to draw the desktop. A user's experience of a Renos infection was a continuous loop in which the shell started, crashed, and restarted. While a Renos-infected system was useless to a user, the system booted far enough to allow reporting the error to WER—on computers where automatic error reporting was enabled—and to receive updates from Windows Update (WU).

Figure 2. Renos Malware: Number of error reports per day. Black bar shows when a fix was released through WU.



As Figure 2 shows, the number of error reports from systems infected with Renos rapidly climbed from 0 to almost 1.2 million per day. On February 27, shown in black in the graph, Microsoft released a Windows Defender signature for the Renos infection via WU. Within 3 days enough systems had received the new signature to drop reports to under 100,000 per day. Reports for the original Renos variant became insignificant by the end of March. The number of computers reporting errors was relatively small: a single computer (somehow) reported 27,000 errors, but stopped after being automatically updated.

5.2. Finding bugs

WER augments, but does not replace, other methods for improving software quality. We continue to apply static analysis and model-checking tools to find errors early in the development process.[1] These tools are followed by extensive testing regimes before releasing software to users. WER helps us to rank all bugs and to find bugs not exposed through other techniques. The Windows Vista programmers fixed 5000 bugs found by WER in beta deployments *after* extensive static analysis, but *before* product release.

Compared to errors reported directly by humans, WER reports are more useful to programmers. Analyzing data sets from Windows, SQL, Excel, Outlook, PowerPoint, Word, and Internet Explorer, we found that a bug reported by WER is 4.5–5.1 times more likely to be fixed than a bug reported directly by a human. This is because error reports from WER document internal computation state whereas error reports from humans document external symptoms.

Given finite programmer resources, WER helps focus effort on the bugs that have the biggest impact on the most users. Our experience across many application and OS releases is that error reports follow a Pareto distribution with a small number of bugs accounting for most error reports. As an example, the graphs in Figure 3 plot

Figure 3. Relative number of reports per bucket and CDF for top 20 buckets from Office 2010 ITP. Black bars are buckets for bugs fixed in three-week sample period.

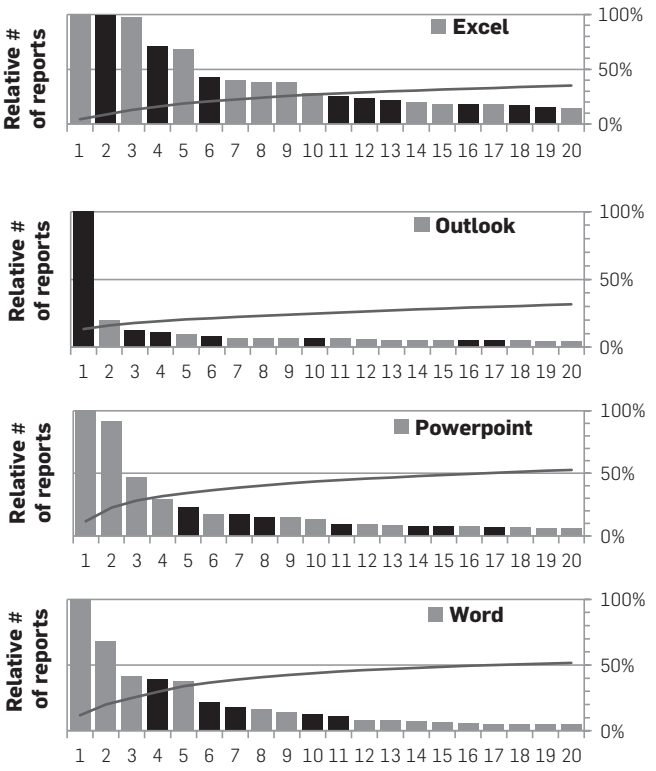
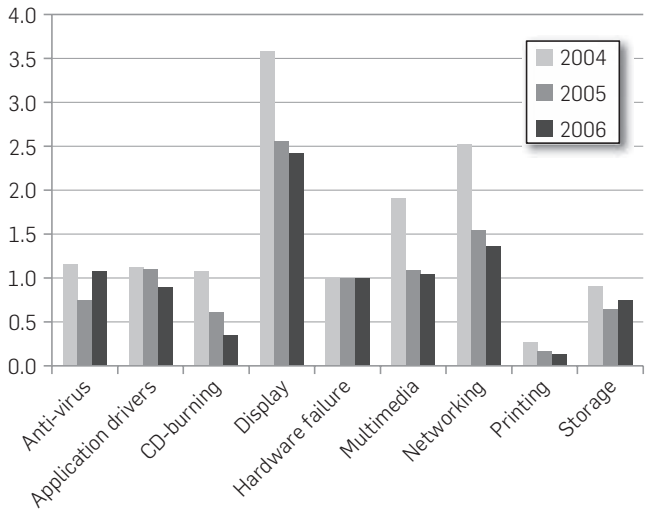


Figure 4. Crashes by driver class normalized to hardware failures for same period.



the relative occurrence and cumulative distribution functions (CDFs) for the top 20 buckets of programs from the Microsoft Office 2010 internal technical preview (ITP). The top 20 bugs account for 30%–50% of all error reports. The goal of the ITP was to find and fix as many bugs as possible using WER before releasing a technical preview to customers. These graphs capture the team's progress just 3 weeks into the ITP. The ITP had been installed by 9000 internal users, error reports had been collected, and the programmers had already fixed bugs responsible for over 22% of the error reports. The team would work for another 3 weeks collecting error reports and fixing bugs, before releasing a technical preview to customers.

An informal historical analysis indicates that WER has helped improve the quality of many classes of third-party kernel code for Windows. Figure 4 plots the frequency of system crashes for various classes of kernel drivers for systems running Windows XP in March 2004, March 2005, and March 2006, normalized against system crashes caused by hardware failures in the same period. Assuming that the expected frequency of hardware failures remained roughly constant over that time period (something we cannot yet prove), the number of system crashes for kernel drivers has gone down every year except for two classes of drivers: anti-virus and storage.

As software providers begin to use WER more proactively, their error report incidences decline dramatically. For example, in May 2007, one kernel-mode driver vendor began to use WER for the first time. In 30 days the vendor addressed the top 20 reported issues for their code. Within 5 months, as WER directed users to pick up fixes, the percentage of all kernel crashes attributed to the vendor dropped from 7.6% to 3.8%.

5.3. Bucketing effectiveness

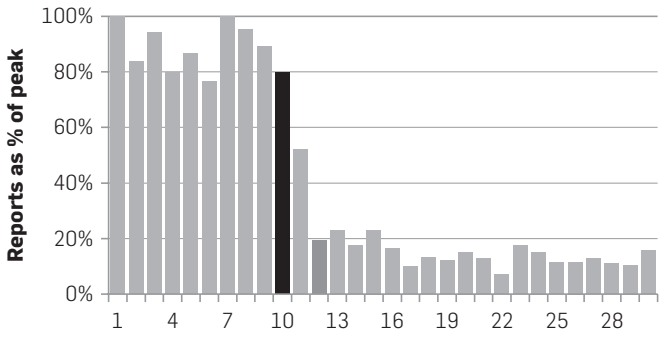
We know of two forms of weakness in the WER bucketing heuristics: weaknesses in the condensing heuristics, which result in mapping reports from a bug into too many buckets, and weaknesses in the expanding heuristics, which result in mapping more than one bug into the same bucket. An analysis of error reports from the Microsoft Office 2010 ITP shows that as many as 37% of these errors reports may be incorrectly bucketed due to poor condensing heuristics. An analysis of all kernel crashes collected in 2008 shows that as many as 14% of these error reports were incorrectly bucketed due to poor expanding heuristics.

While not ideal, WER's bucketing heuristics are in practice effective in identifying and quantifying the occurrence of errors caused by bugs in both software and hardware. In 2007, WER began receiving crash reports from computers with a particular processor. The error reports were easily bucketed based on an increase in system machine checks and processor type. When Microsoft approached the processor vendor, the vendor had already discovered and documented externally the processor issue, but had no idea it could occur so frequently until presented with WER data. The vendor immediately released a microcode fix via WU—on day 10, the black bar in Figure 5—and within 2 days, the number of error reports had dropped to just 20% of peak.

6. CONCLUSION

WER has changed the process of software development at Microsoft. Development has become more empirical, more immediate, and more user-focused. Microsoft teams use WER to catch bugs after release, but perhaps as importantly, we use WER during internal and beta pre-release

Figure 5. Crashes/day for a firmware bug. Patch was released via WU on day 10.



deployments. While WER does not make debugging in the small significantly easier (other than perhaps providing programmers with better analysis of core dumps), WER has enabled a new class of debugging in the large. The statistics collected by WER help us to prioritize valued programmer resources, understand error trends, and find correlated errors.

WER's progressive data collection strategy means that programmers get the data they need to debug issues, in the large and in the small, while minimizing the cost of data collection to users. Automated error analysis ensures programmers are not distracted with previously diagnosed errors. It also ensures that users are made aware of fixes that can immediately improve their computing experience.

As applied to WER, the law of large numbers says that we will eventually collect sufficient data to diagnose even rare Heisenbugs[3]; WER has already helped identify such bugs dating back to the original Windows NT kernel.

WER is the first system to provide users with an end-to-end solution for reporting and recovering from errors. WER provides programmers with real-time data about errors actually experienced by users and provides them with an incomparable billion-computer feedback loop to improve software quality. C

References

1. Bush, W.R., Pincus, J.D., Sielaff, D.J. A static analyzer for finding dynamic programming errors. *Softw. Pract. Exp. 30* (5) (2000), 775–802.
2. Everett, R.R. The Whirlwind I computer. In *Proceedings of the 1951 Joint AIEE–IRE Computer Conference* (Philadelphia, PA), 1951.
3. Gray, J. Why do computers stop and what can we do about it. In *Proceedings of the 6th International Conference on Reliability and Distributed Databases*, 1986, 3–12.
4. Lee, I., Iyer, R.K. Faults, symptoms, and software fault tolerance in the tandem GUARDIAN90 operating system. In *Digest of Papers of the Twenty-Third International Symposium on Fault-Tolerant Computing (FTCS-23)*. IEEE, Toulouse, France, 1993.
5. Walter, E.S., Wallace, V.L. Further analysis of a computing center environment. *Commun. ACM 10* (5) (1967), 266–272.

Kinshuman Kinshumann, Kirk Glerum, Steve Greenberg, Gabriel Aul, Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt
Microsoft Corporation.

© 2011 ACM 0001-0782/11/07 $10.00

Take Advantage of ACM's Lifetime Membership Plan!

- ACM Professional Members can enjoy the convenience of making a single payment for their entire tenure as an ACM Member, and also be protected from future price increases by taking advantage of ACM's Lifetime Membership option.
- ACM Lifetime Membership dues may be tax deductible under certain circumstances, so becoming a Lifetime Member can have additional advantages if you act before the end of 2011. (Please consult with your tax advisor.)
- Lifetime Members receive a certificate of recognition suitable for framing, and enjoy all of the benefits of ACM Professional Membership.

Learn more and apply at:
http://www.acm.org/life

Association for Computing Machinery
Advancing Computing as a Science & Profession

ACM Transactions on Reconfigurable Technology and Systems



This quarterly publication is a peer-reviewed and archival journal that covers reconfigurable technology, systems, and applications on reconfigurable computers. Topics include all levels of reconfigurable system abstractions and all aspects of reconfigurable technology including platforms, programming environments and application successes.

www.acm.org/trets
www.acm.org/subscribe

Association for Computing Machinery

The Microsoft Research – University of Trento COSBI

Principal Investigator in computer science

We are seeking an outstanding, highly motivated and experienced Principal Investigator in modeling and simulation.

PhD in Computer Science and at least eight years experience of research activity in this field are required.

Skills and experience:

- international reputation, distinguished record of scientific results, large number of invited talks, high bibliometric indexes, participation in steering committees, program committees and advisory bodies;
- scientific and administrative coordination of large international projects;
- demonstrated ability to establish strong industrial collaborations, start and develop funded research programs;
- consulting experience for industries;
- strong interpersonal and research leadership, vision of the future development of biological and ecological systems modeling;
- demonstrated ability to manage stressing situations and delivering results in time according to pre-defined schedules;
- experience as CS professional is preferred.

We offer a competitive salary. This job is in Northern Italy. Other vacancies at www.cosbi.eu/index.php/open-positions

Send your application letter with statement of research interests, CV and contact information for three references to:

COSBI HR Department • Trento – Italy • hr@cosbi.eu; Ph: +39 0461 282811 • Ref: Principal Investigator in Computer Science

SYMPTOMS:
DEVELOPER SUFFERING FROM BORING PROJECTS, DATED TECHNOLOGIES AND A STAGNANT CAREER.

CURE:
START YOUR NEW CAREER AT BERICO

If you are a skilled Software Engineer with passion and expertise in any of the following areas, we invite you to apply.

- Cloud Computing
- Web Development
- Application Development
- Mobile Application Development

To learn more about Berico and our career opportunities, please visit **www.bericotechnologies.com**

or email your resume to **recruiting@bericotechnologies.com**

BERICO TECHNOLOGIES™
BE SMARTER. BE FASTER.

Ada Core Technologies
Sr. QA & Release Engineer
(New York, NY)

Coordinate the release process for software products using GNAT Pro compiler & Ada programming language. Manage the QA process of the GNAT Pro compiler on UNIX platforms. Develop enhancements to internal infrastructure. Send resumes to Richard Kenner, VP, Ada Core Technologies, Inc, 104 Fifth Ave, 15th Fl, New York, NY 10011. No calls, faxes or emails please! EOE.

Maharishi University of Management
Computer Science Department
Assistant Professor

The Computer Science Department at Maharishi University of Management invites applications for a full-time Assistant Professor position beginning Fall 2011. Qualifications include Ph.D. in Computer Science (or closely related area), or M.S. and seven years of professional software development experience. Highly qualified candidates will be considered for Associate Professor.

The primary responsibility is teaching computer science courses. Participation in scholarly research and publication is also expected. Candidates with a demonstrated potential for acquiring external research funding and/or significant professional software development experience will be given priority. Applications will be reviewed as they are received until the position is filled. To apply, email curriculum vitae (pdf file) to cssearch2011@mum.edu.

For further information, see http://www.mum.edu/ and http://mscs.mum.edu/. MUM is located in Fairfield, Iowa, and is an equal opportunity employer.

ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to *acmmediasales@acm.org*. Please include text, and indicate the issue/or issues where the ad will appear, and a contact name and number.
Estimates: An insertion order will then be e-mailed back to you. The ad will by typeset according to CACM guidelines. NO PROOFS can be sent. Classified line ads are NOT commissionable.
Rates: $325.00 for six lines of text, 40 characters per line. $32.50 for each additional line after the first six. The MINIMUM is six lines.
Deadlines: 20th of the month/2 months prior to issue date. For latest deadline info, please contact: *acmmediasales@acm.org*
Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at: *http://jobs.acm.org*

Ads are listed for a period of 30 days.

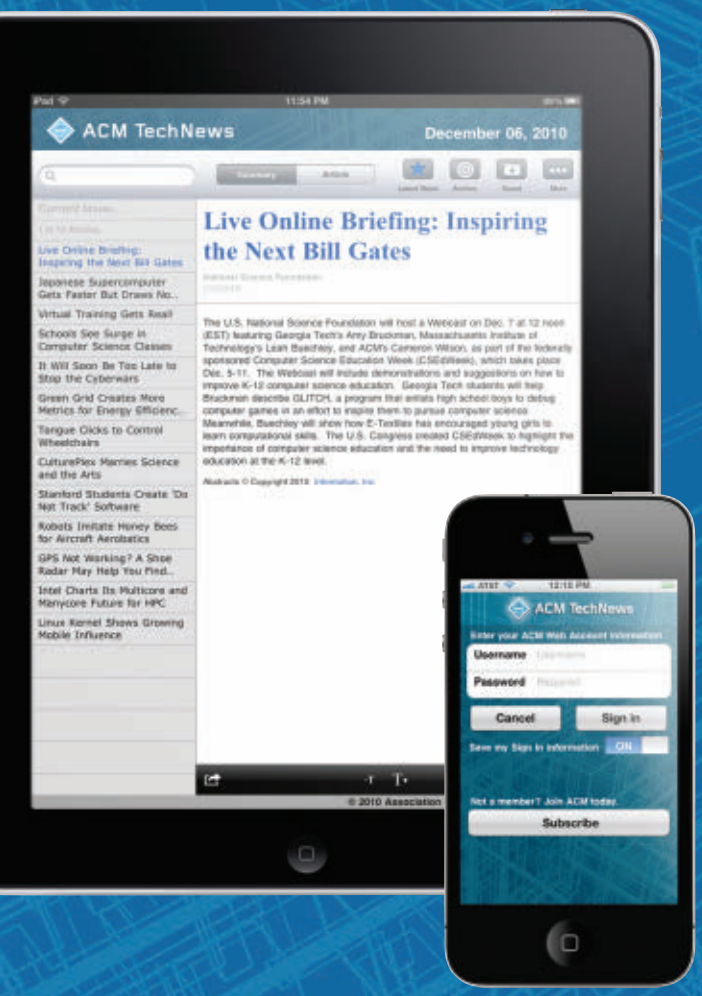
For More Information Contact:
ACM Media Sales
at 212-626-0686 or
acmmediasales@acm.org

ACM TechNews Goes Mobile

iPhone & iPad Apps Now Available in the iTunes Store

ACM TechNews—ACM's popular thrice-weekly news briefing service—is now available as an easy to use mobile apps downloadable from the Apple iTunes Store.

These new apps allow nearly 100,000 ACM members to keep current with news, trends, and timely information impacting the global IT and Computing communities each day.



TechNews mobile app users will enjoy:

- ***Latest News:*** Concise summaries of the most relevant news impacting the computing world
- ***Original Sources:*** Links to the full-length articles published in over 3,000 news sources
- ***Archive access:*** Access to the complete archive of TechNews issues dating back to the first issue published in December 1999
- ***Article Sharing:*** The ability to share news with friends and colleagues via email, text messaging, and popular social networking sites
- ***Touch Screen Navigation:*** Find news articles quickly and easily with a streamlined, fingertip scroll bar
- ***Search:*** Simple search the entire TechNews archive by keyword, author, or title
- ***Save:*** One-click saving of latest news or archived summaries in a personal binder for easy access
- ***Automatic Updates:*** By entering and saving your ACM Web Account login information, the apps will automatically update with the latest issues of TechNews published every Monday, Wednesday, and Friday

The Apps are freely available to download from the Apple iTunes Store, but users must be registered individual members of ACM with valid Web Accounts to receive regularly updated content.

http://www.apple.com/iphone/apps-for-iphone/ **http://www.apple.com/ipad/apps-for-ipad/**

ACM TechNews

ACM LAUNCHES
ENHANCED DIGITAL LIBRARY



The new DL simplifies usability, extends connections, and expands content with:

- Broadened citation pages with tabs for metadata and links to expand exploration and discovery
- Redesigned binders to create personal, annotatable reading lists for sharing and exporting
- Enhanced interactivity tools to retrieve data, promote user engagement, and introduce user-contributed content
- Expanded table-of-contents service for all publications in the DL

Visit the ACM Digital Library at:

dl.acm.org



Association for Computing Machinery

Advancing Computing as a Science & Profession

[CONTINUED FROM P. 120] one of these slime-mold cells. They *like* reverse Polish. I'm overwriting their junk DNA."

"We prefer to speak of sequences that code for obsolete or unactivated functional activity," said Velma, making a playful professor face.

"Like Harry's sense of empathy?" I suggested.

Velma laughed. "I'm waiting for him to code me into the slime mold with him."

A week later, Harry was having conversations out loud with the mold culture on his desk. Intrigued by the activity, one of our techs had interfaced a sound card to Harry's culture, still in its Petri dish. When Harry was talking to it, I couldn't readily tell which of the voices was the real him.

The week after that, I noticed the slime mold colonies had formed themselves into a pattern of nested scrolls, with fruiting bodies atop some of the ridges. Velma was in the office a lot, excitedly discussing a joint paper she was writing with Harry.

"Not exactly a wedding," I joked. "But *still*."

When Velma left, Harry gave me a frown. "You don't ever plan to get on my wavelength, do you, Fletch? You'll always be picking at me."

"So? Not everyone has to be the same."

"By now I would have thought you'd want to join me. You're the younger man. I need for you to extend my research." He was leaning over his desk, lifting up the bell jar to fiddle with his culture.

"I've got my own career," I said, shaking my head. "But, of course, I admit there's genius in your work."

"Your work now," said Harry. "*Yours*." He darted forward and blew a puff of spores into my face. In moments the mold had reprogrammed my wetware. I became a full-on emulation of Harry.

And—I swear—Velma will soon be mine. C

Rudy Rucker (rudy@rudyrucker.com) is a professor emeritus in the CS Department at San Jose State University, San Jose, CA, and author of pop math and CS books, including *The Lifebox, the Seashell and the Soul*. He is also a science fiction writer known for his recent *Postsingular* and cyberpunk *Ware Tetralogy*, which won two Philip K. Dick awards. His autobiography *Nested Scrolls* will appear in late 2011.

© 2011 ACM 0001-0782/11/07 $10.00

Future Tense, one of the revolving features on this page, presents stories and essays from the intersection of computational science and technological speculation, their boundaries limited only by our ability to imagine what will and could be.

DOI:10.1145/1965724.1965750

Rudy Rucker

Future Tense
My Office Mate

I became a biocomputational zombie for science...and for love.

YOU'D BE SURPRISED what poor equipment the profs have in our CS department. Until quite recently, my office mate Harry's computer was a primeval beige box lurking beneath his desk. Moreover, it had taken to making an irritating whine, and the techs didn't want to bother with it.

One rainy Tuesday during his office hour, Harry snapped. He interrupted a conversation with an earnest student by jumping to his feet, yelling a curse, and savagely kicking the computer. The whine stopped; the machine was dead. Frightened and bewildered, the student left.

"*Now* they'll have to replace this clunker," said Harry. "And you keep your trap shut, Fletcher."

"What if the student talks?"

"Nobody listens to them."

In a few days, a new computer appeared on Harry's desk, an elegant new model the size of a sandwich, with a wafer-thin display propped up like a portrait frame.

Although my office mate is a brilliant man, he's a thumb-fingered klutz. For firmly held reasons of principle, he wanted to tweak the settings of his lovely new machine to make it use a reverse Polish notation command-line interface; this had to do with the massive digital archiving project on which he was forever working. The new machine demurred at adopting reverse Polish. Harry downloaded some freeware patches, intending to teach the device a lesson. You can guess how that worked out.

The techs took Harry's dead sandwich back to their lair, wiped its memory, and reinstalled the operating system. Once again its peppy screen shone atop his desk. But now Harry sulked, not wanting to use it.



"This is about my soul," he told me. "I've spent, what, 30 years creating a software replica of myself. Everything I've written: my email messages, my photos, and a lot of my conversations—

"My entire wetware database is flowing into every one of these slime mold cells. They *like* reverse Polish."

and, yes, I'm taping this, Fletcher. A rich compost of Harry data. It's ready to germinate, ready to come to life. But these brittle machines thwart my immortality at every turn."

"You'd just be modeling yourself as a super chatbot, Harry. In the real world, we all die." I paused, thinking about Harry's attractive woman friend of many years. "It's a shame you never married Velma. You two could have had kids. Biology is the easy path to self-replication."

"You're not married either," said Harry, glaring at me. "And Velma says what you said, too." As if reaching a momentous decision, he snatched the shapely sandwich computer off his desk and put it on mine. "Very well then! I'll make my desk into a stinky bio farm."

Sure enough, when I came into the office on Monday, I found Harry's desk encumbered with a small biological laboratory. Harry and his woman friend Velma were leaning over it, fitting a data cable into a socket in the side of a Petri dish that sat beneath a bell jar.

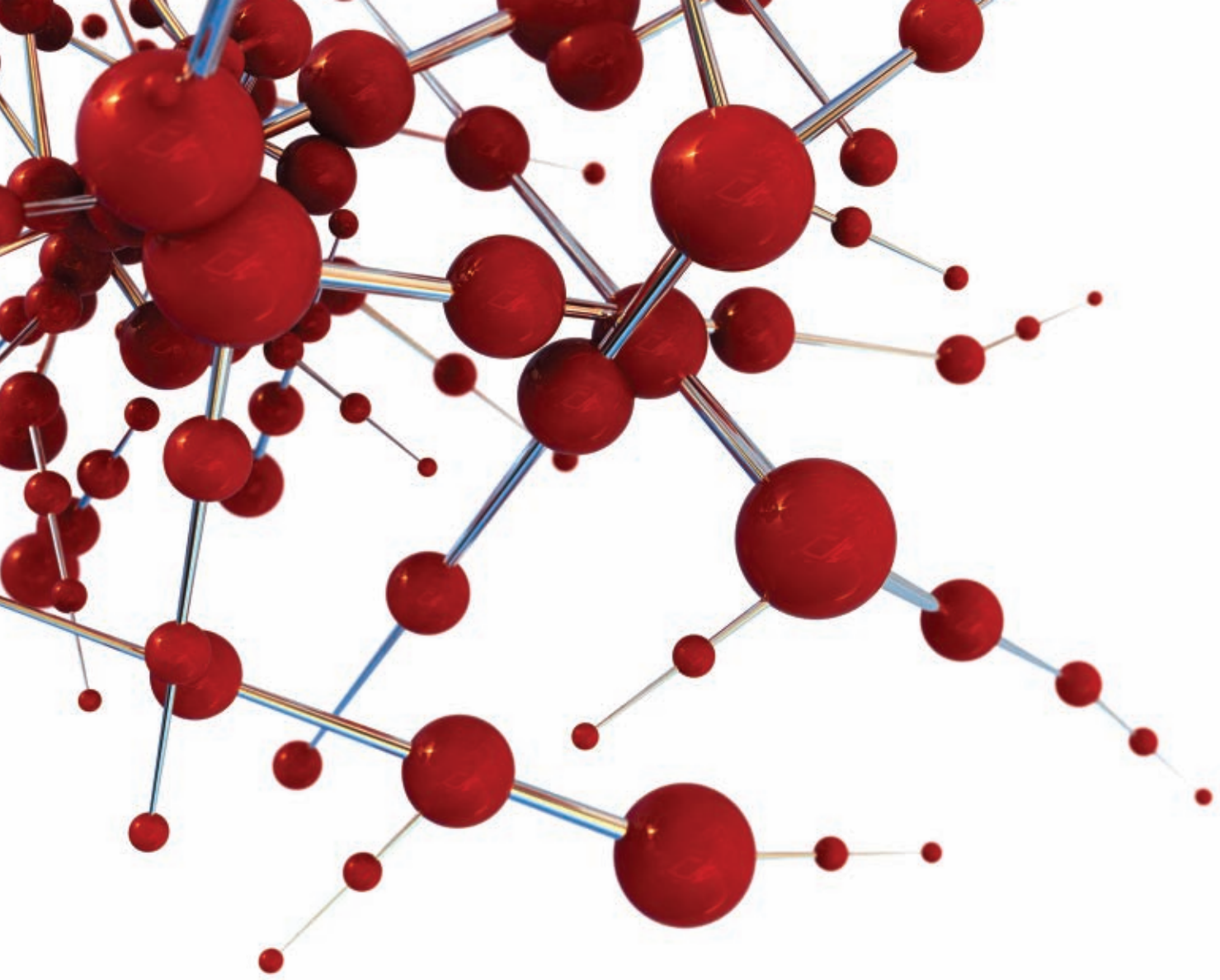
"Hi Fletch," said Velma brightly. She was a terminally cheerful genomics professor with curly hair. "Harry wants me to help him reproduce as a slime mold."

"How romantic," I said. "Do you think it'll work?"

"Biocomputation has blossomed this year," said Velma. "The Durban-Krush mitochondrial protocols have solved our input/output problems."

"A cell's as much a universal computer as any of our department's junkboxes," put in Harry. "And just look at this! My entire wetware database is flowing into every [CONTINUED ON P. 119]

PHOTOGRAPH BY FLICKR USER DOTLIZARD



CONNECT WITH OUR COMMUNITY OF EXPERTS.

www.reviews.com

Association for Computing Machinery

Reviews.com

They'll help you find the best new books and articles in computing.

Computing Reviews is a collaboration between the ACM and Reviews.com.

INTERACTIVE TABLETOPS AND SURFACES



november 13 - 16 2011

ACM ITS 2011

ACM ITS is the premiere venue for presenting research in the design and use of new and emerging tabletop and interactive surface technologies.

ITS 2011 will feature:

- Paper & Notes
- Demos
- Poster
- Doctoral Symposium
- Tutorials / Workshops

Conference Chairs

Jun Rekimoto
Hideki, Koike
Kentaro Fukuchi

Program Chairs

Yoshifumi Kitamura
Daniel Wigdor

THE ACM INTERNATIONAL CONFERENCE ON INTERACTIVE TABLETOPS AND SURFACES

www.its2011.jp