

Innovative Technology for Computer Professionals Computer

OCTOBER 2011

<http://www.computer.org>

SOFTWARE ENGINEERING MEETS ...

OBJECT DIGITIZATION, P. 81

TWITTER MOOD AS A STOCK MARKET PREDICTOR, P. 91

DIGITAL MACHINERY AND ANALOG BRAINS, P. 100

 **IEEE**

 IEEE
computer society

ACCEPT THE NAVY CHALLENGE

Become a member of an elite research and development community involved in basic and applied scientific research and advanced technological development for tomorrow's Navy.

NAVAL RESEARCH LABORATORY

Senior Scientist for Advanced Computing Concepts

ST-1310, \$119,554 to \$179,700* per annum

*Rate limited to the rate for level III of the Executive Schedule (5U.S.C. 5304(g)(2))

Serves as the technical expert in the diverse areas of high performance computing, networking, and storage. Applicants should be recognized as national/international authorities and have demonstrated the scientific vision and organizational skills necessary to bring long term, multi-faceted research programs to successful completion.

As a distinguished scientist and recognized leader, the incumbent will provide vision and technical direction to research efforts in highly integrated and optimized massively parallel computing technology, high performance network technology, and massive storage technology -- including prototype and proof of concept systems. The incumbent must have expertise in all three of these technology areas and application expertise with respect to Department of Defense memory- and speed-intensive computational problems. Specific duties include:

- Developing advanced concepts that will directly improve the ability of the US Department of Defense (DoD) and Intelligence Community (IC) to push computer technology limits.
- Providing technical oversight to a small, talented, highly-motivated research team to push the envelope of high-performance computing (10's to 100's TeraFLOPS with scaling to PetaFLOPS), high-performance networking (100's Giga-bps to Tera-bps), distributed storage and global file systems (100's Petabytes to Exabytes), and advanced visualization and graphics (from handheld PDAs to graphics walls), with data sets range from small transaction sizes, to multigigabyte sizes and very large video and related realtime streams.
- Support and accommodate information system security technologies from data-at-rest to very high speed national security link encryption, as appropriate.
- Briefing DoD senior officials regarding Laboratory research efforts in the above areas; serving as liaison among NRL, the Navy, and other national and international organizations; and consulting on important scientific and programmatic issues.

This position offers enormous potential for advancing the state of the art with respect to high performance computing, networking, and storage technology, and for applying that technology to improve national security. Examples of past accomplishments of this position include:

- Extending High Performance Computing, Networking, and Storage technologies. Examples include NRL's on-going "Large Data" project, which provides Petabyte storage and both tactical and 10-Gbps access to DoD and IC clients across the globe, and NRL's winning the bandwidth challenge at Super Computing 2009.
- Working with academia, industry, and other government in the early 2000's and late 1990's to develop a precursor to Google Earth and to develop and prove out the first progressive HDTV cameras with partners like ABC and Disney to make HDTV progressive imagery the standard for the DoD and IC.
- Developing in the mid-1990's the Joint Broadcast System, a precursor to the Global Broadcast System, to return high-bandwidth data from theater to CONUS.
- Driving industry, DoD, and IC adoption of ATM technology in the early 1990's for core networks and assisting in the development and testing of high-speed Type-1 cryptographic devices to drive dramatic increases in DoD netcentric capabilities.

Because of the sensitivity of some of the research application areas the incumbent must be eligible for TS-SCI security clearance.

For information regarding this vacancy and specific instructions on how to apply, go to www.usajobs.gov, log in and enter the following announcement number: NW1-XXXX-00-K9138767-FL. Please carefully read the announcement and follow instructions when applying. The announcement closes on 10/31/2011. Please contact Ginger Kisamore at ginger.kisamore@nrl.navy.mil for more information.



Navy is an Equal Opportunity Employer

Innovative Technology for Computer Professionals

Computer

Editor in Chief

Ron Vetter
University of North Carolina
Wilmington
vetterr@uncw.edu

Associate Editor in Chief

Sumi Helal
University of Florida
helal@cise.ufl.edu

Associate Editor in Chief, Research Features

Kathleen Swigger
University of North Texas
kathy@cs.unt.edu

Associate Editor in Chief, Special Issues

Bill N. Schilit
Google
schilit@computer.org

Computing Practices

Rohit Kapur
Synopsis
rohit.kapur@synopsys.com

Perspectives

Bob Colwell
bob.colwell@comcast.net

Web/Multimedia Editor

Charles R. Severance
csev@umich.edu

2011 IEEE Computer Society President

Sorel Reisman
s.reisman@computer.org

Area Editors**Computer Architectures**

Tom Conte
Georgia Tech
Steven K. Reinhardt
AMD

Distributed Systems

Jean Bacon
University of Cambridge

Graphics and Multimedia

Oliver Bimber
Johannes Kepler University Linz

High-Performance Computing

Vladimir Getov
University of Westminster

Information and Data Management

Naren Ramakrishnan
Virginia Tech

Multimedia

Savitha Srinivasan
IBM Almaden Research Center

Networking

Ahmed Helmy
University of Florida

Security and Privacy

Rolf Oppliger
eSECURITY Technologies

Software

Robert B. France
Colorado State University

David M. Weiss

Iowa State University

Web Engineering

Simon Shim
San Jose State University

Column Editors**Discovery Analytics**

Naren Ramakrishnan
Virginia Tech

Education

Ann E.K. Sobel
Miami University

Entertainment Computing

Kelvin Sung
University of Washington, Bothell

Green IT

Kirk W. Cameron
Virginia Tech

Identity Sciences

Karl Ricanek
University of North Carolina, Wilmington

In Development

Chris Huntley
Fairfield University

Industry Perspective

Sumi Helal
University of Florida

Invisible Computing

Albrecht Schmidt
University of Stuttgart

The Known World

David A. Grier
George Washington University

The Profession

Neville Holmes
University of Tasmania

Security

Jeffrey M. Voas
NIST

Social Computing

John Riedl
University of Minnesota
Software Technologies
Mike Hinchey
Lero—the Irish Software Engineering Research Centre

Advisory Panel

Carl K. Chang
Editor in Chief Emeritus
Iowa State University
Hal Berghel
University of Nevada, Las Vegas
Doris L. Carver
Louisiana State University
Ralph Cavin
Semiconductor Research Corp.
Rick Mathieu
James Madison University
Naren Ramakrishnan
Virginia Tech
Theresa-Marie Rhyne
Consultant
Alf Weaver
University of Virginia

Publications Board

David A. Grier (chair),
Alain April, **David Bader**,
Angela R. Burgess, **Jim Cortada**,
Hakan Erdogmus, **Frank E. Ferrante**,
Jean-Luc Gaudiot, **Paolo Montuschi**,
Dorée Duncan Seligmann,
Linda I. Shafer, **Steve Tanimoto**,
George Thiruvathukal

Magazine Operations Committee

Dorée Duncan Seligmann (chair),
Erik R. Altman, **Isabel Beichl**,
Krishnendu Chakrabarty, **Nigel Davies**,
Simon Liu, **Dejan Milojević**,
Michael Rabinovich, **Forrest Shull**,
John R. Smith, **Gabriel Taubin**,
Ron Vetter, **John Viega**,
Fei-Yue Wang, **Jeffrey R. Yost**

Editorial Staff

Judith Prow
Managing Editor
jprow@computer.org
Chris Nelson
Senior Editor

Contributing Editors

Camber Agrelius
Lee Garber
Bob Ward

Design and Production

Larry Bauer
Design
Olga D'Astoli
Cover Design
Kate Wojogbe
Jennie Zhu

Administrative Staff

Products and Services Director
Evan Butterfield
Senior Manager, Editorial Services
Lars Jentsch

Manager, Editorial Services

Jennifer Stout
Senior Business Development Manager
Sandy Brown
Senior Advertising Coordinator
Marian Anderson

Circulation: *Computer* (ISSN 0018-9162) is published monthly by the IEEE Computer Society. **IEEE Headquarters**, Three Park Avenue, 17th Floor, New York, NY 10016-5997; **IEEE Computer Society Publications Office**, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314; voice +1 714 821 8380; fax +1 714 821 4010; **IEEE Computer Society Headquarters**, 2001 L Street NW, Suite 700, Washington, DC 20036. IEEE Computer Society membership includes \$19 for a subscription to *Computer* magazine. Nonmember subscription rate available upon request. Single-copy prices: members \$20; nonmembers \$99.

Postmaster: Send undelivered copies and address changes to *Computer*, IEEE Membership Processing Dept., 445 Hoes Lane, Piscataway, NJ 08855. Periodicals Postage Paid at New York, New York, and at additional mailing offices. Canadian GST #125634188. Canada Post Corporation (Canadian distribution) publications mail agreement number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8 Canada. Printed in USA.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *Computer* does not necessarily constitute endorsement by the IEEE or the Computer Society. All submissions are subject to editing for style, clarity, and space.

Innovative Technology for Computer Professionals

Computer

www.computer.org/computer

CONTENTS



ABOUT THIS ISSUE

Authors who are well known in the software engineering community have contributed articles to this special issue to explain the interactions between SE and a very fundamental issue—the theory that sustains SE study—as well as other fields, including open source software, evolutionary computation, space exploration, and services and cloud computing.

COVER FEATURES

17 GUEST EDITORS' INTRODUCTION Where Software Engineering Meets ...

Carl K. Chang, David M. Weiss, and Mike Hinchey

Because software engineering is a relatively young field, research in the discipline benefits from the domain knowledge and wisdom accumulated outside its specific realm.

19 Can Practitioners Neglect Theory and Theoreticians Neglect Practice?

Manfred Broy

Theory helps engineers in other disciplines create and understand methods, evaluate results, and optimize processes. Does it have a role in software engineering as well?

25 Open Source Software: Lessons from and for Software Engineering

Brian Fitzgerald

Despite initial suggestions to the contrary, open source software projects exhibit many of the fundamental tenets of software engineering. Likewise, the existence of category-killer apps suggests that conventional software engineering can draw some lessons from OSS.

31 Software Engineering Meets Evolutionary Computation

Mark Harman

The concept of evolutionary computation has affected virtually every area of software design, not merely as a metaphor, but as a realistic algorithm for exploration, insight, and improvement.

41 Software Engineering for Space Exploration

Robyn Lutz

Software engineering offers tools and techniques that improve the odds spacecraft will survive long missions, contributing to their resilience to new environmental challenges and to their adaptability to updated mission requirements.

47 Software Engineering Meets Services and Cloud Computing

Stephen S. Yau and Ho G. An

Service-oriented software engineering incorporates the best features of both the services and cloud computing paradigms, offering many advantages for software development and applications but also exacerbating old concerns.

54 Software Engineering—Missing in Action: A Personal Perspective

David Lorge Parnas

Although a huge number of articles have been written about software development and many interesting ideas have been proposed, researchers and practitioners have failed to create a new engineering discipline focused on building software-intensive systems.

RESEARCH FEATURE

60 Display Power Management That Detects User Intent

Jae Min Kim, Minyong Kim, Joonho Kong, Hyung Beom Jang, and Sung Woo Chung

In a proposed display power-management scheme, the laptop detects when the user is not looking at the screen, boosting low-power operation to 50 percent and increasing energy savings by up to 13 percent.

For more information on computing topics, visit the Computer Society Digital Library at www.computer.org/csdl.

IEEE Computer Society: <http://computer.org>
 Computer: <http://computer.org/computer>
computer@computer.org
 IEEE Computer Society Publications Office: +1 714 821 8380

Flagship Publication of the IEEE
 Computer Society

October 2011, Volume 44, Number 10

6 The Known World

The Habit of Change
David Alan Grier

9 32 & 16 Years Ago

Computer, October 1979 and 1995
Neville Holmes

NEWS

11 Technology News

Bringing 3D to the Small Screen
Sixto Ortiz Jr.

14 News Briefs

Lee Garber

MEMBERSHIP NEWS

67 IEEE Computer Society Connection

70 Call and Calendar

COLUMNS

81 Entertainment Computing

Object Digitization for Everyone
Justin Clark

84 Invisible Computing

From Data Analysis and Visualization to Causality Discovery
Min Chen, Anne Trefethen, René Bañares-Alcántara, Marina Jirotko, Bob Coecke, Thomas Ertl, and Albrecht Schmidt



88 Software Technologies

Guidelines for Software Development Effort Estimation
Dirk Basten and Ali Sunyaev

91 Discovery Analytics

Twitter Mood as a Stock Market Predictor
Johan Bollen and Huina Mao

95 Education

Learning How to Prepare Computer Science High School Teachers
Mark Guzdial

100 The Profession

Digital Machinery and Analog Brains
Neville Holmes

DEPARTMENTS

- 4 Elsewhere in the CS
- 59 Computer Society Information
- 72 Career Opportunities



IEEE
 computer society



Reuse Rights and Reprint Permissions: Educational or personal use of this material is permitted without fee, provided such use: 1) is not made for profit; 2) includes this notice and a full citation to the original work on the first page of the copy; and 3) does not imply IEEE endorsement of any third-party products or services. Authors and their companies are permitted to post the accepted version of their IEEE-copyrighted material on their own Web servers without permission, provided that the IEEE copyright notice and a full citation to the original work appear on the first screen of the posted copy. An accepted manuscript is a version which has been revised by the author to incorporate review suggestions, but not the published version with copyediting, proofreading and formatting added by IEEE. For more information, please go to: http://www.ieee.org/publications_standards/publications/rights/paperversionpolicy.html.

Permission to reprint/republish this material for commercial, advertising, or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to the IEEE Intellectual Property Rights Office, 445 Hoes Lane, Piscataway, NJ 08854-4141 or pubs-permissions@ieee.org. Copyright © 2011 IEEE. All rights reserved.

Abstracting and Library Use: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy for private use of patrons, provided the per-copy fee indicated in the code at the bottom of the first page is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

IEEE prohibits discrimination, harassment, and bullying. For more information, visit www.ieee.org/web/aboutus/whatis/policies/p9-26.html.

ELSEWHERE IN THE CS

Computer Highlights Society Magazines

The IEEE Computer Society offers a lineup of 13 peer-reviewed technical magazines that cover cutting-edge topics in computing, including scientific applications, design and test, security, Internet computing, machine intelligence, digital graphics, and computer history. Select articles from recent issues of Computer Society magazines are highlighted below.

Software

Successful videogame developers must contend with the difficulties of ensuring that their product is fun. Addressing this soft requirement, incorporating nontrivial multimedia, and other domain-specific concerns bring novel challenges to software development. *Software's* September/October special issue on "Engineering Fun" presents articles that illustrate the difficulties in designing and producing applications that must somehow result in an enjoyable player experience while still satisfying the demands of a large, complex, graphical, real-time, and distributed application.

Intelligent Systems

Cyber-physical-social systems (CPSSs) provide an ideal paradigm for the design and construction of command and control organization. In the July/August issue of *IS*, "Cyber-Physical-Social Systems for Command and Control" presents the concept of a CPSS for command and control and discusses its operational process and self-synchronization mechanism. The authors propose a CPSS that incorporates the essential characteristics of an operational mechanism and connects physical networks, cyberspace, mental space, and social networks.

IEEE Computer Graphics AND APPLICATIONS

In "Virtual Prototyping of Shoes" in the September/October issue of *CG&A*, a team of authors from the Chemnitz University of Technology proposes a VR-based system for prototyping shoes that employs a user interface

that mimics a designer's conventional work style. Using a pen and a shoe last as 3D proxy objects, users can sketch design lines onto the last. The system employs an algorithm for real-time sketch recognition on curved surfaces. On the basis of the framework of curves, the system creates further design elements (for example, leather pieces) to complete the model. Users can employ three pen-based interaction metaphors to interact directly with design elements, to input values, or to make choices from a menu. A user study shows that conventional designers can easily learn and apply these interaction methods.

Computing IN SCIENCE & ENGINEERING

Writing in the September/October issue of *CiSE*, a team of medical researchers from Indiana and Washington suggests a new model for image-guided radiation therapy in "Survey: Real-Time Tumor Motion Prediction for Image-Guided Radiation Treatment." Tumor motion caused by a patient's breathing creates challenges for accurate radiation dose delivery to a tumor while sparing healthy tissues. IGRT helps, but a lag time remains between tumor position acquisition and delivering a radiation dose to that position. An efficient and accurate predictive model is thus an essential requirement for IGRT success.

IEEE SECURITY & PRIVACY Building Dependability, Reliability, and Trust

Information security is fast becoming a cyber-industrial complex, and as researchers have discovered, complex systems have notable side effects, including the potential for the disastrous rise of misplaced power. Writing in the July/August issue of *S&P*, Daniel E. Geer Jr. of In-Q-Tel looks at the risks of power imbalances associated with complex systems in "Eisenhower Revisited."

IEEE pervasive COMPUTING MOBILE AND UBIQUITOUS COMPUTING

This year, the car celebrates its 125th birthday. In 1886, Karl Benz applied for a patent for a motorcar, the Benz Patent-Motorwagen. The sole function of this mechanical device was providing individual transportation. Since

then, cars have changed enormously. Safety and comfort have become major drivers of innovation in the field. In the past 30 years, computing technology has changed cars fundamentally. Many mechanical functions are now computer-controlled or linked by digital networks. Today, innovation in the automotive industry is largely due to information technology. The July-September issue of *PvC* covers the ever-evolving world of automotive pervasive computing.

IEEE Internet Computing

“Adversarial Machine Learning,” by J.D. Tygar of the University of California, Berkeley, briefly introduces the emerging field of adversarial machine learning, in which opponents can cause traditional machine learning algorithms to behave poorly in security applications. Tygar gives a high-level overview and mentions several types of attacks, as well as several types of defenses, and describes theoretical limits derived from a study of near-optimal evasion. Read Tygar’s article in the September/October issue of *IC*.

IEEE micro

The July/August issue of *Micro* focuses on big chips, highlighting some of the tradeoffs inherent in large processors with respect to performance capability, power density challenges, scalability of communication, and packaging and cooling costs. This issue also explores new technologies that can achieve the effect of big chips, such as 3D integration, along with associated tradeoffs. Read more in “Big Chips and Beyond,” by IBM’s Erik R. Altman.

IEEE MultiMedia

MultiMedia guest editors Jian Lu of Shanda Innovations, Xian-Sheng Hua of Microsoft, and Dong Xu of Nanyang Technological University introduce the magazine’s July-September issue in “Visual Content Identification and Search.”

The past few years have witnessed rapid development and commercialization of visual content identification and search technologies that are concerned with identifying and searching visual content, particularly image and video content, based on visual similarities. This special issue contains several research articles covering a diverse range of topics in visual content identification and search.

IT Professional

TECHNOLOGY SOLUTIONS FOR THE ENTERPRISE

Advances in social computing can help firms enhance and leverage internal capabilities, such as knowledge work and collaboration, and seamlessly integrate the vast

information resources available on the Internet. Many firms, however, are still developing the practices that will enable them to leverage social networking in the enterprise. Read “Social Networking in the Enterprise” in *IT Pro*’s July/August issue to learn more about the role of social networks in a business setting.

IEEE Annals of the History of Computing

Throughout the 1980s and 1990s, the high-end Swedish camera manufacturer Hasselblad struggled to integrate its product lines with emerging digital imaging technology. Hasselblad’s history illustrates how digital technology emerges in various high-end niche applications and later enters the mainstream markets and displaces incumbents. The Hasselblad case exemplifies how incumbent firms encounter difficulties when such technologies render their skills and products obsolete. Read more about these challenges and solutions in “Hasselblad and the Shift to Digital Imaging,” by Christian Sandström of Chalmers University of Technology, in the July-September issue of *Annals*.

Call for Participation: IEEE Computer Society SPECIAL TECHNICAL COMMUNITY ON SOCIAL NETWORKING

IEEE Computer Society Special Technical Communities (STCs) offer a new engagement model for Computer Society (CS) members and much broader computer practitioner world to collaborate for their individual and mutual benefit, specifically to advance technical topics to the benefit of the profession. The main objectives of STCs are:

- Extend CS beyond traditional membership and activities.
- Give new outlets for the membership to create and distribute more IP.
- Enrich professional activities (e.g. newsletters, sharing of best practices)
- Strengthen governance through new dynamic organizational structure

STC on Social Networking (SN)

The STC on Social Networking intends to be Agora for researchers with similar interests to meet and gather. We are interested in (but not limited to) the following topics:

- Social network services, applications, and tools
- Social computing and social search
- Social multimedia and social communications
- Mobile social networking
- Social network analysis and visualization
- Standardization trends and federated social Web initiatives
- Business models
- Societal issues (e.g., privacy and data protection)

STC Contact Christian Timmerer, Alpen-Adria-Universität Klagenfurt, Austria;
E-mail: christian.timmerer@itec.uni-klu.ac.at; Web: <http://research.timmerer.com>

Chairs

Christian Timmerer
Klagenfurt University, Austria
George K. Thiruvathukal
Loyola University Chicago, USA

Why to participate?

THE entry point for researchers in SN
Communication with like-minded people
through existing SN tools

How to participate?

Come and join us via:
<http://computer.org/stcsn>
<http://facebook.com/stcsn>
<http://twitter.com/stcsn>
<http://linkd.in/stc-sn>

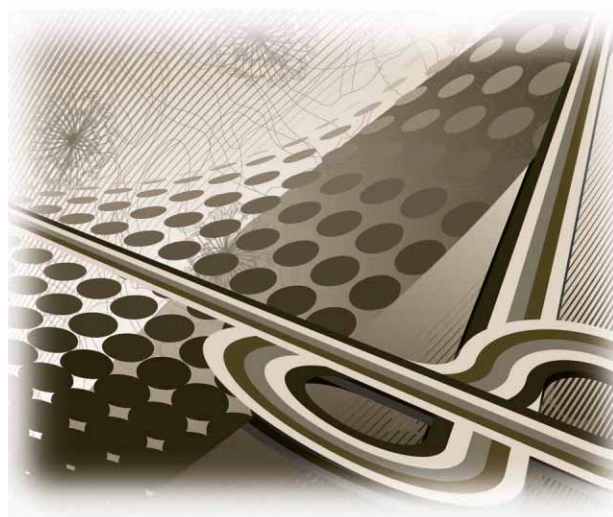
IEEE  computer society

THE KNOWN WORLD

The Habit of Change

David Alan Grier

George Washington University



As technology has changed, the fundamental way of extracting meaning from data has also changed.

We're living in a time that discourages wholesale change. I know many people who are trying to squeeze an extra year or perhaps two from a laptop of uncertain age or a server that should rightly be retired. The optimistic promises of Moore's law, which traditionally offered substantial improvements for a very modest expense, don't seem as enticing as they once did. Rather than bear the disruption of a change, many seem to be delaying the process of upgrading machines, installing new software, and converting large data files.

DATA CONVERSION CONCERNS

Problems with data conversion seem to be of increasing concern. As one anecdotal example, I recently saw my friend Darrow take an old PDA from his pocket to use in making a note about something we'd been discussing. The device is easily a decade old and can no longer be replaced on the new equipment market. Its operating system has been invalidated by a lawsuit and

surpassed by other programs. At one point, Darrow noticed me looking at the device and commented, more apologetically than necessary, that the little computer did everything he needed it to do, so he felt no need to replace it.

Our field has regularly employed mockery and scorn to force people to embrace new hardware or software. I recall a recent meeting, not that long ago, in which a division leader shamelessly derided a group of staff members who clung to an older set of productivity tools. In making his case, he suggested that Steve Jobs would be disappointed if they didn't change their ways and openly invited others who were present to shun the holdouts. To this day, I don't understand why he felt compelled to ridicule those who were using some slightly outmoded programs, but I saw that he was willing to push those employees very hard to upgrade their systems.

There are, of course, many reasons to resist changes. New machines don't always work as advertised, and new programs often are incompatible with older versions. Yet, we're rapidly

reaching a point where our plans for the future are dictated less by hardware and software and more by data. We've expended a great deal of effort to make data portable and to allow it to be processed by many different software programs. Yet, we haven't taken the same care to make sure that the lessons from that data can transfer easily from system to system.

Data now personalizes almost every piece of software that we use. My word processor knows the words I'm likely to misspell, my browser knows the kind of sites I like to visit, and my car knows how I tend to approach traffic problems. In perhaps the crudest example, I have about 30 Gbytes of research data, all carefully cross-referenced and linked by the themes I've found in it. I'm concerned that my insights will be lost when I'm ultimately forced to move to a new platform. When I purchased a new computer in celebration of completing my first book a little more than eight years ago, I lost the collection of ideas behind that work. Hence, I'm approaching my next upgrade with great caution.

EXTRACTING MEANING FROM DATA

The problems of upgrading data concern not merely structure and format. If those were the only obstacles, we should easily be able to create some kind of computer program that manages the inevitable transition from the old world to the new. But the problem isn't merely technological. Technology has given us new ways of looking at data; hence, as technology has changed, the fundamental way of extracting meaning from data has also changed.

Punched card tabulators, for example, not only taught us how to deal with large datasets, but also raised fundamental questions about how these datasets might fail to reveal important information. Using punched card tabulators, pollster George Gallup showed that small sets of data, if properly chosen, could do a better job of capturing information than a large set. Others have shown with high-speed data connections how the actions of our distant neighbors shape our immediate lives. Still others have used clever algorithms to find information thought to be forever hidden in deep layers of noise, the forgotten voice that was overwhelmed by the scratches on a recording.

Each of these advances has required us to give up an old way of looking at data. Gallup faced substantial opposition from critics who believed that he simply wasn't talking to enough people when he gathered data about public opinion. He had to make the case that it was more important to manage the information that could be derived from a small set of data than to compile a large dataset that would be unmanageable. The resistance that Gallup experienced has been repeated as we've embraced new tools for studying data such as rendering, learning, and data-mining algorithms.

A PUNCH IN THE CHEST

Slightly more than a decade ago, a punch to the chest taught me that new methods of extracting meaning from data could produce physical resistance. The punch was two-fingered but landed right on my sternum. The blow wasn't enough to do substantial damage, but it stung enough to tell me that the person who delivered it was upset and felt that he was being ignored.

The confrontation occurred at an early data-mining conference. I had raised a little money and had recruited one of the pioneers in the field to talk about his methodology to a group of selected statisticians,

When one algorithm has run its course or revealed its limitations, we dispose of it and deploy a new one in its place.

policymakers, and computer scientists. The algorithms he developed implemented ideas that are common to the field: they processed large multivariable datasets and searched for relationships within the data. However, while the algorithms resembled traditional statistical methods, they completely ignored both the field's philosophy and its probability models.

It was that disregard of traditional statistical models that instigated the punch. The person involved was a senior statistician named Harvey. I had only known him for a year or two, but I thought of him as a sweet, grandfatherly sort. I had invited him to the meeting because I thought he might be interested in new ideas.

Initially, Harvey was quite jovial and appeared to enjoy meeting those who were attending the conference. However, as the talks progressed, he became increasingly restless. He twisted in his chair and made

comments to a neighbor that could be clearly heard across the room. Finally, he approached me at the afternoon break, clearly quite angry.

"This is all wrong," he said. "This is the most dangerous thing you could have done."

Although I was grateful to learn something about the cause of his anxiety, I wasn't particularly interested in listening to him. I was running the meeting and had other things to do. As I recall, we were in the process of changing the venue for dinner, having just discovered that the place where we were to dine had been closed because the owner had been arrested for running guns to an ethnic conflict on the other side of the globe. Although I'll admit that it was clearly intended to make him go away, I said something that I thought was respectful.

"You've got to understand," he said, "this data mining isn't science. It's mindless calculation. In real science, you have to formulate a hypothesis, build a model, gather data, and test the results. You don't just calculate every equation that comes into your mind or the mind of your computer. This undermines everything we've done for 40 years."

If Harvey had stopped at this point or I hadn't been so dismissive, we might have had a fair fight. Karl Popper and Bruno Latour would have been escorted into the center ring and had a contest worthy of the ages. However, this wasn't a conflict about ideas. It dealt with the validation of a career. Although Harvey had an office and was involved in research, he was no longer the leader that he once had been.

Harvey's voice rose again. I moved away. The punch landed—two fingers on the sternum.

It was nothing more than a burst of emotion. His anger subsided instantly. Apologies were made. Forgiveness was offered and accepted. However, Harvey opted not to remain for the rest of the seminar.

THE KNOWN WORLD

Even without attending the rest of the talks, Harvey had gauged the impact of data mining better than most of the people who were at the seminar. No one seemed to have witnessed Harvey's punch or, if they did, they were too polite to mention it. At the dinner, they talked about the nature of data-mining methods, the kinds of structures they were capable of detecting, and the class of problems that might benefit from these methods.

I recall that the statisticians talked about how data mining would need to be pulled within the statistical framework. They would need to develop traditional statistical models, articulate the correct notion of probability, and make proper statistical inferences. At the end of the night, I paid the tab and watched the group disperse into the night. The statisticians departed in one direction and the computer scientists in another.

AN EVOLVING DISCIPLINE

Most commentators have noted that data mining and statistical methods both deal with the analysis of data but they have come to occupy independent spheres. "In this work," wrote one of the conference attendees, "there is little probability modeling of random variation in the data." He made some effort to connect

data mining to the work of an early scholar whose writings were so broad that they encompassed many scientific fields. He argued that the kinds of problems data mining was addressing had once been the domain of statisticians but that the "center of gravity of research" had shifted to computer science.


Data mining is only one of the new ways of learning lessons from data. We now deploy a host of learning algorithms to count clicks, track motion, compare transactions, and otherwise delve into the symbolic representations of our lives. When one algorithm has run its course or revealed its limitations, we dispose of it and deploy a new one in its place. Once the replacement is completed, we rarely reuse the lessons of the past. All we can do is to feed this new system a set of data and let it learn afresh. We can't transfer lessons learned. We have no common architecture, as we do with hardware, to mitigate the cycles of change.

In describing how automobiles dealt with cycles of change, industry pioneer Alfred P. Sloan noted that the "necessity of change forced us into regularizing change." The auto industry learned to manage change by updating its designs on an annual basis, incorporating its work into a new model. At the moment, however, we have no such concept for data,

especially for data that is intended to capture the behavior of individuals and organizations.

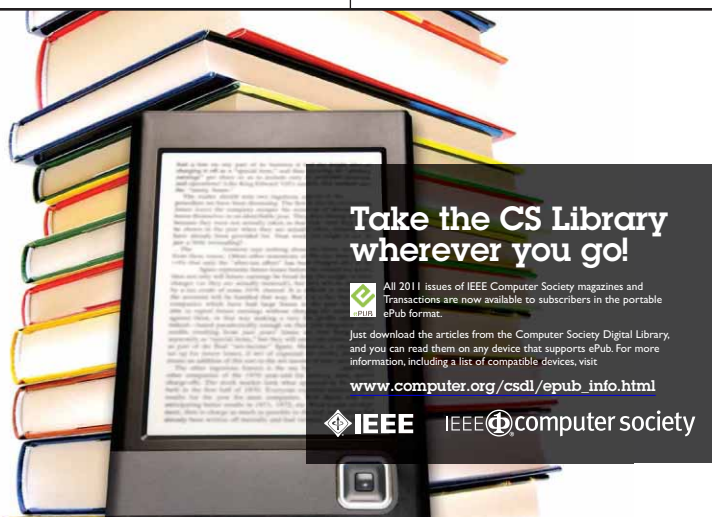
Many who deal with large-scale data believe that we're in the fourth stage of the Information Age. The first stage belonged to hardware, the second to software, and the third to communications. This fourth stage in their hierarchy belongs to data.

Recently, I heard a presentation arguing that we not only can ignore the probability models of statistics, we also can ignore the very concept of causality. Large datasets will give us so much information about phenomena, explained the speaker, that we'll be able to make judgments from the relations that our algorithms find. We'll just accept them as true.


It was a lesson in rational positivism that clearly resonated with those present. Individual after individual rose to express agreement with the presenter's comments, to note how quickly we're finding new ways of processing data, and to predict a great future. It was the end of the day, a time to put cares aside. They had spent the morning and afternoon discussing the problems of how to gather data, how to remove anomalous elements, and how to learn from it. At some point, they'll need to address the problems that come from the habit of change. We're still inclined to abandon the old as we take up the new. However, at some point, we may find that such a habit needs to be put aside. 

David Alan Grier, an associate professor of international science and technology policy at George Washington University, is the author of the upcoming book, The Company We Keep. Contact him at grier@gwu.edu.

cn Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.


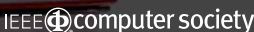


Take the CS Library wherever you go!

 All 2011 issues of IEEE Computer Society magazines and Transactions are now available to subscribers in the portable ePub format.

Just download the articles from the Computer Society Digital Library, and you can read them on any device that supports ePub. For more information, including a list of compatible devices, visit

www.computer.org/csdl/epub_info.html

 IEEE  IEEE computer society

32 & 16 YEARS AGO

OCTOBER 1979

HARDWARE TESTING (p. 7) “The testing of digital systems has grown increasingly complex. LSI circuits, commonplace in today’s systems, require thorough testing at the component level. VLSI adds even more complexity. How are these challenges being met? This special issue surveys the state-of-the-art and attempts to answer the question.”

LOGIC TESTING (p. 9) “The testing problem has two major facets: (1) test generation and (2) fault simulation. With the vast increase in density, the ability to generate test patterns automatically and conduct fault simulations with them has drastically waned. As a result, some manufacturers are foregoing these more rigorous approaches and are accepting the risk of shipping a defective product.”

MEMORY TESTING (p. 23) “Semiconductor memory testing is often thought of as a one-shot operation occurring just prior to product delivery. However, testing plays a much more important role in the life cycle of a product. The three key areas of product production—device design, lithography, and processing—are verified and monitored through the test and characterization of the end product.”

SYSTEM TESTING (p. 32) “... Rapid changes in the electronics industry quickly render any rules or procedures obsolete. Therefore, the only way to ensure that products are actually testable is for the designer to clearly understand the testing process, including its needs and capabilities. And he must be able to implement state-of-the-art test techniques as he designs state-of-the-art products. ...”

ANALOG TESTING (p. 40) “Because an analog electronic circuit or device may be tested many times during its lifetime, testing has become a significant component of life-cycle cost. Automatic test equipment, or ATE, reduces this cost and improves the quality of testing as well. Today’s automatic testers are almost exclusively computer-controlled machines that execute user-created test programs. ...”

SELF-TESTING COMPUTERS (p. 49) “... Controllability and observability problems—long the nemeses of digital test engineers—have become particularly difficult as computer IC densities have increased. New testing approaches have become necessary in many computer applications to provide on-line visibility into computer functional processes. A viable approach to increased on-line computer process visibility is built-in-test, which is being used in digital computers as a means of providing continuous on-line performance monitoring, particularly in modular computer systems. ...”

**PROGRAMMING
SMALL** (p. 61)

“Microdare is a new high-level language system for laboratory automation, signal processing, control, and simulation.

Combining high computing speed with direct execution (i.e., no external compiler, linker, or loader is needed), Microdare is embedded in an advanced Basic dialect which serves for interactive program entry, editing, file manipulation, job control, and for programming multi-run experiments. ...”



FEATURE LEARNING (p. 75) “The usefulness of computer simulations in learning has long been recognized. Indeed, according to some proponents, this is the only mode in which computers should be used! I, however, maintain that it is only one of many valuable computer-based learning approaches; the classroom environment presents a full spectrum of possibilities.”

SCIENCE VERSUS ENGINEERING (p. 88) “... The function of the computer scientist is to know, while that of the software engineer is to do. The computer scientist adds to the store of verified, systemized knowledge of the computer-centered world; the software engineer brings this knowledge to bear on practical problems.”

ATHLETICS (p. 97) “A computerized system will be playing a major role in preparing American athletes for the 1980 Winter and Summer Olympic Games.

“The computer, an Eclipse S/250 donated to the US Olympic Committee by Data General, and a Whizzard 7000 vector refresh graphics system, donated by Megatek, will be used by athletes training for international competition, starting with the 1980 Olympics. Dr. Gideon Ariel, a member of the Sports Medicine Committee and developer of the programming that allows the computer to analyze and improve athletic performance, will direct the effort at the committee’s new Bio-Mechanics Laboratory.”

CONFERENCE (p. 100) “Registration at the ACM SIGGRAPH conference on Computer Graphics and Interactive Techniques, combined this year with the IEEE Computer Society conference on Pattern Recognition and Image Processing, exceeded 2200, double 1978’s 1100. Other numbers set records, too: 80 exhibitors, 1400 exhibitor guests, 2000 paying exhibit visitors. Total: almost 6000.”

Editor: Neville Holmes; neville.holmes@utas.edu.au

32 & 16 YEARS AGO

OCTOBER 1995

FUTURE SCHOOLS (p. 10) “FutureSchools are in the knowledge business, which means they are content providers just like newspaper, magazine, and software publishers, and movie and video game producers. The Hudson Institute, headquartered in Indianapolis, Indiana, reviewed 20 years of research on computer-based instruction and found that students learn 30 percent more in 40 percent less time and at 30 percent less cost when using computer-aided instruction. Who says automated delivery isn’t as good as delivery in the flesh? ...”

INTERNET ADDRESSING (p. 12) “As new mobile-computing technologies emerge and Windows 95 brings thousands of new users on line, the need for more Internet protocol address space is becoming crucial. Because of the rapid growth of the Internet, the Internet Engineering Task Force (IETF) is furiously working on an update to its transmission-control protocol/Internet control protocol, the Internet’s underlying addressing and routing scheme.”

HARDWARE DESCRIPTION (p. 18) “As the name suggests, VHDL [VHSIC Hardware Description Language] is a language for describing digital hardware. It was developed under the auspices of DOD’s Very High Speed Integrated Circuits (VHSIC) program in the 1980s and accepted as an IEEE standard in 1987. VHDL is very similar to a programming language, but the end result is a description of a piece of hardware, not an algorithm to be executed on a processor. A VHDL ‘program’ is usually called a model because it rarely describes the piece of hardware completely; certain details of the hardware’s behavior are always ignored. ... If a VHDL model describes the hardware in sufficient detail, a program called a synthesizer can automatically convert it into a gate-level description that can be manufactured.”

SOFTWARE REUSE (p. 36) “In the future, new application requirements and technological advances can be expected to drive domain evolution, generating a continuous process of domain analysis and domain architecture restructuring. Domain and reuse library artifacts will be continually created and restructured to produce new software product inventories for use in expanded or reorganized domains. As new programming languages and platforms are introduced, we will increasingly be concerned with reengineering selected reuse library components rather than reengineering entire legacy software applications.”

OBJECT TECHNOLOGY (p. 57) “Although the benefits of object-oriented systems are recognized, OT has been embraced somewhat slowly, because it requires adapting existing structured tools and techniques to support an object-oriented approach, providing a smooth learning

curve for people whose experiences and expertise are in developing systems using structured methods, and adaptability of existing structured software through reengineering with the new object-oriented software. However, we should begin preparing carefully for the inevitable transition to object technology. ...”

TEACHING COMPUTING (p. 73) “... To a large extent, computer technology changes far more rapidly than the basic ideas of the science of computing, which center on the notion of an algorithm and its use in computing systems. Based on the sixty years or so since the pioneering days of CS, these ideas have lasting and fundamental value. Thus, although a proposed high school program should enhance a student’s ability to exploit computers beneficially, its backbone must be based on science. The program should provide insight, knowledge, and skills independent of specific computers and programming languages. ...”

TELEVISION (p. 81) “For Advanced Television to play an important role in the emerging NII [National Information Infrastructure], [Apple’s Donald A.] Norman argued, ATV would have to be made fully computer compatible. For that to happen, the computer industry would have to be included in the standards process, and standards would have to be based on quality and long-term flexibility rather than on the short-term cost-minimization strategy that he sees prevailing in the television industry.”

INTERNET SECURITY (p. 101) “These activities seem certain to lead the Internet into a new age characterized by the security equivalent of citizens’ passports. Every Internet user will have one or several public key pairs and corresponding certificates issued by CAs [certification authorities] who act as trusted third parties. The provision of security services such as authentication, data confidentiality and integrity, access control, and nonrepudiation services will be based on the availability of public key certificates on a global scale.”

BENCHMARKING (p. 102) “Function points provide useful metrics on two key components of software quality: measuring defect potentials and calculating defect removal efficiency levels.

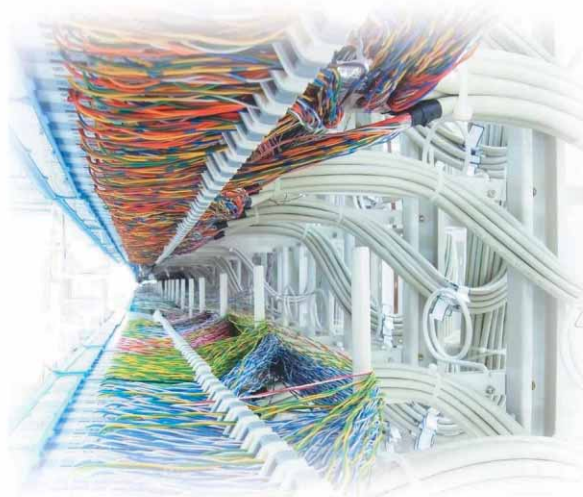
“The defect potential of a software application is the total quantity of errors found in requirements, design, source code, user manuals, and ‘bad fixes’ or secondary defects inserted as an accidental byproduct of repairing other defects. The defect removal efficiency is the percentage of software defects removed prior to delivery.”

PDFs of the articles and departments from Computer’s October 1979 and 1995 issues are available through the IEEE Computer Society’s website: www.computer.org/computer.

TECHNOLOGY NEWS

Bringing 3D to the Small Screen

Sixto Ortiz Jr.



Although touted as the wave of the future, 3D technology hasn't been widely implemented in mobile devices so far. However, that may be about to change.

An important recent trend in consumer technology is delivering 3D content to the mainstream user via devices such as TVs, PCs, smartphones, and game consoles.

Proponents say the improved viewer experience alone will make the trend an important and popular one. However, 3D has not been widely implemented in mobile devices so far.

3D content is data-intensive, and the creation, delivery, and viewing of the material place heavy demands on computing and communications resources. Thus, until recently, 3D technology hasn't been seen as appropriate for resource-constrained mobile devices.

Now, though, vendors are beginning to deploy the technology in mobile settings, one of the fastest-growing segments of the consumer electronics market.

Advances in mobile chips, mobile networks, and autostereoscopic display technology for viewing 3D content without special glasses have enabled these developments.

Game-console and smartphone manufacturers are also starting to

support 3D, according to market-research firm In-Stat.

In-Stat senior analyst Stephanie Ethier said competition between Nintendo and Sony, the two leading handheld game console providers, will accelerate 3D use in these devices. Nintendo's 3DS console, released earlier this year, has led the charge.

Some industry observers say that because mobile devices are small enough to avoid some of the technical issues that larger machines have in displaying 3D, they may actually be the first to widely implement the approach.

However, not much 3D content is currently available for mobile devices, noted Art Lathrop, market development manager at 3M's Optical Systems Division.

This will be the biggest barrier to mobile 3D adoption in the near future, said Ethier.

DEMAND FOR MOBILE 3D

Several potential applications are driving demand for mobile 3D. Games are an obvious example because of the way 3D could enhance the playing experience.

The ability to view movies and other video content generated and shared by users is also driving 3D mobile device adoption.

Sharing videos and photos with friends ranked in the top three uses of 3D cell phones in a recent survey conducted in Japan, noted Philip Lelyveld, program manager for the University of Southern California's Entertainment Technology Center.

Also, studios are developing 3D movies for mobile devices using the H.264 video compression codec, said Veera Raju, engineering manager for Texas Instruments' (TI's) Natural User Interface Group. This codec reduces the size of files and makes them easier to transmit to and play on mobile devices.

Added Lelyveld, researchers and content providers are working on augmented reality applications that superimpose useful 3D information on games and marketing presentations, as well as on other types of content.

This approach could work well with hospitality services that show 3D views of hotel rooms and amenities with overlaid information, noted Mino Patel, director of travel,

TECHNOLOGY NEWS

transportation, and logistics projects for NIIT Technologies, an India-based IT services provider.

It could also work with applications that provide navigation assistance via a superimposed, highlighted 3D path through a city or building, he added.

GETTING 3D ONTO SMALL DEVICES

Sharp released the first 3D phone that didn't require users to wear glasses, the Mova SH251, in 2003 for sale by Japanese cellular services provider NTT DoCoMo.

During the next few years, companies such as Hitachi and Samsung produced a few models of 3D handsets.

The technology

Mobile 3D faces technical challenges. For example, phone and game console manufacturers must develop small-footprint devices with the processing power to run 3D video and images, as well as displays that can clearly show the content.

Also, downloaded or streamed material must be transmitted over resource-constrained wireless networks.

The additional information needed to properly display 3D video or images typically adds 50 percent more data to a content file, said Chris Yewdall, executive director and CEO of 3D software developer Dynamic Digital Depth (DDD). However, he noted, compression techniques can ease the strain of transmitting the large files.

In addition to using compression, DDD's system transmits a file's 2D information plus metadata containing depth-related information—such as the position of objects in three dimensions—about video or images. This adds only 5 percent more data to a file, Yewdall said.

When a device downloads the file, it decompresses and decodes the depth-related data and coordinates it with the transmitted 2D content. The device then generates the 3D image.

DDD's software can also develop 3D scenes from existing 2D video by estimating the depth of objects using various visual cues.

The software then creates depth maps for each video frame in real time. The system uses the depth maps to move pixels in the corresponding 2D image to create pairs of slightly different images for each of the viewer's eyes. This yields the desired 3D effect.

Chips

Mobile devices that can handle 3D content require computationally intelligent processors with strong image-processing and display capabilities along with energy efficiency.

TI's OMAP 5 includes two ARM Cortex-A15 MPCore low-power processors and two Cortex-M4 chips. This architecture includes a low-power image signal processor.

Displays

Mobile devices require displays that can enable viewing of 3D content. There are several 3D mobile display manufacturers, including Reach3D and MasterImage 3D.

A key goal for proponents of 3D on mobile and other devices is eliminating the special glasses used in movie theaters that have relegated the approach to gimmick status.

Autostereoscopic. There are two autostereoscopic approaches.

Vendors are starting to deploy 3D in mobile devices, one of the fastest-growing consumer electronics market segments.

Mobile chipsets accomplish these goals via a reduced instruction set computing architecture. Processor manufacturers frequently place a dedicated GPU, which can efficiently handle the graphics processing, on the same chipset as the CPU.

The processors are energy efficient because they don't run as fast as their PC counterparts. In addition, the 3D mobile chips can go into sleep mode when not in use.

Several mobile processors currently offer these capabilities, such as the Apple-designed A5 (manufactured by Samsung), Intel's Atom series, Nvidia's Tegra chips, and TI's OMAP 5 processors.

Some of these products include chips by ARM, a well-known mobile-processor designer.

Tegra features a dual-core ARM Cortex-A9 processor combined with 1 Gbyte of memory and an 8-core ULP (ultra-low-power) Nvidia GeForce GPU. The chip can work with multiple video formats.

A *multiview* display uses barriers, special lenses, or strategically placed lights and prisms to send one perspective of a scene to one of the viewer's eyes, and another perspective to the other. The user's optical system fuses them into a single 3D image.

Light-field displays, on the other hand, generate pixel-like elements that recreate the actual pattern of light—including its direction and angle of arrival—that would travel from all parts of a 3D object to a viewer's eyes.

With large autostereoscopic displays, viewers must stand in certain positions relative to the screen for the best effect. This isn't generally an issue with small devices, which typically have just one viewer.

Display film. 3M has developed Vikuiti 3D Display Film, which helps enable the showing of 3D content on mobile devices.

The polymer film, which Figure 1 shows, is added to a traditional LCD

screen. The LCD panel first displays a 2D version of an image from the perspective of a viewer's left eye. The device illuminates the LEDs on the viewer's left side and sends the image to only the left eye, with the help of the 3M film. The system then shuts off the LEDs on the left side and repeats the cycle for the viewer's right eye.

When this process is performed quickly, the user's optical system combines the images, yielding a 3D effect.

IMPLEMENTATIONS

Several 3D mobile devices are available.

Sharp's Sharp Galapagos 003SH and 005SH smartphones use an autostereoscopic display. Sharp is also rolling out a mobile 3D TV featuring a 3.4-inch LCD screen.

Many industry observers believe games will lead 3D adoption by consumers. Nintendo's 3DS console is the first handheld gaming device that displays 3D without requiring special glasses. The console includes a slider that can turn off the 3D effect or adjust its intensity.

Flipping Samsung's B710 smartphone—which runs DDD software—from a vertical to a horizontal orientation converts an image from 2D to 3D.

HTC's EVO 3D smartphone uses DDD's TriDef 3D Mobile software to render still and video content in three dimensions.

The phone uses a dual 5-megapixel camera that can capture 2D and 3D photos. Users can move a slider to toggle the device's camera mode between 2D and 3D.

LG Electronics' Optimus 3D (called the Thrill 4G in the US) also uses DDD's TriDef 3D Mobile software.

CHALLENGES IN MULTIPLE DIMENSIONS

By and large, there is little 3D content available today for viewing on mobile devices outside of a small amount of video and a few games.

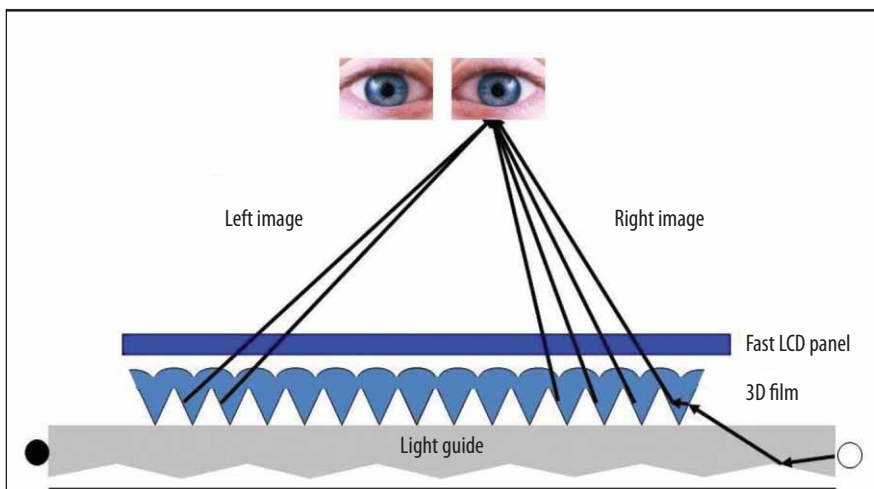


Figure 1. 3M's Vikuiti 3D Display Film is added to a traditional LCD screen to enable 3D viewing on mobile devices. The LCD panel first shows a 2D version of an image from the perspective of a viewer's left eye. The device illuminates the LEDs on the viewer's left side and sends the image only to the left eye, with the help of the 3M film. The system repeats the cycle for the right eye.

However, 3M's Lathrop said the movie industry is beginning to work on more mobile 3D material.

DDD's Yewhall predicted the amount of content will increase as more consumers adopt devices capable of playing it.

However, some industry experts predict that mobile 3D hardware development will progress faster than content production. Hardware makers thus might have to pace their product releases to market demand to make sure there's enough content to attract buyers.

Mobile 3D technology's visual quality isn't uniformly high yet, Lathrop said. Moreover, he added, some devices don't do a good job displaying both 2D and 3D content, which could be frustrating for viewers.

Another challenge, Yewhall said, is that using traditional lossy 2D video

compression with 3D content, which many providers do, could reduce image quality.

Texas Instruments' Raju predicted that 3D technology will steadily enter the mainstream, especially for game content, which has a shorter development cycle than movies and other types of material.

However, 3M's Lathrop said, content and quality problems must be resolved before broad adoption can occur. **□**

Sixto Ortiz Jr. is a freelance technology writer based in Amarillo, Texas. Contact him at sortiz1965@gmail.com.

Editor: Lee Garber, Computer;
l.garber@computer.org

STAY
CONNECTED

TWITTER | @ComputerSociety
| @ComputingNow
FACEBOOK | facebook.com/IEEEComputerSociety
| facebook.com/ComputingNow
LINKEDIN | IEEE Computer Society
| Computing Now

NEWS BRIEFS

New Platform Offers Gaming Cubes

A new product uses cubes with LCD screens and near-field communications (NFC) to provide a novel approach to gaming.

Users play games with the Sifteo Cubes by physically moving them around and touching them to one another, rather than by pressing keys or moving joysticks and then watching what happens on a computer display.

The battery-powered blocks, made by Sifteo Inc., measure 1.5 inches square, stand one-half-inch tall, and have 128 × 128 pixel color screens on top.

The cubes use NFC sensors, which wirelessly communicate and recognize when the blocks are close to one another and which specific sides are adjacent. This allows characters and objects to move from one screen to another as desired.

These controls enable a range of games, including those in which characters are guided from one cube to another or in which words are unscrambled. Flipping or shaking the cubes can reset a game or return a user to the main menu.

To use the cubes, players must first download the free SiftRunner program, which lets users buy games, download them to their computers, and move them to the cubes.

San Francisco-based Sifteo—founded by Stanford University and MIT graduates David Merrill and Jeevan Kalanithi—touts the tactile nature of its platform. However, industry observers say the product faces several major challenges.

The cubes aren't really portable, as they won't function unless a computer is nearby. They work with a dongle that must be inserted into a computer's USB port. Sifteo says it is looking for ways to use a portable device, such as a smartphone or tablet, to work with its cubes.

Another challenge is that three blocks cost \$150, and additional blocks cost \$45 each.

Currently, only 13 Sifteo games are available, with some free and some costing up to \$5. However, the company says it is working on acquiring more, including some from major game developers.

Sifteo says its prime target market is children between the ages of six and 12.

Researchers Develop Electronic 'Tattoo' for Health Monitoring

Engineers have developed flexible electronic circuitry that can be mounted on a user's skin to perform tasks such as health monitoring.

The University of Illinois at Urbana-Champaign researchers said their epidermal electronic system (EES) includes electrodes, sensors, radio-communications and medical diagnostics components, and a user interface. The circuitry would be embedded in an ultrathin polymeric patch. The scientists hope to add Wi-Fi capabilities soon.

Their system links the electronics and sensors via nanoribbon and nanomembrane elements to allow the patch to be both small and flexible.

The first EES versions used silicon photovoltaic cells for electricity, but the small size limited the amount of power they could produce.

In an attempt to address this problem, the researchers are now experimenting with wireless induction, in which a nearby external source transmits power to a device without interconnecting conductors.

The EES could be used with medical applications such as electroencephalograms, electrocardiograms, and electromyograms to track nerve, heart, and muscle activity. Usually, EEG, EKG, and EMG testing—which measures electrical activity—requires that doctors attach electrodes to patients with adhesive tape and conductive gel, which generally limits their use to laboratory, hospital, or office settings.

On the other hand, medical personnel would only have to mount the EES onto an ultrathin piece of water-



The new Sifteo Cubes platform provides a novel approach to gaming via blocks that have LCD screens and that use near-field communications (NFC). Users play games with the blocks by physically moving them around and touching them to one another. The cubes' NFC sensors wirelessly communicate and recognize when they're close to one another and which sides are adjacent.

soluble plastic before attaching it to a patient's skin with water.

The researchers, led by professor John A. Rogers, say their tests yielded results comparable to those of traditional EEG, EKG, and EMG tests. And, they added, the EES captured data for up to six hours and could be used for 24 hours without losing effectiveness.

Rogers and his company, mc10 (<http://mc10inc.com>) are working to commercialize the technology.

Hackers Steal \$13 Million in One Day via Online Security Breach

Hackers recently hit the jackpot by using an innovative, complex, highly coordinated attack to breach the security of a large US debit-card processing company and steal \$13 million in just one day.

After compromising Fidelity National Information Services' (FIS's) debit-card database, they cloned 22 cards and sent them to co-conspirators in Greece, Russia, Spain, Sweden, Ukraine, and the UK. These participants then withdrew the money from various ATMs.

The intruders first broke into FIS's network and gained access to its database, where the company stores debit-card balances. To control potential losses, FIS limits the amount that account holders can take out during a 24-hour period and allows no withdrawals once users reach a card's prepaid balance unless they add more money to their account.

In the FIS database, the hackers evaded these restrictions by remotely removing or increasing the withdrawal limits on the 22 cards they cloned. And when the cards' balances got low, they remotely updated the accounts' balances, allowing further withdrawals.

Some security experts say the attack was one of the most complicated of its kind and is similar in some ways to those carried out in the past by hackers in Estonia and Russia.

ARTISTS BUILD MASSIVE LANDSCAPE WITH 65,000 CDS

A French artist and an architect finally figured out what to do with CDs now that so much of the world gets its music from digital downloads.

Artist Elise Morin and architect Clémence Eliard used 65,000 CDs to build what they call *WasteLandscape*. This metallic landscape will appear in several exhibitions.

During its first public display, the CD dunes sprawled over 1,640 square feet at Le Centquatre, a Parisian art space.

Morin noted that she and Eliard sewed the CDs together, creating a fabric. With helpers, they placed the fabric over inflatable plastic hills.

Morin said that *WasteLandscape* demonstrates the large impact a small everyday object can have and also addresses issues such as environmental problems and art's role in society.

"It is well known that CDs are condemned to gradually disappear from our daily life," she noted.

At one time, the CD was king of the music world. During the 1980s, recording companies moved from phonograph records to CDs, which Sony developed in the mid-1970s. By 1988, discs were outselling records and before long, records were hard to find.

In recent years, however, the music business has transitioned again. Now, users download music and store it on hard drives or flash drives. File sharing and devices such as the iPod have replaced CD usage.

As a result, CD sales have fallen about 50 percent from their peak in 2000.

According to Morin, when the last of the *WasteLandscape* exhibitions ends, the 65,000 CDs will be recycled into polycarbonates. These materials are commonly used in electronic applications such as capacitors, in product packaging, and in dome lights found on the inside and outside of vehicles.



Artist Elise Morin and architect Clémence Eliard used 65,000 CDs to build what they call *WasteLandscape*, shown here on exhibition at Le Centquatre, a Parisian art space. The metallic landscape will appear in several other shows.

The attack occurred earlier this year, but FIS didn't publicly report the incident until recently, in a quarterly earnings statement.

As a result of the attacks, FIS said 7,170 prepaid accounts might have been jeopardized and three cardholders' private information

might have been released.

The company said it has worked with the affected clients by, for example, blocking and reissuing prepaid cards. FIS also stated it is working to enhance security and assisting US law-enforcement agencies in their investigation.

NEWS BRIEFS

Report: Internet Explorer Will Have Less than Half of Browser Market Next Year

Microsoft's Internet Explorer, the world's most popular browser since 1999, will no longer control the majority of the browser market by June 2012, according to Net Applications, a Web analytics, applications, and consulting firm.

The company also predicted that Google's Chrome—released in 2008—will overtake Mozilla's Firefox—which debuted in 2004—as the second most popular browser.

Net Applications' study, based on market trends during the past 12 months, also said

- IE—released in 1995—has lost 6.9 percent of its market share.
- In August 2011 alone, IE lost 0.7 percent of market share, dropping to 55.3 percent overall.
- Most of IE's and Firefox's market share has gone to Chrome.
- Chrome's share rose from 7.8 percent to 15.5 percent.
- This August, Chrome's share increased by 1.2 points, its largest monthly increase ever.
- Firefox, long the second choice to IE, has seen its portion of the market drop by 0.2 percent to

22.6 percent overall. At its peak in April 2010, Firefox claimed 25.1 percent of browser users.

- Apple's Safari and Opera Software's Opera maintained their market shares at 4.6 percent and 1.7 percent, respectively.

Despite these findings, Microsoft said it is focusing on the success of IE9—its latest browser version, released last March.

However, Net Applications noted, this version's adoption rate slowed in August, compared to previous months. This coincided with the end last June of Microsoft's policy of automatically updating users to IE9.

IE8 is the most popular version of the company's browser. It holds a 30 percent overall browser-market share and accounts for 54.4 percent of all IE versions in use today.

To obtain browser usage data, Net Applications monitors 160 million unique visitors to 40,000 selected websites.

IBM and 3M Will Develop Adhesives for Building Powerful Multilayer Chips

IBM and 3M have announced plans to jointly develop adhesives that could enable the stacking of up to 100

chips, thereby creating microprocessors that run up to 1,000 times faster than today's versions. These processors could be used in high-powered servers, PCs, gaming devices, smartphones, advanced consumer electronics, and other applications.

Bonding individual chips is relatively common. However, IBM and 3M, say that gluing together large numbers of chips requires new types of adhesives that efficiently move heat through the stack of processors and away from sensitive components. They also want an adhesive that could coat multiple chips simultaneously.

"Our scientists are aiming to develop materials that will allow us to package tremendous amounts of computing power into a new form factor: a silicon skyscraper," said IBM vice president of research, Bernard Meyerson. "We believe we can advance the state of the art in packaging and create a new class of semiconductors that offer more speed and capabilities while keeping power usage low."

At some point, many experts say, chip manufacturers will reach the limit of being able to make chips more powerful and energy efficient by shrinking feature sizes. Instead, they argue, it will be necessary to stack multiple chips into a 3D array.

They say this would be more effective than linking an increasing number of chips arranged horizontally because the wires connecting stacked components would be shorter, thereby reducing signal transmission times and energy consumption.

For their upcoming project, IBM plans to create new semiconductor packaging processes, while 3M will develop and make the adhesives. The companies have not announced a timetable yet.

computing now

ACCESS | DISCOVER | ENGAGE

Let us bring technology news to you.



<http://computingnow.computer.org>
Subscribe to our daily newsfeed

Editor: Lee Garber, *Computer*;
l.garber@computer.org

GUEST EDITORS' INTRODUCTION



Where Software Engineering Meets ...

Carl K. Chang and David M. Weiss, *Iowa State University*

Mike Hinchey, *Lero—the Irish Software Engineering Research Centre*

Because software engineering is a relatively young field, research in the discipline benefits from the domain knowledge and wisdom accumulated outside its specific realm.

Software engineering is a relatively young field. When the term was first coined in 1968,¹ many other engineering disciplines had already existed for many years. Since then, software has become pervasive in almost every aspect of our modern life, and SE has become increasingly prominent as a highly practical and widely practiced field.

Because of its youth, SE has yet to develop the depth of theory that older disciplines possess; therefore, in many ways, SE research benefits from the domain knowledge and wisdom accumulated outside its realm. The ubiquitous interaction between SE and other scientific and engineering fields is the theme of this collection of articles. However, we do not intend to create another forum for the decades-long debate on whether SE is a true engineering discipline: is SE = E?

THE INFLUENCE OF OTHER DISCIPLINES

What other fields of study have had a major impact on advancing the principles and practices of SE? Since its beginnings in 1968, SE has been provocatively influenced by “engineering” viewpoints. Critical thinkers who are strong advocates for elevating SE into a true engineering discipline have argued that when developing complex, but necessarily reliable, software systems, we must rigorously understand and faithfully apply pertinent engineering principles and practices. Many researchers, such as Harlan Mills and colleagues, who introduced the concept of clean-

room software engineering,² and Barry Boehm, who has emphasized the importance of SE process and economic models,³ have been exemplary in pushing the envelope to the extent that the SE discipline has pragmatically adopted many engineering terminologies and methods.

At the profession level, some may accept the notion that across the traditional engineering landscape we should embrace SE as a newcomer that is a field of intense study. Many other fields of study, while not necessarily regarded as professions per se, have noticeably influenced SE research and practice.

As software includes many facets of human work in a large spectrum of professional and creative activities, SE becomes indispensable to various human undertakings, including petroleum engineering, aerospace engineering, automotive engineering, space exploration, climate control, environmental protection, national security, smart buildings and cities, finance and economics, and health-care. In these substantial human endeavors, software is increasingly the dominant controller, with widespread proliferation; many such systems require high reliability and are mission or life critical. It is a common belief that SE must assume its bona fide duty to tame complex work that could result in unaffordable or disastrous failures. However, many of us are convinced that SE is not a panacea.

IN THIS ISSUE

Because of space limitations in this theme issue, we can only provide a sample of a small fraction of the computing world to inform *Computer's* readers about where and how SE meets other fields. Specifically, authors who are well known in SE have contributed articles to explain the interactions between SE and a very fundamental issue—the theory that sustains SE study—as well as open source

GUEST EDITORS' INTRODUCTION

software (OSS), evolutionary computation (EC), space exploration, and services and cloud computing.

In “Can Practitioners Neglect Theory and Theoreticians Neglect Practice?,” Manfred Broy points out that our goal is to develop and evolve “great software.” But what makes software “great?” Many development methods and approaches are based on good principles, but without a theory, it is impossible to evaluate them for qualities such as efficiency and effectiveness. Broy argues that no engineering discipline can hold without an underlying theory, and software engineering needs theory too. He emphasizes that theory does not equate with the use of formalisms, which many developers have found to be impractical.

Brian Fitzgerald considers the seeming contradiction that open source software is popular because it is considered to be of high quality, delivered quickly and for free, using the best developers, yet it seems to breach all of the tenets of modern software engineering. Nevertheless, in “Open Source Software: Lessons from and for Software Engineering,” Fitzgerald points out that OSS benefits greatly from two important SE principles—modularity (information hiding) and configuration management. He also describes how many of the important areas in modern SE originated in OSS.


Mark Harman is well-known for having coined the term *search-based software engineering*. Yuanyuan Zhang, a member of Harman's research group, maintains an open online repository that preserves and makes available prior and current SBSE research exploiting the use of evolutionary computation techniques in the study of SE (http://crestweb.cs.ucl.ac.uk/resources/sbse_repository). In “Software Engineering Meets Evolutionary Computation,” Harman provides a broad-based yet insightful assessment of the significant influence of EC on SE. Those curious to know how EC interacted with SE research in the past two decades as well as how EC might continue to find its applications in SE study will find much useful information in Harman's contribution to this theme issue. As he suggests, and we concur, optimization issues are inherent in many SE problems, and SE professionals definitely should not overlook advances in EC.

When a spacecraft is light-minutes or light-hours distant from us, its utility and survival must depend on the software that controls it. Such software must be reliable, updatable, and adaptable to a changing environment in situations where humans cannot provide real-time control. In “Software Engineering for Space Exploration,” Robyn Lutz addresses the role that SE played and continues to play in the development of such software. Not only must the software be reliable, able to operate autonomously in real-time, and adaptable to exceptional conditions, it must also be maintainable over many years as the human team maintaining it changes and evolves. The SE process that NASA and other organizations use must preserve critical

knowledge about the software over its often decades-long lifetime while ensuring that the software meets a space mission's critical requirements.

In “Software Engineering Meets Services and Cloud Computing,” Stephen S. Yau and Ho G. An provide a concise introduction to the intersection issues between SE and these two computing paradigms. This article encapsulates numerous key concepts in services and cloud computing, and it enumerates the most critical, fast-emerging SE challenges that our readers are acutely facing today.

Finally, in “Software Engineering—Missing in Action: A Personal Perspective,” David Lorge Parnas offers his view of what engineering and science are and shares his observations concerning why SE is not yet an engineering profession. Parnas envisions what SE needs to do to become an engineering discipline and notes some of the current impediments to progress in achieving this vision, offering examples from the other articles in this issue.

We are enthusiastic about this collection of articles on a profound subject, and we anticipate that readers will also find them both informative and interesting. The reality is that SE has become a highly sought-after career, and the debate about SE ≠ E is necessarily a key concern to many SE professionals. In the current job market, analysts often label SE as one of the top professions for college students to consider as a career. As educators, we hope that future SE graduates will have the opportunity to read the articles included in this theme issue, and that doing so will help them understand and appreciate the interaction issues between SE and other fields. 

References

1. P. Naur and B. Randell, eds., *Software Engineering: Report of a Conference Supported by the NATO Science Committee*, NATO, 1968.
2. H. Mills, M. Dyer, and R. Linger, “Cleanroom Software Engineering,” *IEEE Software*, vol. 4, no. 5, 1987, pp. 19-25.
3. B. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

Carl K. Chang, a professor and chair of the Department of Computer Science at Iowa State University, is Computer's editor in chief emeritus. Contact him at chang@iastate.edu.

David M. Weiss is the Oanh and Lanh Nguyen Professor of Software Engineering in the Department of Computer Science at Iowa State University. Contact him at weiss@mail.iastate.edu.

Mike Hinchey is the director of Lero—the Irish Software Engineering Research Centre and a professor of software engineering at the University of Limerick, Ireland. Contact him at mike.hinchey@lero.ie.

COVER FEATURE



Can Practitioners Neglect Theory and Theoreticians Neglect Practice?

Manfred Broy, *Technical University of Munich, Germany*

Theory helps engineers in other disciplines create and understand methods, evaluate results, and optimize processes. Does it have a role in software engineering as well?

Software engineering is by definition part of a practical discipline,¹ but scientific insight and knowledge are not its primary measures of success—rather, its practitioners are chiefly focused on building and further evolving software systems in a timely manner at a reasonable cost and quality. The measure of success of the discipline is the extent to which it provides concepts, methods, techniques, and processes that allow developers to master the task of software evolution. In addition to software development and maintenance, this task presents diverse challenges, including requirements analysis, specification architecture, implementation, integration, and validation.

Engineering disciplines must be based on scientific practices and theory to justify their approaches and to give scientific evidence for why and where their methods work properly. Fortunately, many different kinds of scientific methods—including those related to theoretical concept analysis, empirical and experimental research, as well as related concepts from disciplines such as psychology and economics—can help with this task. Researchers can create, adapt, and further develop these scientific methods to more properly support the discipline of software engineering.

CAN THEORY IMPROVE SOFTWARE ENGINEERING?

A recent article by Ivar Jacobson and Ian Spence on theory in software engineering included the following thought-provoking statement: “Our greatest challenge: understanding how to build great software.”² But what exactly is “great” software? Do methods exist to distinguish between great and not-so-great software? Even simple questions like these use a theory of quality as the basis for software construction and evaluation.


Looking more closely at methodology, if the task of software engineers is to build and further evolve software systems, and if the methodological and technical maturity of software engineering is insufficient, who is responsible for improving development methods? Research and improvement initiatives from both industry and academia have sought better methodology in software engineering, but they have encountered several challenges:³

- establishing precise and adequate terminology (for instance, better terms than “great” for addressing software quality);
- developing techniques for building software of adequate quality;⁴ and
- identifying methods for measuring, judging, and evaluating both the software itself and the effectiveness of the processes, concepts, and tools used to develop it.

Suggesting new engineering methods without their careful justification is of limited help. Unfortunately, researchers often advocate methods without offering

COVER FEATURE

substantial analysis or evidence as to how well, under which constraints, and why they work. For example, the so-called *agile manifesto* states several principles that seem to have promise, but it does not provide scientific evidence of the extent to which agile methods actually help with building software of adequate quality. Software engineering needs theory to evaluate the efficiency and effectiveness of development techniques, which would ultimately result in justified and empirically verified methodological knowledge.



The more complex an engineering discipline is, the more important its theoretical foundation—and software can become very complex.

CRAFT, SCIENCE, OR ENGINEERING DISCIPLINE?

Identifying the differences between craft, scientific theory, and engineering helps in understanding the role of theory in engineering:

- *Craft* applies traditional techniques to produce goods and provide services.
- *Science* aims to gather, verify, and document knowledge and insight through research.
- *Engineering* applies both knowledge and scientifically analyzed and justified methods to develop and produce technical products.

Applying scientific theories to create, analyze, and justify methods and techniques is essential to move something from craft to engineering. For craft, it is not important to understand why methods work or how they could be proven and optimized; it is sufficient if they produce satisfactory results. In contrast, for engineering, the questions of how to construct new methods systematically, why different techniques work, and how scientific methods can justify, relate, prove, and optimize them are key.

Craft and engineering have much in common because they have similar overall goals: creating constructive solutions to practical problems in terms of products and services. Consequently, the separation of craft from engineering might not seem clear—for example, some might say that software development as practiced today is to a large extent a craft. However, by definition, craft applies traditional techniques. Because software development is still a young field, it does not have a large number of such techniques. In some respects, the discipline works in an ad hoc manner, applying “best practices” that often

cannot be called well proven. The evolution of traditional techniques requires time, and the software field is still experiencing fast-moving technical progress.

Other subtle differences emerge as well. In contrast to traditional craftsmanship, which typically involves smaller groups, software projects and their budgets can be massive. In addition, development teams for huge, complex software systems must be managed using organizational team structures and management processes. Further challenges arise with the development of large families of software systems—to evolve product lines and families strategically requires something more rigorous than typically found in craft, which brings us full circle to theory.

To successfully accomplish complex tasks, theory proves indispensable. The more complex an engineering discipline is, the more important its theoretical foundation—and software can become very complex.

WHY THEORY?

Successful software engineering requires insights into many aspects of software and its evolution, such as

- methodology, including requirements engineering and specification, architecture and design, code quality, integration and verification, deployment and migration, and modification and improvement of software systems;
- organizational competency and management of software evolution;
- domain-specific knowledge;
- technical information about hardware;
- human-machine interaction; and
- the economic aspects of software, marketing, and entrepreneurship.

Software engineering also deals with numerous abstract concepts that are often hard to understand because of the imprecise terms used to describe them, such as *modularity*, *compatibility*, and *tracing*. These notions often appear in publications or in practice without establishing a consistently clear understanding of what they mean or where and why they are important. This is why theory matters—it helps determine and evaluate the concepts that provide the basis for identifying terminology and developing engineering methods.

Theory and methods

An engineering discipline without a theory cannot work. But software engineering also requires methods to help solve the problems that arise when developing and evolving a software system. For methods to be effective, researchers must evaluate, justify, and classify them to determine how they work best and to identify their limitations.

Like other disciplines, software engineering has many facets, such as people, management, and teams; economy and cost; methodology and development processes and tools; human-machine interaction factors; and actual technology—hardware, operating systems, and programming languages.

Researchers can use theory to interpret and explain obstacles in engineering practice and provide constructive foundations for engineering methods and tools. How much theory and methods can contribute to engineering is evident in the field of compiler construction. Applying theory systemically in this field using context-free grammars and compiler generators has turned it into one of the first mature areas of software construction.

Logical theories also play a prominent role in some aspects of software engineering because developers can use mathematical logic to deal with the formality. Research in formal techniques is indispensable and can lead to insights, perhaps finally resulting in solid engineering methods. However, there is a considerable gap between conducting research on formalization and formal theories, and developing tractable methods. Of course, such methods will also require a careful analysis of scalability, tractability, applicability, and efficiency as well as effectiveness.

In spite of the reported benefits of applying formal methods to some areas of software engineering, some counterarguments still exist as to their practicality.⁵

Formal methods

- do not scale, per se;
- are too difficult for engineers who are not properly trained;
- if not carefully designed, do not capture essential aspects; and
- are not cost-effective if they are not well integrated into development processes.


Perhaps part of the problem is that when the term *formal theory* is mentioned in software engineering circles, *formal methods* come to mind. But reducing formal theory to formal methods is both too narrow and misleading. Formalization is a scientific method that can help analyze, validate, and create engineering methods, but formal methods should not be regarded as a silver bullet for software evolution. As David Parnas says, “Our role model should be engineers, not philosophers or logicians.”⁶

Theory and education

Educating software engineers without giving them theoretical foundations is unthinkable. Even if some theories are not immediately applicable to practical engineering, they give a framework of understanding. One example is

assertion logic. Even if formal proofs of programs through assertions are not used in practice, understanding the ideas of Hoare triples, loop invariants, and termination proofs is a valuable conceptual exercise. The same applies to algebraic data types, relational data models, and modular models of software architectures.

It is essential, however, to teach theory the right way. Students need to get a deep understanding of the basic engineering concepts and their theoretical justification to relate them to practical tasks. Applied topics such as operating systems, protocols, and databases should also be taught and explained in terms of adequate theories.



Logical theories also play a prominent role in some aspects of software engineering because developers can use mathematical logic to deal with the formality.

Software evolution is an engineering discipline based on informatics—the scientific discipline of information and information processing. Several related aspects of theory are important for software engineering.

Classical areas such as formal languages, computability, and complexity theory are obviously important for software engineers. How can researchers develop ambitious software systems without understanding the limits of computability and decidability? How can they deal with performance without understanding computational complexity? How can they design software engineering tools for modeling or programming without understanding formal languages?

Techniques for describing the semantics of programming languages, program specification, and program verification are parts of a theory of programming in the small. Certainly, programming in the small is a basis of software engineering, but it is not software engineering per se. Indeed, software engineering is concerned with programming and developing in the large. Some theories directly address the foundations of programming in the large and lay the foundations for software engineering.⁷ Examples include ways of modeling and mastering key aspects of software systems such as requirements, architecture, interfaces, composition, and quality.


In addition to the foundational theories of informatics, software engineering also must be based on more general subfields of informatics such as the theory of data types, databases, hardware structures, operating systems, protocols, and so on.

Although using formal techniques might not always be directly practical in software engineering, they help in

COVER FEATURE

understanding the basic software engineering questions and concepts. How could we understand notions such as correctness, specification, or verification without appropriate theory? Even if in practice such techniques often are not applied in the rigorous way formal methods suggest, at least understanding them is useful.

Another interesting foundation of software engineering is mathematical logic, a field that has developed completely independently of software engineering. There is a close relationship between computing and mathematical logic, and many software engineering issues are closely related to logic. The reason is obvious: software is by nature a formal artifact. Because it is formal, computing machinery can execute software with interpretations independent of human beings. Hence, software exhibits behaviors and



Software engineers apply both scientific—mathematical, economic, and social—and practical knowledge when they design and build software systems.

properties that are subject to formal analysis, even if a programming concept is not formalized or if software is written in ad hoc languages and executed on machinery by ad hoc interpreters.

That said, software is radically different from other technical artifacts. In contrast to conventional engineering disciplines, software has no material nature—it is pure logic. It describes dynamic behavior by static means using rigorous formalisms such as programming languages.

However, in practice, software's entirely formal nature is obscured by the technical consideration of execution environments, which have many different mechanisms and contribute to the generation of highly complex behavior. Software is tightly connected with its environment, and the connected electronic, mechanical, and organizational processes usually are not formal, thereby requiring a more comprehensive view of software systems as formal entities, separate from their operational context. This split requires two complementary views: a formal one that captures the interaction between interface and software system contexts, and a pragmatic view directed at the impact of the software on its context.

THEORY IN SOFTWARE ENGINEERING RESEARCH

Software engineers apply both scientific—mathematical, economic, and social—and practical knowledge when they design and build software systems. Theory

can help engineers avoid repeating the same mistakes when building new systems. It is depressing to see the extent to which well-established theoretical principles are ignored in pragmatic work, such as in developing UML⁸ and Java. One of the frustrations of the field is that we have not managed to bring together theoretical knowledge and practical work in a way that offers a mutual benefit.

Insufficient knowledge about practice in academia and limited competency in practice are key factors in the failure to bridge this gap. Most companies have narrow views about the field of software engineering—even about how to use software processes and the basic rules and principles of the field. Accordingly, it is difficult for them to see the value in theory-based methods. Moreover, due to the lengthy amount of time it takes to introduce advanced incremental theory-based techniques, even if they contribute to cost savings and quality in the long run, it is difficult for nonspecialists to recognize their methodological potential. This is, of course, reinforced in cases in which we do not have enough evidence that approaches based on theory actually contribute practically. In fact, introducing advanced engineering methods brings risks even for management with expertise.

Nevertheless, software engineering theories have made enormous progress in the past 40 years, in particular those related to specification and verification, architecture and design, testing, software project management, and software economics. However, we are still far from comprehensive and established theoretical foundations.

Much more work on theory is needed to master the enormous future challenges related to software evolution, when we can easily foresee having to construct large software systems, software families, or cyberphysical systems. Theoretical foundations will ultimately prove indispensable for dealing with the inevitable behavioral and nonfunctional properties related to safety, security, reliability, usability, and maintainability, especially for highly distributed, iterative networks of software systems.

So why are the results of theoretical software engineering research not applied more often in practice? In part, the answer to this question has to do with the attitudes of researchers working in this field. There are several different approaches to this research.

Theory based on practical observation and experience

In this approach, researchers try to discover theories that help them understand, interpret, and deal with observations and obstacles they have discovered in practice. A typical example would be the pointer and reference structures found in many programming languages. Developing a theory of pointer structures for reasoning about programs with languages like Pascal (with its typical pointer

structures) or with object-oriented languages like Java (with its object identifiers, which are essentially nothing more than pointer structures) is a challenge. Accordingly, a typical scientific approach would try to work out a theory and then use that theory to help in the classical tasks of program specification, analysis, and verification. Further examples would include formalized techniques that deal with architectures of software systems.

Theory in isolation

Another approach is independent observations of practice to develop helpful scientific theories that hopefully can later contribute to a better understanding of the practical issues of software engineering. In this case, researchers work independently from practical issues and just concentrate on getting the theory into clearer form.

From ad hoc methods to theory


This third category characterizes approaches in which practitioners encounter practical problems and then develop ad hoc methods to deal with them. Later, they might try to provide a theory for them. Examples include UML and architectural description languages. Both show that ad hoc solutions often must change fundamentally after careful analysis because ad hoc method constructions typically fail to get everything right in the first cut. This usually becomes evident when working out the theoretical foundations, further indicating that giving foundations to ad hoc methods exposes their shortcomings and superficial complexity.⁸

Obviously, there are crucial differences between these three approaches. It is very dangerous to develop theory and claim that it is practically useful without being knowledgeable and working in close relationship with practitioners. A more promising approach is to work in iterations between steps in developing theory and any attempts to apply them in practice. Doing useful theory requires sufficient capabilities in performing theoretical work. Not all software engineers with deep practical insights and understanding are knowledgeable enough in scientific work to express their insights in adequate theory.

ADDRESSING PRACTICAL QUESTIONS

In the long run, theory for engineering must address practical questions. We do not need theorists who work out beautiful calculi that are too good for the dirty work of practice but claim to contribute to software engineering and its practical problems. Similarly, working out an algebraic calculus and deciding about certain properties independently of engineering issues is dangerous because issues of aesthetics and scientific quality for algebraic calculi might not necessarily coincide with practical needs and tractability in the real world.

A typical example explains this viewpoint. In the 1980s, based on the observation that some of the deduction rules of linear logic showed similarities to phenomena in concurrent systems, some researchers claimed that linear logic would contribute to concurrency theory. However, in the end, linear logic could not contribute at all either to concurrency theory or to the practice of engineering concurrent and distributed software systems. Another example is the POM sets by Vaughn Pratt.⁹ Although they presented an interesting step into understanding the structure of concurrent processes, delving deeply into the details of POM sets and Chu spaces proved to be useless in practice, in the end contributing nothing to resolving engineering challenges.



As long as theoreticians do not take into account issues from practice, their work will not be very helpful from a pragmatic viewpoint and will not find an immediate road into practice.

In fact, much of the very basic work and theory is not properly related to practical issues. An example is the field of process algebras, a branch of theoretical informatics that was an active research field a few years ago. Certainly, there is a rich world of process algebras similar to the mathematical groups as studied in group theory. But most pure branches of theoretical informatics study different kinds of process algebras completely independent of any questions related to software engineering practice.

Of course, it is rewarding to work out concepts and their underlying theories to support certain software engineering tasks, such as specification, design, and verification. But this can be done with a mathematical attitude leading to instances of theory that contribute to the understanding of engineering and give a proof of concept for engineering methods. As long as theoreticians do not take into account issues from practice, their work will not be very helpful from a pragmatic viewpoint and will not find an immediate road into practice. There are several reasons why: software engineering lives in a dirty and imperfect world, and many compromises must be accepted in practical software evolution, not just because of a lack of education of the engineers doing the work but also due to many obstacles encountered in practice, including hardware, tools, and organizational structures. Bringing proper theory into the unsystematic world of practice is a difficult endeavor.

Theories that characterize essential engineering concepts through useful relationships between levels of

COVER FEATURE

abstraction in architectures, for example, those based on Galois connections, could contribute to practice. Galois connections are an interesting and deep concept in algebra, but both theoreticians and engineers can find them difficult to understand. However, proving levels of abstractions that form Galois connections could be a helpful observation because it allows choosing levels of abstraction independently and relating them methodologically, bringing together theoretical and tool support issues.

As a matter of principle, engineering disciplines are based on their own scientific foundations. Electrical engineering and mechanical engineering are based on physics. Software engineering is based on logics and theories of information and computation.

But in addition to a theoretical foundation, we need a joint understanding and agreement that, in the long run, software engineering cannot be established without a solid body of scientific theory. In fact, software engineering requires an understanding of theory, what it can offer, and its limits. This requires a comprehensive understanding of practice, its needs, and its open challenges. A structured presentation of theory and its connection to the engineering of software systems is a must in software engineering education. **□**

Acknowledgments

I thank Benedikt Hauptmann and Holger Pfeifer for reading and commenting on drafts of this article.

References

1. P. Naur and B. Randell, *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee*, NATO, 1968.
2. I. Jacobson and I. Spence, "Why We Need a Theory for Software Engineering," *Dr. Dobbs J.*, 2 Oct. 2009; <http://drdobbs.com/architecture-and-design/220300840>.
3. M. Broy, "The 'Grand Challenge' in Informatics: Engineering Software-Intensive Systems," *Computer*, Oct. 2006, pp. 72-80.
4. I. Jacobson and B. Meyer, "Methods Need Theory," *Dr. Dobbs J.*, 6 Aug. 2009; <http://drdobbs.com/architecture-and-design/219100242>.
5. R.A. De Millo, R.J. Lipton, and A.J. Perlis, "Social Processes and Proofs of Theorems and Programs," *Comm. ACM*, vol. 22, no. 5, 1979, pp. 271-280.
6. D.L. Parnas, "Really Rethinking Formal Methods," *Computer*, Jan. 2010, pp. 28-34.
7. M. Broy, "Toward a Mathematical Foundation of Software Engineering Methods," *IEEE Trans. Software Eng.*, 2001, pp. 42-57.
8. M. Broy and M.V. Cengarle, "UML Formal Semantics: Lessons Learned," to appear in *Software & Systems Modeling*, 2011.
9. V.R. Pratt, "Chu Spaces and Their Interpretation as Concurrent Objects," *Computer Science Today: Recent Trends and Developments*, LNCS 1000, J. van Leeuwen, ed., Springer, 1995, pp. 392-405.

Manfred Broy is a professor in the Department of Informatics at the Technical University of Munich, Germany. His research interests focus on the theoretical and practical aspects of software and systems engineering. Broy received a Habilitation degree in informatics from the Technical University of Munich. Contact him at broy@in.tum.de.

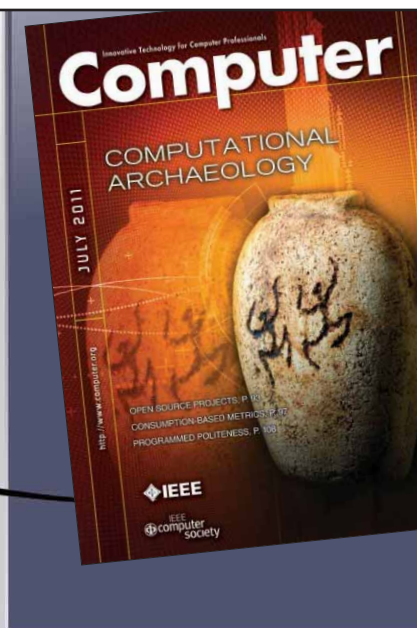
TIMELY, ENVIRONMENTALLY FRIENDLY DELIVERY

DIGITAL EDITIONS

Keep up on the latest tech innovations with new digital editions from the IEEE Computer Society. At **more than 65% off regular print prices**, there has never been a better time to try one. Our industry experts will keep you informed through a format that's timely, easy to search and save, and environmentally friendly.

- Email notification. Receive an alert as soon as each digital edition is available.
- Two Formats. Choose the enhanced PDF edition OR the web browser-based edition.
- Quick access. Download the full issue in a flash.
- Convenience. Read your digital edition anytime —at home, work, or on your mobile.
- Digital archives. Subscribers can access the digital issues archive dating back to January 2007.

Interested? Go to www.computer.org/digitaleditions to subscribe and see sample articles.



COVER FEATURE



Open Source Software: Lessons from and for Software Engineering

Brian Fitzgerald, Lero—the Irish Software Engineering Research Centre

Despite initial suggestions to the contrary, open source software projects exhibit many of the fundamental tenets of software engineering. Likewise, the existence of category-killer apps suggests that conventional software engineering can draw some lessons from OSS.

Open source software can elicit strongly contrasting reactions. Advocates claim that OSS is high-quality software produced on a rapid time scale and for free or at very low cost by extremely talented developers. At the same time, critics characterize OSS as variable-quality software that has little or no documentation, is unpredictable as to stability or reliability, and rests on an uncertain legal foundation—the result of a chaotic development process that is completely alien to software engineering's fundamental tenets and conventional wisdom.

Research suggests a more balanced view. On one hand, OSS is not the “silver bullet” championed by its most vocal partisans. On the other hand, it does not radically diverge from traditional software engineering practice as its severest detractors claim, and, as evidenced by some notable successes, OSS offers many tangible benefits.

OSS AS A SILVER BULLET

Twenty-five years ago, IBM software engineer Fred Brooks famously contended that “there is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement within a decade in productivity, in reliability, in simplicity.”¹ However, many claim that OSS is indeed such a silver bullet.

Defenders argue that OSS, beyond its obvious cost advantages, is of very high quality. Contributors to OSS projects are in the top 5 percent of developers worldwide in terms of ability, and are self-selected and thus highly motivated. Furthermore, the testing pool is global, and peer review is truly independent.

Another key advantage cited is the rapid development time of projects. The OSS community has taken odds with Brooks' law—namely, that “adding manpower to a late software product makes it later,”² a conclusion based on his experience managing development of the IBM OS/360—by endorsing Linus's law: “Given enough eyeballs, every bug is shallow.”³


There are many examples of OSS products of exceptional quality and reliability across a range of application domains—indeed, “category killers” such as the Linux kernel and Apache webserver perform so well that there is no market for an alternative.

COVER FEATURE

NOT SO FAST

Critics concede that a staggering mélange of OSS products is readily available for free download, but they claim it is virtually impossible to predict the usability, stability, and reliability of these products. The uneven quality is not helped by a lack of documentation and the reliance on support and upgrades from a voluntary community who must be convinced to accept changes to suit specific circumstances. These flaws are exacerbated by a complex licensing situation in which even the lawyers cannot definitively resolve IP rights issues.⁴

Furthermore, OSS arises from a development process that seems to flout traditional best practices. For example, typically there is no real formal design process, no risk assessment or measurable goals, often no direct monetary incentives for developers or organizations, informal coordination and control, and much redundancy as tasks are duplicated in parallel initiatives. All of this is anathema to conventional software engineering.



Sound software engineering principles such as a modular architecture and sophisticated configuration management are very much at the heart of successful OSS projects.

Other analyses of OSS say that 30 years of prior software engineering research cannot be discounted so easily. The claims in relation to the quality of OSS products and of community feedback are particularly questionable when exposed to scrutiny.

Quality

A study by Ioannis Stamelos and colleagues assessed quality issues in the SuSE Linux 6.0 release.⁵ Using the Logiscope code analysis tool, they examined more than 600,000 lines of code across 100 modules and found that only 50 percent were acceptable. Of the remainder, 31 percent required comments, 9 percent required further inspection, 4 percent required further testing, and 6 percent needed to be completely rewritten. These results are quite average in the software industry: only half of all modules meet generally accepted standards.

In a similar vein, Srdjan Rusovan, Mark Lawford, and David Parnas studied the implementation of the Address Resolution Protocol in the Linux TCP/IP implementation and identified numerous software quality problems.⁶

Community feedback

The claim of high-quality feedback from the OSS community is also questionable. A study of OSS development

by Niels Jørgensen revealed that while simpler code gets more feedback, it is generally not all that useful.⁷ A sort of inverse Pareto principle is likely at work, in that 99 percent of OSS developers spot 80 percent of the bugs, but only about 1 percent of the developers can identify the more difficult 20 percent. Furthermore, the Jørgensen study showed that there was very little feedback on design issues, a significant deficiency.

Also, the fact that OSS is the choice of the technologically literate could be problematic. On his OS/2 Headquarters website, Tom Nadeau argued that proprietary software vendors always gear their software to “the most ignorant customers,” while OSS developers cater to the “smartest customers” and can thus cut back on niceties such as a user-friendly interface.⁸ This phenomenon appears to be somewhat borne out by the comments of one Linux user who, after installing the OS, posted a message referring to the “thrilling adventure” of the installation.

LESSONS FROM SOFTWARE ENGINEERING

In light of these critiques of OSS, it is worth considering the lessons and principles that OSS has drawn from software engineering. It is readily apparent that sound software engineering principles such as a modular architecture and sophisticated configuration management are very much at the heart of successful OSS projects.

Linux offers one demonstration of the importance of *modularity* in OSS. Linux benefited greatly from the elimination of defects and fleshing out of requirements in Unix.⁹ Indeed, the manner in which different individuals take responsibility for various self-contained modules within Linux is acknowledged as a major factor in its successful evolution.

The Sendmail utility offers additional evidence of the role of modularity in OSS. Sendmail was first developed in the late 1970s at the University of California, Berkeley, by Eric Allman, who made the source code available to all interested parties. However, when problems in integrating these efforts emerged as the utility began to evolve through others' contributions, Allman rewrote Sendmail to follow a more modular structure. This ensured that the program would be a suitable candidate for massive parallel development, a characteristic of OSS, as developers could largely work independently on different aspects. Sendmail is now the dominant internet e-mail router, handling an estimated 80 percent of all Internet e-mail.¹⁰

The modular approach applies to project structure as well as the code base: large OSS projects tend to be aggregations of smaller projects.¹¹ This allows developing, fixing, and releasing components more independently.

Configuration management is likewise a vitally important factor in OSS, and several sophisticated tools exist for this purpose. In addition, the software engineering

principles of *independent peer review and testing* are highly evolved within OSS.

In short, the code in OSS products is often very structured and modular, and developers carefully vet and incorporate contributions in a disciplined fashion in accordance with good configuration management and independent peer review and testing. OSS development does not depart significantly from many sensible and proven software engineering principles, and it is simplistic to characterize OSS as a “bazaar” with an undisciplined development process.

LESSONS FOR SOFTWARE ENGINEERING

Despite overblown hype at times, there are undoubted and notable OSS successes. OSS can contribute much to software engineering knowledge, including open innovation, global software development, inner source, and time-based release management.

Open innovation

Open innovation has become a holy grail in organizational endeavors, including software development. Recognizing that no single organization will have a monopoly on creative people, open innovation seeks to leverage ideas from a wider, ideally global, talent pool.

Certain characteristics are important stimulants to innovation, including

- *autonomy*, which forms the basis for self-organizing and increases the possibility that individuals will motivate themselves to form new knowledge;
- *creative chaos*, whereby individuals do not have to follow organizational rules but are challenged to investigate alternatives and rethink assumptions;
- *information redundancy*, whereby individuals have information that goes beyond their immediate needs for a particular task; and
- *requisite variety*, whereby individuals have the diverse skills needed to match the complexity and variability of the environment they face.¹²

All these characteristics are readily found in OSS communities. Developers tend to self-select and are largely autonomous in relation to the tasks they undertake. Given that most OSS developers work outside organizational boundaries, creative chaos can exist. The openness of the code at mature points in the development process facilitates information redundancy. And the cosmopolitan nature of OSS developer communities ensures requisite variety.

Much of OSS obeys a power law.¹³ An interesting property of power-law distributions is that they do not have a peak at the average—hence they scale. This is evident in typical OSS projects. For example, while some might

suggest that Firefox has too many developers,¹⁴ several hundred thousand people use test versions of the browser, and about 20 percent take the time to contribute bug reports. This pool of users is an extremely useful resource.

OSS has also been a source of inspiration in terms of innovative business models. One model is to offer a free open source version of a proprietary product that entices customers to purchase the enterprise version with some additional functionality.¹⁵ Also, innovations and new features emerge from the OSS community’s creative mindset.

As Eric Raymond memorably observed, most OSS developers have “a personal scratch to itch.”⁵ It is thus no accident that many successful OSS products are general purpose. Given Jørgensen’s finding that feedback on design issues in OSS development is rare,⁷ it appears that OSS is best suited to horizontal domains in which there is widespread agreement on the design architecture and the general composition of the software requirements is



OSS can contribute much to software engineering knowledge, including open innovation, global software development, inner source, and time-based release management.

fairly well known and unproblematic. This is probably essential with a large base of contributors from a wide variety of industrial and academic backgrounds. On the other hand, in vertical domains where requirements and design issues are a function of specific domain knowledge that can only be acquired over time—the case with many business environments—there are not likely to be as many OSS offerings.

Given OSS’s potential for innovation, it is ironic that many early efforts replicated proprietary software products. However, unique features originated in these OSS clones that were typically ported back into their proprietary counterparts.

Global software development

GSD dramatically increases coordination, communication, and control challenges in software development.¹⁶ Given the current trends of outsourcing and globalization, GSD is an issue of increasing significance for organizations today.

OSS resolves coordination issues in GSD with simple communication tools—e-mail, newsgroups, and version control systems.¹⁷ The “secret sauce” seems to lie in the coordination structures present in OSS. At the center is a team of experts with varied experience who tend to coordinate their work informally but are aware of one

COVER FEATURE

another's expertise. This relatively small core group does the vast majority of coding, but it is complemented by larger teams of bug-fixers and testers drawn from the user population. The latter boost productivity and reduce defect density but do not add interdependencies, as finding and reporting bugs does not involve code changes. Consequently, several studies have reported efforts to transfer OSS lessons to GSD within organizations.^{17,18}

Inner source

The phenomenon of adopting OSS practices within a corporate setting is known as *inner source*,¹⁹ also called corporate open source²⁰ and progressive open source.²¹ While there is no standard set of OSS practices, some common ones include open sharing of source code, large-scale independent peer review, the community development model, and the expanded role of users.²² Leveraging a product's users as codevelopers can improve

Several OSS projects have radically changed their release management processes and moved to a time-based strategy.

quality and generate specialized new features that are important to a wider audience.²³ Although OSS practices are generally more applicable to large organizations due to their inherent geographic distribution, smaller organizations can also benefit from OSS development practices.²⁴

Companies usually employ inner source to capitalize on the success of certain open source projects. However, there are important differences between open source and closed source development and their respective communities.²⁵

In traditional software development, developers and user testers are typically in separate departments or locations. This can lead to employees being unaware of other projects and innovations, all too frequently resulting in a lack of mutual respect or voluntary interaction.

While early open source developers were users of actual products, as OSS has evolved, the situation has changed. In the absence of a traditional software development company, users need to become more intimately involved in the development process, as technical staff cannot simply send a checklist of requirements to the vendor. It is a widely held belief that deploying open source can lead to a sense of shared adventure, which is not a common scenario in the proprietary software arena. Also, it has been reported that OSS developers take greater pride in their work and feel a greater sense of responsibility to deliver high-quality code because peers they truly respect will review their efforts.²⁶

Time-based release management

Release management has been the subject of little research in the software engineering field. Traditional models focused on the initial release of a software product and ignored subsequent releases,²⁷ but the industry now recognizes that a continuous-release strategy delivers both fixes and new functionality to users. This strategy also staves off obsolescence by maintaining the software's value.

The norm is to release a new version of software when it meets a specific set of criteria and has attained certain goals, usually features important to customers. In commercial software release management, this strategy requires delicate balancing as introducing a new release too early could erode the market share and revenue-generating potential of the existing one.²⁸

To mitigate risk from some OSS practices such as the lack of deadlines, the reliance on volunteers, and ad hoc coordination and management, numerous OSS projects appear to have formalized their release management process.²⁹ This is an important part of quality assurance because developers stop adding new features during the preparation for a release and instead focus on identifying and removing defects. The feedback obtained after a release also provides information about which parts of the software might need more attention.

In the case of OSS, however, it is not obvious how a team of loosely connected, globally distributed volunteers can work together to release high-quality software, some of which consists of millions of lines of code written by thousands of people, in a timely fashion. There is much evidence that this is a serious problem. For example, the Debian OS has increasingly experienced delays and unpredictability, with up to three years between stable releases. However, this pales compared to the compression utility gzip, with 13 years between stable releases (1993-2006).

Consequently, several OSS projects have radically changed their release management processes and moved to a time-based strategy. This approach sets a specific release date well in advance and creates a schedule so contributors can plan accordingly. Prior to the release, there is a cutoff date on which developers evaluate all features for stability and maturity and then decide whether to include them in the upcoming release or postpone them to the next one.

While the specific time-based approach differs from project to project, there is a common pattern of staged progress toward a release in which each stage is associated with increasing control over permitted changes. These control mechanisms are known as *freezes* because development is slowly halted. Freeze categories include

- *feature freeze*: no new functionality can be added—the focus is on removing defects;

- *string* freeze: no messages displayed by the program, such as error messages, can be changed—this allows translating as many messages as possible before the release; and
- *code* freeze: permission is required to make any change, even to fix bugs.²⁹

In modular component releases, developers can fix and release defective modules while using a time-based strategy to combine components and test the integrated product, as is the case with Debian and GNOME (GNU Object Model Environment).²⁹

The trend toward software as a service suggests that a release management strategy focused on big-bang features is not suitable, as customers prefer to obtain continuous improvements from a vendor website rather than buy a new shrink-wrapped product. A time-based release management strategy is ideal for regularly adding new functionality.

Open source software promises to be part of the software landscape for some time to come.³⁰ While the notion of OSS as a silver bullet might be an inaccurate stereotype, OSS projects clearly exhibit many of the fundamental tenets of software engineering. Likewise, the fact that OSS provides some category killer apps developed in a GSD context—recognized to be a complex development environment—suggests that conventional software engineering can draw lessons from OSS.

Acknowledgments

Thanks to Klaas-Jan Stol for providing useful feedback in the development of this article. Support for this work came from Science Foundation Ireland through its grant to Lero—the Irish Software Engineering Research Centre.

References

1. F.P. Brooks, "No Silver Bullet—Essence and Accident in Software Engineering," *Proc. IFIP 10th World Computing Conf.*, Elsevier Science, 1986, pp. 1069-1076.
2. F.P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1975.
3. E.S. Raymond, *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly Media, 1999.
4. K.-J. Stol and M.A. Babar, "Challenges in Using Open Source Software in Product Development: A Review of the Literature," *Proc. 3rd Int'l Workshop Emerging Trends in Free/Libre/Open Source Software Research and Development* (FLOSS 10), ACM Press, 2010, pp. 17-22.
5. I. Stamelos et al., "Code Quality Analysis in Open Source Software Development," *Information Systems J.*, Jan. 2002, pp. 43-60.
6. S. Rusovan, M. Lawford, and D. Parnas, "Open Source Software Development: Future or Fad?," *Perspectives on Free and Open Source Software*, J. Feller et al., eds., MIT Press, 2005, pp. 107-121.
7. N. Jørgensen, "Putting It All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project," *Information Systems J.*, Oct. 2001, pp. 321-336.
8. T. Nadeau, "Learning from Linux: OS/2 and the Halloween Memos," OS/2 Headquarters, 1999; www.os2hq.com/archives/linmemo1.htm.
9. S. McConnell, "Open-Source Methodology: Ready for Prime Time?," *IEEE Software*, July/Aug. 1999, pp. 6-11.
10. B. Costales et al., *Sendmail*, 4th ed., O'Reilly Media, 2007.
11. K. Crowston and J. Howison, "The Social Structure of Free and Open Source Software Development," *First Monday*, Feb. 2005; <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1478/1393>.
12. I. Nonaka, "A Dynamic Theory of Organizational Knowledge Creation," *Organization Science*, Feb. 1994, pp. 14-37.
13. G. Madey, V. Freeh, and R. Tynan, "The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory," *Proc. 8th Americas Conf. Information Systems* (AMCIS 02), Assoc. for Information Systems, 2002, pp. 1806-1813.
14. G. Hayes, "Firefox Has Too Many Developers," blog, 14 Dec. 2009; www.trollaxor.com/2009/12/firefox-has-too-many-developers.html.
15. P.J. Agerfalk and B. Fitzgerald, "Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy," *MIS Q.*, June 2008, pp. 385-410.
16. P.J. Agerfalk and B. Fitzgerald, "Flexible and Distributed Software Processes: Old Petunias in New Bowls?," *Comm. ACM*, Oct. 2006, pp. 26-34.
17. A. Mockus and J.D. Herbsleb, "Why Not Improve Coordination in Distributed Software Development by Stealing Good Ideas from Open Source?," *Proc. 2nd ICSE Workshop Open Source Software Eng.* (ICSE 02), ACM Press, 2002, pp. 35-37.
18. J.R. Erenkrantz and R.N. Taylor, "Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community," *Proc. 1st Workshop Open Source in an Industrial Context* (OSIC 03), ACM Press, 2003; <http://flosshub.org/system/files/erenkrantz2003.pdf>.
19. J. Wesselius, "The Bazaar inside the Cathedral: Business Models for Internal Markets," *IEEE Software*, May/June 2008, pp. 60-66.
20. V.K. Gurbani, G. Anita, and J.D. Herbsleb, "A Case Study of a Corporate Open Source Development Model," *Proc. 28th Int'l Conf. Software Eng.* (ICSE 06), ACM Press, 2006, pp. 472-481.
21. J. Dinkelacker et al., "Progressive Open Source," *Proc. 24th Int'l Conf. Software Eng.* (ICSE 02), ACM Press, 2002, pp. 177-184.
22. K.-J. Stol et al., "A Comparative Study of Challenges in Integrating Open Source Software and Inner Source Software," to appear in *Information and Software Technology*, 2011, doi:10.1016/j.infsof.2011.06.007.
23. T. O'Reilly, "Lessons from Open Source Software Development," *Comm. ACM*, Apr. 1999, pp. 33-37.
24. K. Martin and B. Hoffman, "An Open Source Approach to Developing Software in a Small Organization," *IEEE Software*, Jan./Feb. 2007, pp. 46-53.

COVER FEATURE

25. W. Scacchi et al., "Understanding Free/Open Source Software Development Processes," *Software Process: Improvement and Practice*, Mar./Apr. 2006, pp. 95-105.
26. C. Melian, "Progressive Open Source: The Construction of a Development Project at Hewlett-Packard," PhD dissertation, Stockholm School of Economics, 2007.
27. K.D. Levin and O. Yadid, "Optimal Release Time of Improved Versions of Software Packages," *Information and Software Technology*, Jan./Feb. 1990, pp. 65-70.
28. M.S. Krishnan, "Software Release Management: A Business Perspective," *Proc. 1994 Conf. Centre for Advanced Studies on Collaborative Research (CASCON 94)*, IBM Press, 1994, pp. 36-43.
29. B. Fitzgerald and M. Michlmayr, *Time-Based Release Management in Free/Open Source (FOSS) Projects*, tech. report TR-2011-04, Lero—the Irish Software Eng. Research Centre, Univ. of Limerick, 2011; <http://lero.ie/sites/default/files/Lero-TR-2011-04.pdf>.
30. B. Fitzgerald, "The Transformation of Open Source Software," *MIS Q.*, Sept. 2006, pp. 587-598.

Brian Fitzgerald is a principal investigator at Lero—the Irish Software Engineering Research Centre and founding director of the Lero Graduate School in Software Engineering at the University of Limerick, Ireland, where he also holds the Frederick A. Krehbiel Chair II in Innovation in Global Business and Technology and is vice president of research. His research focuses on software development, encompassing development methods, global software development, agile methods, and open source software. Fitzgerald received a PhD in computer science from the University of London. He is a fellow of the Irish Computer Society and the British Computer Society. Contact him at bf@ul.ie.



Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.



CYBERSECURITY

Enroll now.

ON THIS BATTLEFIELD,
EDUCATION IS YOUR BEST DEFENSE.

Cyber attacks are being waged all over the world, creating an unprecedented demand for trained professionals to protect our country's data assets and develop cybersecurity policies. Help meet the demand with a bachelor's or master's degree in cybersecurity. Whether you plan to work for Cyber Command taking down cyber terrorists or for private industry battling hackers, UMUC can help you make it possible.

- Designated as a National Center of Academic Excellence in Information Assurance Education by the NSA and DHS
- BS and MS in cybersecurity and MS in cybersecurity policy available
- Programs offered entirely online
- Interest-free monthly payment plan available, plus financial aid for those who qualify

800-888-UMUC • umuc.edu/cyberwarrior



Copyright © 2011 University of Maryland University College

COVER FEATURE



Software Engineering Meets Evolutionary Computation

Mark Harman, *University College London*

The concept of evolutionary computation has affected virtually every area of software design, not merely as a metaphor, but as a realistic algorithm for exploration, insight, and improvement.

Software evolves. This fact was recognized early in the history of software engineering.¹ Although the term “software evolution” has come to refer to the process by which successful software installations continually adapt to cater to the changing requirements and environments in which they operate, this is a figurative allusion to Darwinian evolution, not a specifically technical term.

Independently, an entire computer science community has developed that uses the term *evolutionary computation* with a specifically technical meaning: the study of algorithms that incorporate aspects of fitness-guided selection to search a space of candidate solutions for those well-adapted to solving a specific problem. This community has its own conferences and journals that constitute a considerable body of knowledge concerning the best way to develop and apply evolution as a driver for innovation and adaption in an automated metaheuristic optimization process.

Computer scientists have used evolutionary computation to optimize the design of artifacts and processes from an astonishingly wide variety of general engineering dis-

ciplines. However, perhaps surprisingly, until the past 10 years, comparatively little work delved into the application of evolutionary computation (and other related search-based optimization) techniques to software engineering. This was the motivation for the foundation of the field now known as *search-based software engineering*, which focuses on the application of search-based optimization techniques to problems in software engineering.

In the past decade, researchers have applied SBSE to a wide range of software engineering topics, including requirements,^{2,3} estimation and prediction,⁴ design,⁵ testing,⁶⁻⁹ and refactoring.^{10,11} Numerous search-based optimization techniques have been used, with a recent comprehensive survey reporting 15 different techniques.¹²

There is no reason why SBSE must be concerned solely with evolutionary computation; other optimization algorithms can and have been used. For example, in the 830 papers in the SBSE repository as of June 2011, 587 use one or more optimization techniques ([http://crestweb.cs.ucl.ac.uk/resources/sbse repository](http://crestweb.cs.ucl.ac.uk/resources/sbse_repository)). The percentages of papers using each technique are as follows: evolutionary algorithms (no specific style mentioned), 9.0 percent; genetic algorithms, 45.5 percent; genetic programming, 13.5 percent; evolution strategies, 0.6 percent; particle swarm optimization, 1.8 percent; estimation of distribution algorithms, 1.4 percent; and scatter search, 0.8 percent. However, evolutionary computation has been used in 71 percent of all papers on SBSE, and it is the only optimization technique to have been applied to every software engineering application area.¹²

COVER FEATURE

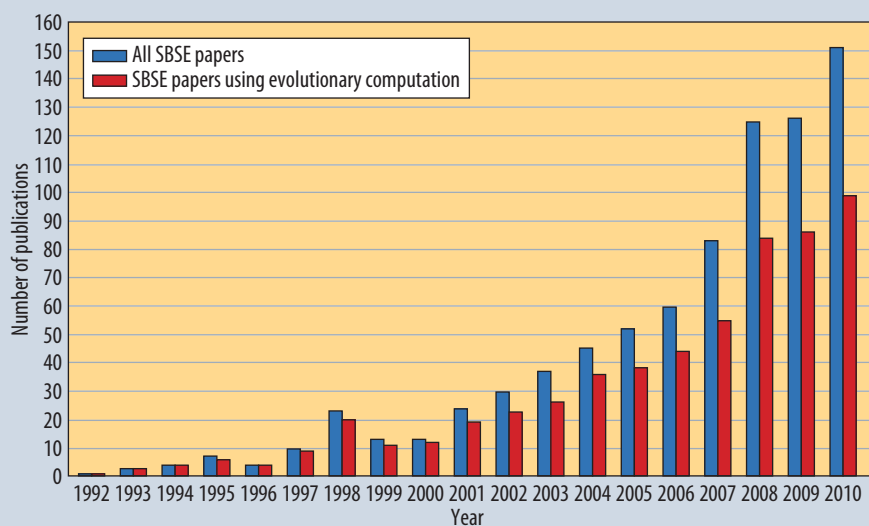


Figure 1. Increase in papers on SBSE and concomitant growth in papers using evolutionary computation for software engineering.

THE GROWTH OF A FIELD

This interest in SBSE in general and evolutionary computation for software engineering in particular has increased rapidly in the past 10 years. Figure 1 shows the growth in publications in SBSE and the concomitant increase in papers within the SBSE field that use evolutionary computation.

SBSE is not only an academic research area—it increasingly provides a set of methods, tools, and techniques that are finding widespread industrial application. The first (and still the most widely) studied area of research targets the application of SBSE to automated test data generation.¹² Known as *search-based software testing*, this widely surveyed area has its own coherent body of literature, and SBST serves as the topic area for a dedicated annual workshop.⁶⁻⁹

One of the earliest industrial examples of the application of SBST in industrial practice was at Daimler Chrysler, where Joachim Wegener and his research team implemented a system for evolutionary testing.¹³ This system used a genetic algorithm to search for branch-adequate test data, returning a set of test data and associated coverage metrics to the developer. Daimler also experimented with search-based techniques for the functional testing of a parking system¹⁴ and the temporal testing of air bag controllers.¹⁵

More recently, Microsoft incorporated search-based techniques for incorporating floating-point computation into its PeX software testing tool,^{16,17} while Google incorporated multiobjective regression test optimization into its test process.¹⁸ NASA,¹⁹ Motorola,²⁰ and Ericsson²¹ have experimented with SBSE for requirements analysis and optimization, while Ericsson has also used genetic

programming (GP) to predict fault slip-through on two large projects.²²

EVOLUTIONARY COMPUTATION

As with so much of significance in computer science, Alan Turing²³ was the first to introduce the idea that Darwin's theory of evolution might find a sympathetic counterpart in computation, although this very initial formulation was arguably more Lamarckian than Darwinian. John Holland's subsequent experiments with evolutionary computation played an important role in popularizing the field.²⁴

It is hard to overstate the impact of evolutionary computation on computational thinking. To give some quantifiable indication of this impact, on 22 June 2011, Holland's book had attracted 24,597 citations (according to Google Scholar), while David E. Goldberg's more recent account had 38,706 citations.²⁵ These citations come from varied fields of engineering and design, not just from software engineers and computer scientists. There is hardly any aspect of design that has not been profoundly affected by the concept of evolutionary optimization, not merely as a metaphor, but as a realistic algorithm for exploration, insight, and improvement.

With such exceptional and far-reaching impact, it is not surprising that software engineering would also fall under the evolutionary spell. The only surprising aspect of this history is that it took so long. With the benefit of hindsight, it is extraordinary that what is essentially a *software* technique could have failed to find application to *software* engineering for so many years. Perhaps this is simply a reflection of the time it took for the research community to recognize that activities associated with software development are, indeed, essentially *engineering* activities and that, consequently, *optimization* was a natural approach.

Fortunately, despite its slowness to take up evolutionary computation, the software engineering community has recently made up for this sluggishness. Recently, it has even been argued that software's "virtual" nature renders it the most suitable of all engineering materials for optimization techniques.²⁶

The formulation of the generic genetic algorithm shown in Figure 2 is taken from a 2003 survey of SBSE.²⁷ Although there might be variations on the theme in the extensive literature on the topic, the central principles of fitness com-

putation and selection can be found in most formulations of evolutionary computation.

The most widely studied variations on this theme are evolution strategies²⁸ (which have been used in test data generation²⁹) and GP,³⁰ in which the artifact to be optimized is not a list but a tree—the abstract syntax tree of some programming notation. Of these, GP has been more widely used in SBSE.

The first authors to suggest evolutionary algorithms for software engineering were S. Xanthakis and colleagues,³¹ who advocated the use of genetic algorithms for software test data generation. Soon after, Carl Chang and colleagues developed SPMNet, a tool for software project planning based on evolutionary algorithms.³² In an editorial that presaged the advent of SBSE as an integral field of study, Chang argued for the more widespread application of evolutionary computation in software engineering.³³

The term SBSE was coined in 2001 to capture the general application of search-based optimization techniques, including evolutionary computation, to software engineering problems.³⁴ This paper first articulated a vision for a research field of SBSE, arguing that search-based optimization offered a potentially generic, unifying, and potent approach to the spectrum of software engineering activities and products. A recent 10-year retrospective study confirms the growing importance of SBSE research activity.³⁵

THE ROLE OF TESTING

Of all the areas of software engineering activity to which researchers have applied SBSE techniques, software testing is both the first area to be tackled and the area that has received the most widespread study.

The general idea behind all approaches to search-based test data generation is that the set of test cases forms a search space and that the test adequacy criterion is coded as a fitness function. For example, to achieve branch coverage, the fitness function assesses how close a test input comes to executing an uncovered branch. In contrast, to find worst case execution time, fitness is simply the duration of execution for the test case in question.

Researchers have used search to attack a variety of testing goals, including structural testing,^{36,37} functional testing,¹⁴ safety testing,³⁸ security testing,³⁹ robustness testing,⁴⁰ stress testing,⁴¹ integration testing,⁴² Web application testing,⁴³ and quality-of-service testing.⁴⁴ Most of this work has concerned the problem of generating inputs that provide a test suite that meets a test adequacy criterion. Although the problem of test input generation is often called automated test data generation (ATDG), strictly speaking, without an oracle, only the input is generated. Figure 3 illustrates the ge-

```

Set generation number,  $m := 0$ 
Choose the initial population,  $P(0)$ 
Evaluate fitness  $P(0)$ ,  $F(P_i(0))$ 
loop
Recombine:  $P(m) := R(P(m))$ 
Mutate:  $P(m) := M(P(m))$ 
Evaluate:  $F(P(m))$ 
Select:  $P(m+1) := S(P(m))$ 
 $m := m + 1$ 
exit when goal or stopping condition is satisfied
end loop;

```

Figure 2. A generic genetic algorithm.

neric form of the most common approach in the literature, in which test inputs are generated according to a test adequacy criterion. The test adequacy criterion, the human input to the process, determines the testing goal.

The adequacy criterion can be almost any form of testing goal that can be defined and assessed numerically. For example, it can be structural (covering branches, paths, statements), functional (covering scenarios), temporal (finding worst/best case execution times), and so on. SBSE's generic nature is a considerable advantage and one reason why many researchers have been able to adapt it to different testing problems.

A human-defined fitness function must capture the adequacy criteria. Once the fitness function for a test adequacy criterion, C , has been defined, the generation of C -adequate test inputs can be automated using SBSE. The SBSE tools that implement different forms of testing all follow the broad structure outlined in Figure 3. They code the adequacy as a fitness, using it to assess the fitness of

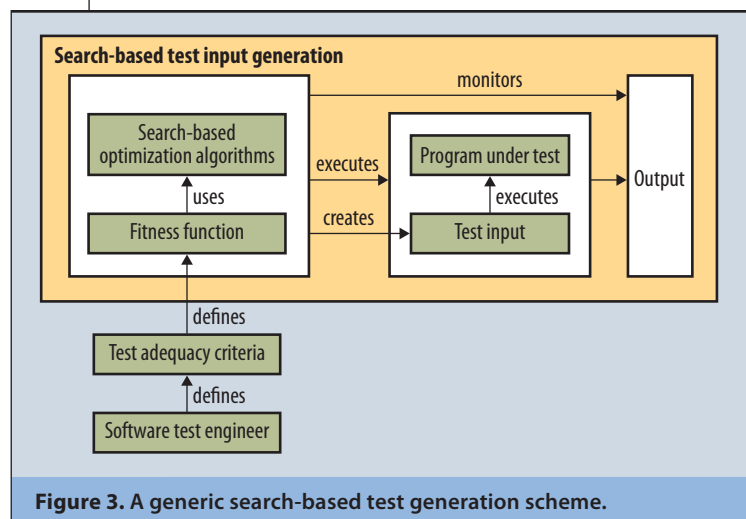



Figure 3. A generic search-based test generation scheme.

COVER FEATURE

candidate test inputs. To assess fitness, the ATDG system must cause the program to be executed for the candidate inputs. The ATDG system then monitors the execution to assess fitness: how well does the input meet the test adequacy criterion?

GENETIC PROGRAMMING

Since program notation is one of the engineering materials with which software engineering is principally concerned, we might expect that GP would have found widespread application in software engineering. Indeed, there has been much work in GP for software engineering, but not always to create the program code for the software system itself. Program construction remains a largely and stubbornly human-centric activity. One goal of the SBSE



There has been much work in genetic programming for software engineering, but not always to create the program code for the software system itself.

research agenda is the creation of optimization techniques that can maximally automate the program construction.

This remains an open grand challenge for SBSE, as it does for several other related research communities. Although full achievement of this grand challenge remains some way off, there are many highly effective ways to use GP in SBSE.

Predictive modeling

It is not necessary for GP to evolve a complete program for it to be useful; it merely needs to generate a set of rules that capture some important property of a problem. GP is very good at this. It was first used in software engineering to capture the equations that define good software cost-estimation models.⁴⁵ This is a natural fit because these models are often highly nonlinear and exhibit piecewise behavior. The equations that define them are nevertheless relatively small and there is a natural fitness function: the degree of “fit” of the equation to the observed data for a set of projects. Consequently, the space of all possible software cost models forms a natural target for GP work.

Mutation testing

Recent work has shown how GP can be used to search for subtle software faults.⁴⁶ A strongly typed GP system uses a restricted C grammar to generate complex faults. The system computes fitness for two objectives: one syntactic and the other semantic.

The syntactic fitness function measures the syntactic similarity of the original program and the faulty version

to minimize the difference between the two. The semantic fitness uses a set of test cases to measure the degree of semantic similarity between the two programs, seeking a faulty version that is almost (but not quite) identical to the original program. In this manner, the approach seeks faults that are hard to detect either syntactically or semantically.

Automated bug fixing

Most work on SBSE applied to testing has been concerned with the discovery of faults in software. However, more recently, authors have also turned their attention to the use of SBSE to patch software and fix bugs.^{47,48}

Wesley Weimer and colleagues⁴⁵ used GP to evolve patches that fix real bugs, winning a Gold Medal at the 2009 Genetic and Evolution Computation Conference for the achievement of human competitive results. The patching process uses what might be called “plastic surgery” to scavenge fragments of code for fault fixing from elsewhere within the program under test. The patches are evolved using GP, for which fitness is computed in terms of passing and failing test cases, thereby fixing the fault and avoiding regression.

OPEN PROBLEMS AND REMAINING CHALLENGES

In previous work on evolutionary computation for software engineering, the evolution model was an extremely simple one. Whether the technique used was a genetic algorithm or a variant such as genetic programming, there was but a single population, evolving according to a single fitness function. Real-world evolution is far more complex because multiple populations interact with one another, effectively solving multiple objective problems with several different fitness functions. Furthermore, in the real world of coevolution, the fitness of one population can critically impact the fitness of another.

Coevolutionary computation

In coevolutionary computation, two or more populations evolve simultaneously, with the fitness of each depending upon the current population of the other. Coevolution takes two forms: *cooperative* and *competitive*.

In cooperative coevolution, the two populations are sympathetic to each other, and the overall system seeks a symbiotic balance between the two. Biology offers many examples of this, such as the symbiotic relationship between pollinating insects and the plants they pollinate. By contrast, in competitive coevolution, one population plays the role of predator and the other is its prey. This coevolution model’s goal is to create an “arms race” between the two populations so that each continually improves its fitness with respect to the other.

Researchers have applied both cooperative and competitive coevolution to software engineering problems,

with the competitive form the first to be studied. Konstantinos Adamopoulos and coauthors⁴⁹ used it to evolve sets of mutants and sets of test cases, where the test cases act as predators and the mutants as their prey.

Andrea Arcuri and Xin Ya⁴⁷ also developed a competitive coevolutionary model of bug fixing, in which one population essentially seeks out patches that can pass test cases, while an oracle produces cases to test the current population of potential patches. In this way, the patch is the prey, while the test cases act as predators.

In the work of both Adamopoulos and colleagues and Arcuri and Yao, the two populations consist of code fragments and test cases. The primary difference is that for Adamopoulos and colleagues, the goal is to find models of faults, whereas in the work of Arcuri and Yao, the goal is to find patches that fix faults.

Any situation in which code and test cases coevolve offers a natural candidate for a competitive, predator-prey model of coevolution. We can expect much more work on this model because it is well-suited to the way software testing takes place. Indeed, many software developers will already be familiar with the feeling that they are the prey for some demanding “testing predator.” In software security, too, a similar predator-prey culture is prevalent, and we can expect competitive coevolution to make this more than mere analogy.

Many other aspects of software engineering problems lend themselves to a coevolutionary optimization model because software systems are complex and rich in potential populations that could be productively coevolved using both competitive and cooperative coevolution. While software testing and security problems naturally fit a competitive instantiation, many other software engineering problems are better suited to a cooperative model. For example, components, agents, stakeholder behavior models, designs, cognitive models, requirements, test cases, use cases, and management plans are all important aspects of software systems for which optimization is an important concern.

If they can find a suitable fitness function, researchers could use SBSE to constructively and sympathetically coevolve collaborating subsolutions. For example, in recent work on cooperative coevolution in SBSE, Jian Ren and coauthors⁵⁰ explored the way in which different aspects of a software project could be cooperatively coevolved.


Although work on coevolution in software engineering is in its infancy, it seems likely that this area will be the subject of much interest over the coming years because of the close fit to complex software engineering problems.

Scalability

Scalability is probably the biggest generic challenge facing software engineering. Although processing power is increasing, the scale and complexity of software systems is increasing at least as fast. Techniques for software de-

velopment, testing, and verification are required that can scale to meet this challenge.

Fortunately, SBSE techniques such as evolutionary computation are highly scalable because they can be easily parallelized. Genetic algorithms involve a population to which the same fitness function is applied repeatedly. There is no reason why the entire population's fitness cannot be evaluated on an SIMD architecture in a single step. Several authors have demonstrated that this parallelism can be exploited in SBSE work to obtain scalability through distributed computation.⁵¹⁻⁵³



If they can find a suitable fitness function, researchers could use SBSE to constructively and sympathetically coevolve collaborating subsolutions.

Recent work has shown how this parallelism can be exploited on general-purpose graphical processing units (GPGPUs). Shin Yoo and coauthors⁵⁴ showed how to map the problem of regression testing onto a GPGPU, solving multiobjective regression test selection problems up to 20 times faster than conventional architectures. The results also outperformed multicore CPU computation and made possible large-scale, real-world smoke-testing optimization that would have been impossible without the scalability offered by GPGPUs.

This work opens up the prospect of an inexpensive and effective means of unleashing the parallelism latent in evolutionary computation and harnessing its scalability for software engineering applications. The importance of scalability and the rapid industry-wide migration to multicore computing indicate that parallel SBSE is a topic likely to receive considerable attention in the coming years.

Interactive evolution

Although researchers have successfully applied SBSE to the automation of many areas of software engineering, some aspects of this discipline cannot be automated. The designs we create are ultimately intended for human consumption, and human judgment will always play a critical role in almost every design process. It is difficult to capture some design aspects in a fitness function, and software engineering design challenges are no exception.

Interactive evolution offers an approach that places the human in the fitness computation loop.⁵⁵ SBSE researchers have used interactive evolution to allow humans to play a role in defining a software engineering design's fitness. For example, Christopher L. Simons and coauthors used interactive evolution to search for good designs for an object-oriented class diagram.⁵⁶


COVER FEATURE

This work is intended to support early life-cycle design activity in which human judgment is critical. Since software engineering design involves many aspects of human judgment and remains a labor-intensive engineering activity, we can expect much future interest in interactive forms of evolutionary computation for software engineering.

Characterization of landscapes and problems

As in any field, SBSE generated initial excitement that led to widespread uptake and development of a broad range of applications. Researchers attacked many different aspects of software engineering using evolutionary computation, with early successes stimulating further work and wider development. However, after the initial “gold rush,” there comes a time for consolidation.

In fact, this closely mirrors the behavior of the evolutionary compilation process itself. That is, the early stages of the evolutionary process tend to favor exploration of



Seeking optimal and near-optimal solutions is natural because it lies at the core of all engineering approaches and pervades all engineering activity.

new parts of the search space, while the later phases tend to favor the exploitation of productive areas, seeking improved fitness within them.

There is evidence for the emergence of this second phase of activity in the software engineering research community. Although work continues on developing exciting new areas for applying SBSE for the first time, a consistent effort is emerging that seeks to develop a deeper understanding and scientific basis for the results obtained thus far. Both the theoretical study of algorithm performance and the theoretical and empirical analysis of problems is now common in the well-explored areas of SBST.^{56,57,57} But as the field matures, we can expect that there will be an increasing proportion of activity in the area of theoretical characterization of problems and algorithms that can be used to resolve them.

Hyperheuristic optimization

Seeking optimal and near-optimal solutions is natural because it lies at the core of all engineering approaches and pervades all engineering activity. We have seen from SBSE's widespread application that its generic and highly applicable nature has led to many tailor-made optimization algorithms for problem instances across the spectrum of software engineering activities.

These algorithms have proved to be astonishingly effective, and researchers and practitioners alike are using them.

However, custom-made development will always remain a slow and expensive process compared to the potential offered by hyperheuristics—algorithms devised by combining simpler heuristics to efficiently solve computational search problems—making *hyperheuristic software engineering* a natural next step for the research community.⁵⁸

A primary goal of SBST work is test input generation. However, this work does not address the problem of the generation of test cases, which are, in their most abstract form, input-output pairs. SBST tends to generate inputs that achieve various test adequacy criteria but not the expected outputs; checking the test cases therefore typically remains an unautomated aspect of the overall test activity. Determining the correct output for a given input is the “oracle problem” in software testing. While there has been work on using SBST to reduce the cost of human oracle checking,⁵⁹ little work has combined SBST with oracle construction. One promising recent approach is the work by Gordon Fraser and Andreas Zeller,⁶⁰ which uses SBST to generate test data to kill mutants and simultaneously seeks to construct a partial oracle.

Applications in emerging areas

Much of the SBSE literature focuses on what might be termed “traditional” software engineering application areas, such as requirements, project planning, design, implementation, testing, and maintenance. However, there is no reason why SBSE cannot be applied to other forms of software development. Any software engineering problem in which there are objectives to be optimized offers opportunities for search-based optimization.

For example, because researchers could optimize many competing objectives for agile computing paradigms, the application of evolutionary computation to these paradigms cannot be far off. In addition, initial work has been reported formulating the challenges related to the emerging cloud-based paradigm.⁶¹ It is also likely that the increasing use of mobile and embedded systems will lead to increased nonfunctional properties, for which SBSE has already been applied.⁶ Even in the comparatively “blue sky” field of quantum computation, SBSE has proved to be a potential source of both solutions to potential challenges⁶² as well as a beneficiary of developments in quantum computation.⁶³

WHY IS EVOLUTIONARY COMPUTATION SO POPULAR?

Evolutionary algorithms have proved to be the most widely applied of all SBSE techniques.¹² However, this raises the question of why evolutionary search algorithms should have proved so popular. Is there a particular technical reason for this prevalence?

It is true that these algorithms are widely applicable because they can cater to single and multiple objectives,

they can incorporate human fitness evaluation, and they can be easily parallelized. These important technical considerations provide compelling motivations for the consideration of evolutionary computation as a natural technique for optimizing software engineering. However, evolutionary algorithms also offer many interesting variations and numerous parameters to be tuned and explored. These can prove to be as enticing to researchers as they are frustrating to practitioners.

We must be wary of the unquestioning adoption of evolutionary algorithms merely because they are popular and widely applicable or because, historically, other researchers have adopted them for SBSE problems; none of these are scientific motivations for adoption. Part of the importance of the second phase of research activity concentrating on characterization and algorithmic complexity analysis lies in the way in which it can help determine the best algorithm for a particular software engineering problem. This will not always be an evolutionary algorithm. For example, there is evidence to suggest that, for branch-adequate test data generation, local search algorithms may be better suited to fast and effective coverage.³⁷

PREVIOUS RESEARCH

Since the term SBSE was coined in 2001, there has been an explosion of interest and activity in this area, creating a considerable body of literature.

Surveys covering requirements,³ predictive modeling,⁴ design,⁵ and testing⁶⁻⁹ provide an excellent analysis of the application of SBSE to these topic-specific areas. Other surveys of specific software testing topics include sections devoted to search-based techniques, such as recent reports covering mutation testing⁶⁴ and regression test optimization.⁶⁵

Some reports focus on specific aspects of SBSE, for example, the use of metrics as fitness functions,⁶⁶ structured reviews of the empirical evidence concerning test data generation,⁷ nonfunctional testing,⁶ and predictive modeling.⁴ Other reports identify open problems and future SBSE research agendas for program comprehension,⁶⁷ predictive modeling,⁶⁸ and testability transformation.⁶⁹

The systematic reviews provide a convenient summary of the current empirical evidence base for SBSE, while the agenda-defining papers may provide potential topics and ideas for future research projects and PhD theses. Students interested in finding a more introductory tutorial might consider a forthcoming summary that provides a self-contained introduction to SBSE and offers practical guidance and advice.⁷⁰ Finally, some surveys seek to cover the entire area of SBSE. A 2003 survey that covered project planning, design, maintenance, and testing,²⁷ and the overview and introductory paper from ICSE 2007,⁷¹ are both widely cited sources of general information about SBSE. A comprehensive 2009 survey mapped the entire

field, providing the first data on emerging SBSE trends,¹² and a 2011 retrospective provides a bibliometric analysis of the SBSE literature.³⁵

Researchers can use evolutionary computation algorithms to optimize any artifact of design for which a suitable fitness function can be defined. Software engineering provides a rich and varied source of such artifacts together with metrics that can be readily adapted to their new role as fitness functions. This has led to 10 years of rapid developments in search-based software engineering. Evolutionary computation for software engineering forms a substantial part of this overall growth in SBSE research and practice. There are also many exciting emerging developments in evolutionary computation for software engineering for which we can expect a great deal of future work. **□**

Acknowledgments

As of July 2011, SBSE research had rapidly grown to include publications by more than 800 authors from 270 institutions in more than 40 countries. Although I cannot acknowledge each person here by name, I am grateful to this community for the many stimulating conversations about SBSE over the past 10 years. In particular, I thank Bill Langdon, Richard Torkar, Wes Weimer, and Xin Yao for their comments on an earlier draft of this article. I also thank Yuanyuan Zhang for many discussions on SBSE and her assistance in the production of Figures 1 and 3 as well as her tireless work on the SBSE repository, which provides an excellent resource for this rapidly growing community.

References

1. M.M. Lehman, "On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle," *J. Systems and Software*, vol. 1, no. 3, 1980, pp. 213-221.
2. A.J. Bagnall, V.J. Rayward-Smith, and I.M. Whittle, "The Next Release Problem," *Information and Software Technology*, Dec. 2001, pp. 883-890.
3. Y. Zhang, A. Finkelstein, and M. Harman, "Search-Based Requirements Optimisation: Existing Work and Challenges," *Proc. Int'l Working Conf. Requirements Eng.: Foundation for Software Quality (REFSQ 08)*, LNCS 5025, Springer, 2008, pp. 88-94.
4. W. Afzal and R. Torkar, "On the Application of Genetic Programming for Software Engineering Predictive Modeling: A Systematic Review," *Expert Systems Applications*, vol. 38, no. 9, 2011, pp. 11984-11997.
5. O. Rähkä, "A Survey on Search-Based Software Design," *Computer Science Rev.*, vol. 4, no. 4, 2010, pp. 203-249.
6. W. Afzal, R. Torkar, and R. Feldt, "A Systematic Review of Search-Based Testing for Non-Functional System Properties," *Information and Software Technology*, vol. 51, no. 6, 2009, pp. 957-976.
7. S. Ali et al., "A Systematic Review of the Application and Empirical Investigation of Search-Based Test-Case Generation," *IEEE Trans. Software Eng.*, vol. 36, no. 6, 2010, pp. 742-762.

COVER FEATURE

8. M. Harman, "Automated Test Data Generation Using Search-Based Software Engineering," *Proc. 2nd Int'l Workshop Automation of Software Test (AST 07)*, IEEE CS Press, 2007, p. 2.
9. P. McMinn, "Search-Based Software Test Data Generation: A Survey," *Software Testing, Verification and Reliability*, vol. 14, no. 2, 2004, pp. 105-156.
10. M. O'Keefe and M. Ó Cinnéide, "Search-Based Software Maintenance," *Proc. Conf. Software Maintenance and Re-engineering (CSMR 06)*, IEEE CS Press, 2006, pp. 249-260.
11. M. Harman and L. Tratt, "Pareto Optimal Search-Based Refactoring at the Design Level," *Proc. 9th Ann. Conf. Genetic and Evolutionary Computation (GECCO 07)*, ACM Press, 2007, pp. 1106-1113.
12. M. Harman, A. Mansouri, and Y. Zhang, *Search-Based Software Engineering: A Comprehensive Analysis and Review of Trends, Techniques, and Applications*, tech. report TR-09-03, Dept. of Computer Science, King's College London, 2009.
13. J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary Test Environment for Automatic Structural Testing," *Information and Software Technology*, vol. 43, no. 14, 2001, pp. 841-854.
14. J. Wegener and O. Bühler, "Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System," *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2004)*, LNCS 3103, Springer, 2004, pp. 1400-1412.
15. J. Wegener and F. Mueller, "A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints," *Real-Time Systems*, vol. 21, no. 3, 2001, pp. 241-268.
16. C. Cadar et al., "Symbolic Execution for Software Testing in Practice: Preliminary Assessment," *Proc. 33rd Int'l Conf. Software Eng. (ICSE11)*, ACM Press, 2011, pp. 1066-1071.
17. K. Lakhotia et al., "FloPSy—Search-Based Floating Point Constraint Solving for Symbolic Execution," *Proc. 22nd IFIP Int'l Conf. Testing Software and Systems (ICTSS 10)*, LNCS 6435, Springer, 2010, pp. 142-157.
18. S. Yoo, R. Nilsson, and M. Harman, "Faster Fault Finding at Google Using Multiobjective Regression Test Optimisation," *Proc. 8th European Software Eng. Conf. and ACM SIGSOFT Symp. Foundations of Software Eng. (ESEC/FSE 11)*, ACM Press, Sept. 2011; <http://2011.esec-fse.org/industrial-track>.
19. S.L. Cornford et al., "Optimizing Spacecraft Design—Optimization Engine Development: Progress and Plans," *Proc. IEEE Aerospace Conf. (AeroConf 03)*, IEEE Press, 2003, pp. 3681-3690.
20. P. Baker et al., "Search-Based Approaches to Component Selection and Prioritization for the Next Release Problem," *Proc. 22nd Int'l Conf. Software Maintenance (ICSM 06)*, IEEE Press, 2006, pp. 176-185.
21. Y. Zhang et al., "Today/Future Importance Analysis," *Proc. ACM Genetic and Evolutionary Computation Conf. (GECCO 10)*, ACM Press, 2010, pp. 1357-1364.
22. W. Afzal et al., "Search-Based Prediction of Fault-Slip-Through in Large Software Projects," *Proc. 2nd Int'l Symp. Search-Based Software Eng. (SSBSE 10)*, IEEE CS Press, 2010, pp. 79-88.
23. A.M. Turing, "Computing Machinery and Intelligence," *Mind*, Jan. 1950, pp. 433-460.
24. J.H. Holland, *Adaption in Natural and Artificial Systems*, MIT Press, 1975.
25. D.E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989.
26. M. Harman, "Why the Virtual Nature of Software Makes It Ideal for Search-Based Optimization," *Proc. 13th Int'l Conf. Fundamental Approaches to Software Eng. (FASE 10)*, IEEE CS Press, 2010, pp. 1-12.
27. J. Clark et al., "Reformulating Software Engineering as a Search Problem," *IEE Proceedings—Software*, vol. 150, no. 3, 2003, pp. 161-175.
28. H.-P. Schwefel and T. Bäck, "Artificial Evolution: How and Why?" *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, D. Quagliarella et al., eds., John Wiley & Sons, 1998, pp. 1-19.
29. E. Alba and F. Chicano, "Observations in Using Parallel and Sequential Evolutionary Algorithms for Automatic Software Testing," *Computers & Operations Research*, Oct. 2008, pp. 3161-3183.
30. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
31. S. Xanthakis et al., "Application of Genetic Algorithms to Software Testing," *Proc. 5th Int'l Conf Software Eng. (ICSE 92)*, IEEE CS Press, 1992, pp. 625-636.
32. C.K. Chang et al., "SPMNet: A Formal Methodology for Software Management," *Proc. 18th Ann. Int'l Computer Software and Applications Conf. (COMPSAC 94)*, IEEE CS Press, 1994, p. 57.
33. C.K. Chang, "Changing Face of Software Engineering," *IEEE Software*, vol. 11, no. 1, 1994, pp. 4-5.
34. M. Harman and B.F. Jones, "Search-Based Software Engineering," *Information and Software Technology*, Dec. 2001, pp. 833-839.
35. F.G. Freitas and J.T. Souza, "Ten Years of Search-Based Software Engineering: A Bibliometric Analysis," *Proc. 3rd Int'l Symp. Search-Based Software Eng. (SSBSE 11)*, Sept. 2011; www.springerlink.com/content/q2tr7835534pj4444.
36. A. Arcuri, "It Does Matter How You Normalise the Branch Distance in Search-Based Software Testing," *Proc. Int'l Conf. Software Testing (ICST 10)*, IEEE CS Press, 2010, pp. 205-214.
37. M. Harman and P. McMinn, "A Theoretical and Empirical Study of Search-Based Testing: Local, Global and Hybrid Search," *IEEE Trans. Software Eng.*, vol. 36, no. 2, 2010, pp. 226-247.
38. A. Baresel, H. Sthamer, and J. Wegener, "Applying Evolutionary Testing to Search for Critical Defects," *Proc. Conf. Genetic and Evolutionary Computation (GECCO 04)*, LNCS 3103, Springer, 2004, pp. 1427-1428.
39. C. Del Grosso et al., "Improving Network Applications Security: A New Heuristic to Generate Stress Testing Data," *Proc. Conf. Genetic and Evolutionary Computation (GECCO 05)*, ACM Press, 2005, pp. 1037-1043.
40. A.C. Schultz, J.J. Grefenstette, and K.A. De Jong, "Test and Evaluation by Genetic Algorithms," *IEEE Expert (also IEEE Intelligent Systems and Their Applications)*, vol. 8, no. 5, 1993, pp. 9-14.
41. L.C. Briand, Y. Labiche, and M. Shousha, "Stress Testing Real-Time Systems with Genetic Algorithms," *Proc. Conf. Genetic and Evolutionary Computation (GECCO 05)*, ACM Press, 2005, pp. 1021-1028.

42. L.C. Briand, J. Feng, and Y. Labiche, "Using Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders," *Proc. 14th Int'l Conf. Software Eng. and Knowledge Eng. (SEKE 02)*, ACM Press, 2002, pp. 43-50.
43. N. Alshahwan and M. Harman, "Automated Web Application Testing Using Search-Based Software Engineering," *Proc. IEEE/ACM Int'l Conf. Automated Software Eng. (ASE 11)*, Nov. 2011, to appear.
44. M. Di Penta et al., "Search-Based Testing of Service Level Agreements," *Proc. 9th Ann. Conf. Genetic and Evolutionary Computation (GECCO 07)*, ACM Press, 2007, pp. 1090-1097.
45. J.J. Dolado and L. Fernandez, "Genetic Programming, Neural Networks and Linear Regression in Software Project Estimation," *Proc. Int'l Conf. Software Process Improvement, Research, Education and Training (INSPIRE III)*, British Computer Soc., 1998, pp. 157-171.
46. W.B. Langdon, M. Harman, and Y. Jia, "Efficient Multi-objective Higher Order Mutation Testing with Genetic Programming," *J. Systems and Software*, vol. 83, no. 12, 2011, pp. 2416-2430.
47. A. Arcuri and X. Yao, "A Novel Co-evolutionary Approach to Automatic Software Bug Fixing," *Proc. IEEE Congress on Evolutionary Computation (CEC 08)*, IEEE CS Press, 2008, pp. 162-168.
48. W. Weimer et al., "Automatically Finding Patches Using Genetic Programming," *Proc. Int'l Conf. Software Eng. (ICSE 09)*, IEEE CS Press, 2009, pp. 364-374.
49. K. Adamopoulos, M. Harman, and R.M. Hierons, "How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-Evolution," *Proc. Conf. Genetic and Evolutionary Computation (GECCO 04)*, LNCS 3103, Springer, 2004, pp. 1338-1349.
50. J. Ren, M. Harman, and M. Di Penta, "Cooperative Co-evolutionary Optimization on Software Project Staff Assignments and Job Scheduling," *Proc. 3rd Int'l Symp. Search-Based Software Eng. (SSBSE 11)*, Sept. 2011; www.springerlink.com/content/a611526179255p80.
51. F. Asadi, G. Antoniol, and Y.-G. Guéhéneuc, "Concept Location with Genetic Algorithms: A Comparison of Four Distributed Architectures," *Proc. 2nd Int'l Symp. Search-Based Software Eng. (SSBSE 10)*, IEEE CS Press, 2010, pp. 153-162.
52. K. Mahdavi, M. Harman, and R.M. Hierons, "A Multiple Hill Climbing Approach to Software Module Clustering," *Proc. IEEE Int'l Conf. Software Maintenance (ICSM 03)*, IEEE CS Press, 2003, pp. 315-324.
53. B.S. Mitchell, M. Traverso, and S. Mancoridis, "An Architecture for Distributing the Computation of Software Clustering Algorithms," *Proc. Working Conf. Software Architecture (WICSA 01)*, IEEE CS Press, 2001, pp. 181-190.
54. S. Yoo, M. Harman, and S. Ur, "Highly Scalable Multi-Objective Test Suite Minimisation Using Graphics Cards," *Proc. 3rd Int'l Symp Search-Based Software Eng. (SSBSE 11)*, Sept. 2011; www.springerlink.com/content/5381g43g1pp41811.
55. P. Funes et al., "Interactive Multiparticipant Task Allocation," *Proc. IEEE Congress on Evolutionary Computation (CEC 04)*, IEEE Press, 2004, pp. 1699-1705.
56. C.L. Simons, I.C. Parmee, and R. Gwynllwy, "Interactive, Evolutionary Search in Upstream Object-Oriented Class Design," *IEEE Trans. Software Eng.*, vol. 36, no. 6, 2010, pp. 798-816.
57. P.K. Lehre and X. Yao, "Runtime Analysis of Search Heuristics on Software Engineering Problems," *Frontiers of Computer Science in China*, vol. 3, no. 1, 2009, pp. 64-72.
58. E.K. Burke et al., "A Graph-Based Hyper-Heuristic for Educational Timetabling Problems," *European J. Operational Research*, vol. 176, no. 1, 2007, pp.177-192.
59. M. Harman et al., "Optimizing for the Number of Tests Generated in Search-Based Test Data Generation with an Application to the Oracle Cost Problem," *Proc. 3rd Int'l Workshop Search-Based Software Testing (SBST 10)*, IEEE CS Press, 2010, pp. 182-191.
60. G. Fraser and A. Zeller, "Mutation-Driven Generation of Unit Tests and Oracles," *Proc. 19th Int'l Symp. Software Testing and Analysis (ISSTA 10)*, ACM Press, 2010, pp. 147-158.
61. H. Wada et al., "Evolutionary Deployment Optimization for Service-Oriented Clouds," *Software Practice and Experience*, vol. 41, no. 5, 2011, pp. 469-493.
62. P. Massey, J.A. Clark, and S. Stepney, "Human-Competitive Evolution of Quantum Computing Artefacts by Genetic Programming," *Evolutionary Computation*, vol. 14, no. 1, 2006, pp. 21-40.
63. R.J. Hall, "A Quantum Algorithm for Software Engineering Search," *Proc. Automated Software Eng. (ASE 2009)*, IEEE CS Press, 2009, pp. 40-51.
64. Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," *IEEE Trans. Software Eng.*, 2011, doi:10.1109/TSE.2010.62.
65. S. Yoo and M. Harman, "Regression Testing Minimisation, Selection and Prioritisation: A Survey," *J. Software Testing, Verification and Reliability*, 2011, doi:10.1002/stvr.430.
66. M. Harman and J. Clark, "Metrics Are Fitness Functions Too," *Proc. 10th Int'l Software Metrics Symp. (Metrics 04)*, IEEE CS Press, 2004, pp. 58-69.
67. M. Harman, "Search-Based Software Engineering for Program Comprehension," *Proc. 15th Int'l Conf. Program Comprehension (ICPC 07)*, IEEE CS Press, 2007, pp. 3-13.
68. M. Harman, "The Relationship Between Search-Based Software Engineering and Predictive Modeling," *Proc. 6th Int'l Conf. Predictive Models in Software Eng. (PROMISE 10)*, 2010; www.cs.ucl.ac.uk/staff/mharman/promise-keynote.pdf.
69. M. Harman, "Open Problems in Testability Transformation," keynote, *Proc. 1st Int'l Workshop Search-Based Testing (SBT 08)*, 2008; http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4567008.
70. M. Harman et al., "Search-Based Software Engineering: Techniques, Taxonomy, Tutorial," *Empirical Software Engineering and Verification: LASER 2009-2010*, B. Meyer and M. Nordio, eds., Springer, 2012.
71. M. Harman, "The Current State and Future of Search-Based Software Engineering," *Proc. Future of Software Eng. (FOSE 07)*, IEEE CS Press, 2007, pp. 342-357.

Mark Harman leads the Software Systems Engineering Group and is director of the Centre for Research on Evolution Search and Testing in the Department of Computer Science at University College London. His research focuses on source code analysis and testing, and he was instrumental in founding the field of search-based software engineering. Contact him at mark.harman@ucl.ac.uk.

CVPR 2012 Call for Papers

25th IEEE International Conference on Computer Vision and Pattern Recognition

18-20 June 2012

Providence, Rhode Island, USA

CVPR is the premiere annual computer vision event. With its high quality and low cost, it provides an exceptional value for students, academics, researchers and industry professionals interested in any aspect of computer vision & pattern recognition.

Papers due by 21 November 2011

<http://www.cvpr2012.org/>



SC11

International Conference for High Performance Computing, Networking, Storage and Analysis

12-18 November 2011

Seattle, Washington, USA

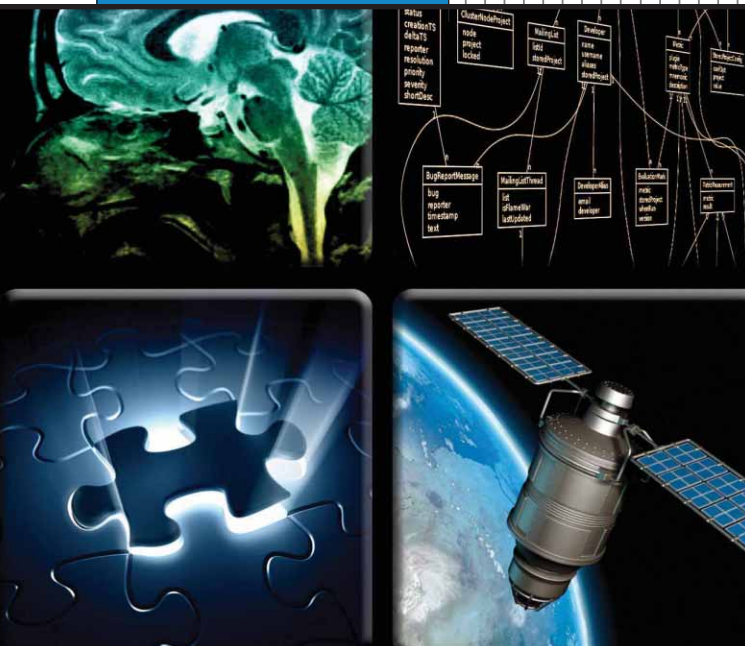
SC11 conference continues a long and successful tradition of engaging the international community in high performance computing, networking, storage and analysis.

Register today!

<http://sc11.supercomputing.org/>



COVER FEATURE



Software Engineering for Space Exploration

Robyn Lutz, *Iowa State University and Jet Propulsion Laboratory, California Institute of Technology*

Software engineering offers tools and techniques that improve the odds spacecraft will survive long missions, contributing to their resilience to new environmental challenges and to their adaptability to updated mission requirements.

Innovations in robotic spacecraft are transforming space exploration. Spacecraft are becoming more software intensive to support smarter remote control, faster and more accurate troubleshooting during flight, and improved scientific data gathering and analysis to gain a better understanding of our universe.

NASA spacecraft launched or set to launch this year have missions to map the moon's gravitational field (Gravity Recovery and Interior Laboratory, GRAIL; <http://science.nasa.gov/missions/grail>), study how Jupiter and other giant planets formed (Juno; <http://science1.nasa.gov/missions/juno>), and land a new robotic rover, Curiosity, to investigate the Martian environment (Mars Science Laboratory, MSL; <http://mars.jpl.nasa.gov/msl>).

Despite engineers' best efforts, exploration is perilous for robotic spacecraft in both near-Earth and deep space environments. Some environmental factors are hostile to onboard software systems. For example, radiation-induced bit flips can cause software to fail or operate improperly. While some surprises are welcome—for example, *Voyager 2* witnessed a rare supernova—other unexpected events or circumstances can cripple or destroy systems.

SOFTWARE ENGINEERING: A VITAL ROLE IN SPACE EXPLORATION

A variety of software supports space exploration. Flight software guides planetary orbiters, lander spacecraft, and space probes; navigation and hazard-avoidance software drive Martian rovers; onboard software acquires scientific data from rovers and instruments and transmits it back to Earth for further processing; operational software on Earth monitors and controls the missions; and analytical software helps scientists to visualize and interpret the results, and to apply the newly discovered knowledge to other problems on Earth.

Software engineering plays a vital role by providing tools, techniques, and a systematic approach to developing and servicing the many different kinds of software needed for space exploration. Advances such as model-based software development, formal verification, and product-line development help ensure that the software meets mission needs and that spacecraft operate safely and correctly.

Robotic spacecraft—unmanned vehicles controlled remotely by software commands sent from Earth or encoded in programs onboard the spacecraft—are the advance guard of space exploration because they can go to places well beyond the reach of human missions. Because many robotic spacecraft travel far and need to survive a long time, they must be exceedingly robust. To detect and respond to anomalies that occur during operations, their software is highly fault tolerant. Spacecraft teams also can update the software in flight to adapt to changes in the mission's needs or to accommodate component failure.

COVER FEATURE

DEEP SPACE: THE FARTHEST HUMAN-MADE OBJECTS

The two Voyager spacecraft, launched in 1977, are now the farthest human-made objects from Earth. They continue to return invaluable data as they approach the heliopause, the boundary of the sun's solar winds (<http://voyager.jpl.nasa.gov>).

Voyager 2 recently discovered a strong magnetic field that holds together the interstellar cloud. For the spacecraft to get that far, mission engineers had to update its software during flight.

Soon after launch, *Voyager 2*'s primary receiver failed, and its backup receiver was reduced to a very narrow, changing frequency band. Software engineering practices helped the mission team respond quickly to the problem. They designed and implemented a new algorithm that enabled ground operations to tune the transmission of software commands to the receiver's current state.

Among the software updates was a new algorithm required to obtain crisp images of Neptune at low sunlight levels, like that in Figure A. The algorithm automatically fired the spacecraft's attitude jets to compensate for torque imparted at the longer exposure rate.



Figure A. Neptune clouds, taken by the *Voyager 2* spacecraft two hours before closest approach. Image courtesy of NASA/JPL-Caltech.

While the focus here is on robotic spacecraft developed or managed by NASA's Jet Propulsion Laboratory (JPL), other NASA facilities as well as international space agencies (http://en.wikipedia.org/wiki/List_of_space_agencies), commercial organizations, and research institutions produce software for space exploration missions. Limited budgets, different areas of expertise, risk sharing, and cooperative opportunities all encourage collaboration among multiple entities to achieve what no single one could. Multinational collaboration has become a hallmark of space exploration. For example, the 2016-2018 Exobiology on Mars (ExoMars) Trace Gas Orbiter is a joint European Space Agency/NASA-JPL mission that will study

the sources of methane and other gases on Mars (<http://exploration.esa.int>).

DESIGNING FOR CHANGE

A space exploration mission typically evolves over time. Software systems must be designed to accommodate this change, especially on long-lived spacecraft traveling to distant planets. The *New Horizons* spacecraft launched in 2006 to study Pluto and its moons will not reach its destination until 2015 (<http://pluto.jhuapl.edu>).

Software engineering for spacecraft thus devotes considerable effort to anticipating and planning for change. Some changes involve updates to requirements; others entail updates to onboard software technology, such as improved image compression algorithms or additional autonomous reconfiguration options. The spacecraft must also be robust enough to maintain its essential functional capabilities despite failures.

Mission teams regularly update spacecraft software during operations, fixing bugs, uploading adaptations to contextual changes, and enhancing what the software can do or how well it can do it. On long-distance missions, they often develop the software needed for later surface or orbit operations during flight. Changes and augmentation after launch let engineers sequentially update and customize software to the mission's next phase. The "Deep Space: The Farthest Human-Made Objects" sidebar describes instances in which *Voyager 2* engineers had to develop new algorithms for ground and flight software.

The ability to update software during operations is essential. A study of 199 critical, software-related post-launch anomalies on seven spacecraft revealed that about 7 percent resulted in a new software requirement either to handle a rare but high-consequence event—for example, failure of redundant units—or to accommodate hardware failures or limitations.¹ Such requirement changes are typical of systems in which the hardware is inaccessible (such as spacecraft) or difficult to access (such as implanted medical devices), or where the operating environment is dangerous (for example, high radiation levels or extreme temperatures). In such systems, software often must be updated to compensate for hardware failure or degradation.

Many spacecraft and rovers have excellent hardware and software reliability, allowing them to far exceed their planned mission lifetimes. For example, the twin Mars Exploration Rovers, Spirit and Opportunity, finished their original 90-day primary mission in 2004 but, 90 months later, Opportunity still continues its trek across Mars (<http://marsrover.nasa.gov>).

The software architecture for spacecraft is thus designed to accommodate changes. Given the turnover in engineering personnel that can occur over a mission's

lifetime and the likelihood that altered circumstances will result in modifications to the software requirements, it is important to retain the reasoning behind design decisions and flight rules. This helps prevent the introduction of errors when software is updated.

FAULT TOLERANCE

Fault tolerance is a specialized kind of planning for change. Given the long lives of spacecraft and the hostile environments they encounter, failures are expected. Developing robust software that can handle failures and other conditions that threaten the mission begins with a thorough systems and software hazards analysis. Developers derive software requirements to protect the spacecraft from mission-critical failures from these hazards analyses.

An example of successful fault tolerance occurred when the *Cassini* spacecraft, currently on an extended mission to Saturn and its satellites (<http://saturn.jpl.nasa.gov>), ceased communication with ground operations on 2 November 2010. The *Cassini* team later determined that this was caused by a flipped bit. The onboard fault-protection software quickly switched to a backup computer, shut off nonessential power loads, and activated an alternate low-gain antenna, pointing it toward the sun to improve the chances of reaching human operators. Communication was restored within the hour, at which point the operators began a slow and careful check of the spacecraft, followed by recovery activities such as clearing error log files and reactivating scientific instruments. These actions were completed on 24 November, and *Cassini* continues its science mission today.

Fault-protection software on spacecraft continually monitors for discrepancies between the expected and actual state. This software is especially important because the system usually invokes it when something has already gone wrong. As spacecraft and space exploration missions become more complicated, the number of potential undesirable scenarios increases. Software must handle not just failures, as was the case with the *Cassini* bit flip, but also a range of novel usage situations and unexpected contingencies, such as debris temporarily blinding the spacecraft camera. However, extending onboard fault-protection techniques to cover a wider range of software problems can also increase algorithmic complexity, which makes verifying the software more difficult.²



Figure 1. Mars rover evolution. From left to right: one of the twin 2004 Mars Exploration Rovers (Spirit and Opportunity); the 1997 Mars Pathfinder Rover; and the 2011 Mars Science Laboratory Rover, Curiosity. Image courtesy of NASA/JPL-Caltech.

AUTONOMY

To accommodate the long distances between spacecraft and Earth, ground operators must give the spacecraft some autonomy. The round-trip light time (RTLTL) to send a radio signal command to *Cassini* and receive an acknowledgment is more than two and a half hours. Many activities must be accomplished much faster than this, forcing human operators to provide limited real-time control to the remote robotic spacecraft's software.

For example, when the spacecraft for the Mars Exploration Rovers (MER) mission landed in 2004, onboard software used radar, image, and sensor data to track and respond to wind velocity. "Had DIMES [the Descent Image Motion Estimation System] not been available to measure the steady state velocity," a subsequent report noted, "the EDL [Entry-Descent-Landing] system would not have fired TIRS [the retro rockets] and the total velocity would have been just on the threshold of airbag performance. Furthermore, the velocity would have been to the East toward the rockier terrain."³

The MER Opportunity Rover, for which the RTLTL is currently about 40 minutes, routinely drives itself short distances. The MSL Curiosity Rover, to be launched later this year, has autonomous navigation software that allows it to devise a plan and drive to an interesting feature as far as 50 meters away while avoiding hazards in its way. Figure 1 shows how the Mars rovers have evolved.

The Autonomous Sciencecraft Experiment (ASE; <http://ase.jpl.nasa.gov>), a co-winner of NASA's 2005 Software of the Year, has been running on the Earth Observing-1 (EO-1) satellite for several years. ASE decides what scientific observations to make and what data to downlink to Earth. When Antarctica's Erebus Volcano erupted in

COVER FEATURE

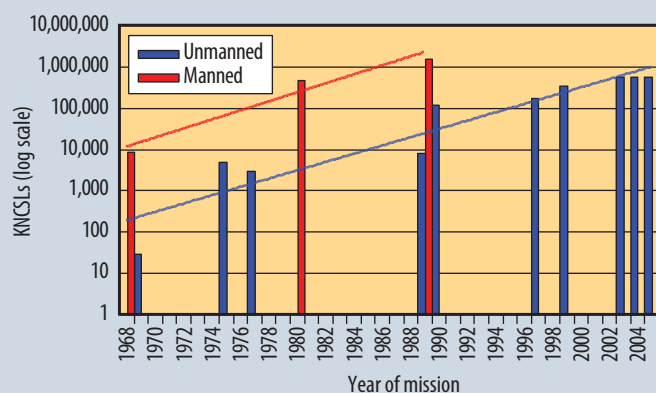


Figure 2. Growth in code size for NASA missions. Size has increased by a factor of 10 every 10 years. Figure from D.L. Dvorak, ed., *NASA Study on Flight Software Complexity*, 3 Mar. 2009, NASA Office of Chief Engineer; http://www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf.

2004, the software detected it and prompted EO-1 to collect additional event data within six hours; without ASE self-direction, the response time would have been from 12 to 26 days. Two years later, acting as part of NASA's Volcano Sensor Web, EO-1 collected precise data regarding the eruption and expected direction of the lava flow from the Nyamuragira Volcano near a populated area in the eastern Democratic Republic of the Congo to help planners on the ground.⁴

The autonomy ASE provides was estimated to reduce EO-1 mission costs by more than \$1 million per year. Software engineering techniques that contribute to its high reliability include a redundant, layered architecture and supporting models; a rigorous systems and requirements engineering process; a multilevel testing process; and incremental spiral development.⁵

GRAPPLING WITH GROWTH

A simple system is easier to understand and hence easier to build correctly, test, and update as needed over time. However, improved knowledge discovery will depend on more and better data, and on smarter spacecraft to acquire and process such data. Making spacecraft smarter in terms of adaptability (to respond to new mission requirements), fault tolerance (to isolate and recover quickly from failures), and autonomy (to make onboard decisions without human intervention) encourages the development of bigger and more complicated software.

As Figure 2 shows, the size of human and robotic NASA missions from 1968 through 2005, as measured in thousands of noncommentary source lines (KNCSLs) of code, has increased by a factor of 10 approximately every 10 years.² One of the key challenges of software engineering for spacecraft is how to handle this exponential growth

in code size. Unlike other systems, once a spacecraft has been launched, there is no way to pull it back into the shop to repair it. Smarter but more fragile software is an unacceptable tradeoff.

An example of how easy it is to introduce a critical design flaw and how hard it can be to fix it in flight is the “Sol 18” anomaly that occurred on the MER Spirit rover on 21 January 2004. An error in a commercial off-the-shelf (COTS) file services module allowed incorrect dynamic memory allocation during operation. Two configuration errors then compounded the impact of this design flaw. As a result, out-of-memory events caused repeated processor resets that interrupted communication with operators on Earth and depleted battery power, threatening the spacecraft's mission.

A protective feature in the software design called “crippled mode” was essential to the rover's eventual recovery. This contingency mode enabled the MER

team to prevent continuous resets and, after two weeks of intensive team effort, restore the spacecraft's functionality. “Without this capability, the MER mission may well have been lost,” noted team members Glenn Reeves and Tracy Neilson. “The crippled mode mechanism allowed us to regain control of the vehicle.”⁶

This incident highlighted the importance of assuring compaction for deleted files in the directory structures for long-duration missions, the difficulty of understanding and managing all interactions, the risk of unexpected behavior by COTS software, the criticality of verifying operational assumptions, the need for telemetry that reveals the current software state, the impracticality of operational tests long enough to reveal such behavior (in this case, over 11 sols, or Martian days), and the central role of automated test data analysis.

SOFTWARE ENGINEERING IN EMBEDDED SYSTEMS

Software engineering goes hand-in-hand with systems engineering on robotic spacecraft. As part of a larger system, the software monitors and controls the devices in the system. Software engineering is fundamental to the integrated infrastructure that makes spacecraft exploration possible.

Experience with the *Galileo* spacecraft exemplifies how any software solution must take into account broader system and environmental interactions. The spacecraft's high-gain antenna (HGA), which had been folded up like an umbrella during launch in 1989, failed to unfurl when commanded to do so in 1991. During the four years before *Galileo's* arrival at Jupiter, engineers made extraordinary efforts to open the antenna. Without the HGA, the spacecraft could transmit to Earth only a small percentage of the scientific data it collected.

The solution involved what team members Erik Nilsen and Trisha Jansma called the “first complete reload of flight software ever performed on a deep-space mission.”⁷ But recovery from the HGA anomaly required more than just a software solution. The workarounds “were truly a team effort involving a system approach that included science, flight, ground, hardware, and software.” High-rate data from the Jupiter encounter would need to be buffered so that it could slowly trickle to Earth over the low-gain antenna (LGA).

To accomplish this, the mission team uploaded new data-processing and data-compression software that let *Galileo* send 10 times the number of images and data than was otherwise possible without the HGA. They also made hardware and software improvements to the deep-space network that the spacecraft used to communicate with Earth, including advanced antenna arraying, low-noise receivers, and improved modulation schemes. These changes greatly increased the amount of data that could be received from *Galileo*. While the data returned did not equal what was envisioned in the original plan, the effective data downlink increased more than 400 times, from 10 bits per second (the LGA rate) to about 4.5 Kbps, enabling most of the mission’s goals to be achieved.

OPPORTUNITIES AND CHALLENGES


Many of the software engineering techniques (such as hazards analysis, requirements-based testing, and formal verification) used for other safety-critical systems (such as airplanes, cars, pacemakers, power plants, and factory-floor robots) have been adapted and extended for space exploration. The flow of ideas is two-way, resulting in many market spin-offs.

Software engineering solutions for robotic spacecraft that have been applied to other similar systems include model-based diagnostics and recovery, design for reuse, risk management, delay-tolerant network software (which preserves loss-less communication even if communication is slow or interrupted), distributed sensor network technologies, redundancy management, product-line engineering, and resource-constrained (power, memory, mass, bandwidth) system design.

JPL’s Laboratory for Reliable Software (LaRS; <http://lars-lab.jpl.nasa.gov>) develops technologies to improve software on spacecraft and other mission elements. Many of these advances can increase software robustness and productivity in other critical as well as noncritical domains. Examples include model checking to verify critical components and interfaces such as the MSL spacecraft’s flash file system for flight, static source code analyzers as standard practice, and institutional coding standards that can reduce risk related to multithreaded software. LaRS also developed a software tool, called SCRUB, to support code reviews. The SCRUB software combines results

from multiple source code analyzers with results from code walk-throughs to make the review process more efficient and the code more reliable. It has been used on the MSL project for three years for all reviews of both manually written code and code autogenerated from high-level formats.⁸

The long operational lifetimes of many space exploration missions inevitably involve personnel turnover. This means that the documentation, preservation, and maintenance of knowledge about the spacecraft are especially important for such systems. Software product-line engineering techniques offer a promising means to help software designers of single, long-lived robotic spacecraft identify options, document assumptions, and make decisions regarding alternatives and changes, based on cost and value, during both development and operations.⁹ By modeling anticipated change as variability, making such a change after launch is comparable to making different choices for a new product.



Many findings and techniques from developing software for space exploration missions transfer readily to other safety-critical systems.

Spacecraft mission design balances highly innovative technologies that promise more and better scientific data with an aversion to risk driven by unique opportunities. For example, only about once every 175 years do the planets Jupiter, Saturn, Uranus, and Neptune line up so that a single spacecraft can fly past all of them, as *Voyager 2* did.

More broadly, many findings and techniques from developing software for space exploration missions transfer readily to other safety-critical systems:

- rare events do occur and must be planned for,
- static analysis and requirements-based testing significantly reduce operational defects,
- overly strict requirements unnecessarily add complexity and consequent risk,
- traceability to requirements and rationale forestalls injection of design defects,
- it is important to design not just for failures but also for undesirable contingencies,
- restraint in the introduction of complexity reduces risk,
- requirements always change over the system’s lifetime,
- partial autonomy enables scientific gains not otherwise possible because of long communication times,

COVER FEATURE

- time invested in a well-thought-out architectural design pays off in increased sustainability,
- there is no substitute for expertise and the power of a committed design team, and
- model-based engineering and good traceability can reduce loss of knowledge over the lifetime of a long-lived system.

Despite the achievements of software engineering for space exploration, many challenges remain. One that is essential to address is the analysis, visualization, and management of very large amounts of scientific data. For example, the Mars Reconnaissance Orbiter (<http://mars.jpl.nasa.gov/mro>), which was launched in August 2005 and went into orbit the following March, gathered 100 terabits of data in its first four years. This was more than three times the data collected up to then by all other deep-space robotic spacecraft combined. Another area of interest is the software engineering of multiple, distributed, formation-flying spacecraft, perhaps with some having specialized scientific instruments that can cooperate to learn about space in ways not previously possible.

As the capabilities of robotic spacecraft increase, the need for software engineering for space exploration will continue to grow. “Exploration is in our nature,” Carl Sagan famously said. “We began as wanderers, and we are wanderers still. We have lingered long

enough on the shores of the cosmic ocean. We are ready at last to set sail for the stars.”¹⁰ Or, as one JPL engineer put it: “If you can’t have a good time coming to work and building robots to send to Mars, give it up, man.”¹¹ **□**

Acknowledgments

Thanks to Margaret Smith, Martin Feather, and Scott Morgan for additional examples. This work was supported in part by National Science Foundation grant 0916275 with funds from the American Recovery and Reinvestment Act of 2009.

References

1. R.R. Lutz and I.C. Mikulski, “Empirical Analysis of Safety-Critical Anomalies during Operations,” *IEEE Trans. Software Eng.*, Mar. 2004, pp. 172-180.
2. D.L. Dvorak, ed., *NASA Study on Flight Software Complexity*, 3 Mar 2009, NASA Office of Chief Engineer; http://www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf.
3. A.E. Johnson et al., “Design through Operation of an Image-Based Velocity Estimation System for Mars Landing,” *Int’l J. Computer Vision*, Sept. 2007, pp. 319-341.
4. M. Scott, “Observing Volcanoes, Satellite Thinks for Itself,” 6 Dec. 2007, NASA Earth Observatory; <http://earthobservatory.nasa.gov/Features/VolcanoSensorWeb>.
5. S. Chien, “Integrated AI in Space: The Autonomous Spacecraft on Earth Observing One,” *Proc. 21st Nat’l Conf. Artificial Intelligence and 18th Innovative Applications of Artificial Intelligence Conf.* (AAAI 06), AAAI Press, 2006; www.aaai.org/Papers/AAAI/2006/AAAI06-238.pdf.
6. G. Reeves and T. Neilson, “The Mars Rover Spirit FLASH Anomaly,” *Proc. 2005 IEEE Aerospace Conf.*, IEEE Press, 2005, pp. 4186-4199.
7. E.N. Nilsen and P.A. Jansma, “Galileo’s Rocky Road to Jupiter,” *ASK Magazine*, vol. 42, spring 2011; www.nasa.gov/offices/oc/e/appel/ask/issues/42/42s_galileo_rocky_road_jupiter.html.
8. G.J. Holzmann, “SCRUB: A Tool for Code Reviews,” *Innovations in Systems and Software Eng.*, Dec. 2010, pp. 311-318.
9. R. Lutz et al., “Software Product Line Engineering for Long-Lived, Sustainable Systems,” *Proc. 14th Int’l Software Product Line Conf.* (SPLC 10), Springer, 2010, pp. 430-434.
10. C. Sagan, *Cosmos*, Random House, 1980.
11. J. Kluger, “Scientific Illiteracy After the Shuttle: Are America’s Smartest Days Behind Her?,” *Time*, 11 July 2011; www.time.com/time/health/article/0,8599,2082213,00.html.

Robyn Lutz is a professor of computer science at Iowa State University and a member of the software system engineering group at Jet Propulsion Laboratory, California Institute of Technology. Her research interests include software engineering of safety-critical systems, product lines, and the specification and verification of requirements for autonomous systems. Lutz is a senior member of IEEE. Contact her at rlutz@iastate.edu.

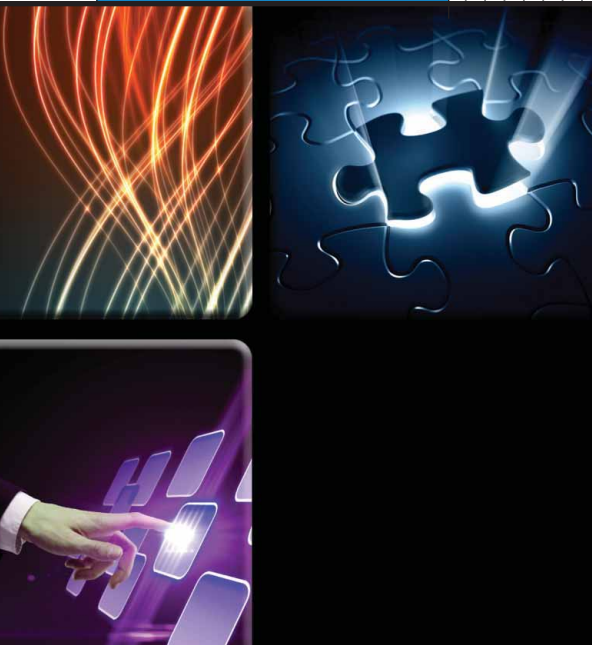


Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.

Computer

 NEXT ISSUE
 APPLICATION
 AND CODESIGN
 ARCHITECTURES

COVER FEATURE



Software Engineering Meets Services and Cloud Computing

Stephen S. Yau and Ho G. An, *Arizona State University*

Service-oriented software engineering incorporates the best features of both the services and cloud computing paradigms, offering many advantages for software development and applications, but also exacerbating old concerns.

Services and cloud computing have garnered much attention from both industry and academia because they enable the rapid development of large-scale distributed applications in areas such as collaborative research and development, e-business, healthcare, grid-enabled applications, enterprise computing infrastructures, military applications, and homeland security. These computing paradigms have made it easier and more economical to create everything from simple commercial software to complex mission-critical applications.

The two paradigms share concepts, such as resource outsourcing and transfer of IT management to service providers, but their emphasis on software engineering differs. Services computing focuses on architectural design that enables application development through service discovery and composition. Cloud computing focuses on the effective delivery of services to users through flexible and scalable resource virtualization and load balancing.

Service-oriented software engineering¹ incorporates the best of these two paradigms. Initially, SOSE was based on services computing, but it evolved to include cloud

computing. In SOSE, a service-oriented architecture (SOA) provides the architectural style, standard protocols, and interfaces required for application development, and cloud computing delivers the needed services to users through virtualization and resource pooling. Combining services and cloud computing in a software engineering framework can help application developers and service providers meet the individual challenges of each paradigm.

Although SOSE is conceptually promising, its realization will require additional research in software engineering to address the challenges, such as security and quality-of-service (QoS) management, that arise in services or cloud computing.

SERVICES COMPUTING

Service developers follow SOA, an architectural model for creating and sharing computing processes, packaged as services.² Each service is an independent software entity with a well-defined standard interface that provides certain functions over networks. Developers can dynamically compose services as a workflow, which forms the basis of an application. In this context, software itself can be a service—a self-contained, stateless, and platform-independent entity with a URL, an interface, and functions that can be described and discovered as XML data.

Different organizations with different policies develop, manage, and govern services. Service-level agreements specify runtime requirements that govern a service's interactions with a user or with other services. A service's SLA describes that service and sets forth the terms, in es-

COVER FEATURE

Table 1. Protocols and interfaces that enable SOSE.

Protocol or interface	Standards body (if applicable)	Purpose
XML	W3C	Represent data in SOA
Simple Object Access Protocol (SOAP)	OASIS	Invoke services remotely across networks and platforms
Representational State Transfer (REST)	Architectural style, not a standard (attributed to Roy Fielding)	Invoke services remotely across networks and platforms
Web Services Description Language (WSDL)	W3C	Describe interfaces and service functions
Universal description, discovery, and integration (UDDI)	OASIS	Automatically publish and discover services
Electronic Business XML (ebXML)	OASIS and United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT)	Automatically publish and discover
Business Process Execution Language (BPEL)	OASIS	Orchestrate services in a workflow

sence becoming a service contract that service providers must fulfill.

Using standard protocols and interfaces, application developers can dynamically search, discover, compose, test, verify, and execute services in their applications at runtime. SOA-based application development is through service discovery and composition, which involves three stakeholders:

- A *service provider* (or developer) is the party who develops and hosts the service.
- A *service consumer* is a person or program that uses a service to build an application.
- A *service broker* helps service providers publish and market their services and helps service consumers discover and use the available services.

Application developers need not integrate service code into applications because the service runs at its provider's site and is loosely coupled with applications through standard messaging protocols. Consequently, services and applications do not have to be in the same programming language or run on the same platform. Unlike an application, which provides a user interface, a service typically provides an application programming interface (API) so that an application or other services can invoke that service.

As this description implies, services have several attractive characteristics. They are

- *loosely coupled*—there are no direct dependencies among individual services;
- *abstract*—beyond the SLA description, a service hides its logic from the outside world;
- *reusable*—services aim to support potential reuse;
- *composable*—a service can comprise other services, and developers can coordinate and assemble services to form a composite;

- *stateless*—to remain loosely coupled, services do not maintain state information specific to an activity, such as a service request; and
- *discoverable*—services let a service consumer use mechanisms to discover and understand their descriptions.

When taken together, these characteristics empower the rapid development of applications in services computing.

Standards bodies, such as the Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium (W3C) have established a variety of protocols and service interfaces that enable application development using SOA. Table 1 gives a sampling of these protocols and interfaces.

CLOUD COMPUTING

Cloud computing enables convenient, on-demand network access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services, which the cloud system can rapidly provision and release automatically. Cloud computing lets a consumer (user or program) request computing capabilities as needed, across networks anytime, anywhere. Some researchers envision the future Internet as a kind of supercomputer that will depend heavily on cloud computing features, such as resource pooling and virtualization, on-demand service, and ubiquitous access.

There are four cloud types. A *public cloud* provides services to the public. A *private cloud* provides services to only users within one organization. A *community cloud* provides services to a specific community of organizations and individuals. A *hybrid cloud* is any combination of the first three types.

As Figure 1 shows, the cloud computing architecture has three layers: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Developers can implement and use each layer as a service.

Software as a service

Software that performs various tasks is not on the client machine. Instead, third-party service providers host and manage the software services in the cloud. SaaS includes both software components (for application developers) and applications (for users). An SaaS application is often a service-oriented program so that it is easy to integrate with other SaaS applications.

Platform as a service

PaaS provides a development platform with services to assist application design, implementation, testing, deployment, monitoring, and hosting in the cloud. It requires no software download or installation and supports geographically distributed collaborative work.

Infrastructure as a service

IaaS virtualizes the data centers' computing power, storage, and network connectivity. Users can scale these computing resources up and down on demand.

APPLICATION DEVELOPMENT WITH COMBINED PARADIGMS

Services computing and cloud computing are two separate paradigms, and each provides many advantages for software development and application. Application developers can use services computing alone, cloud computing alone, or a combination of the two. We believe that combining these two paradigms in a software engineering framework will help alleviate some of the software engineering challenges that services and cloud computing have individually. For example, a major challenge of services computing is to manage the runtime QoS of loosely coupled services involving distributed service providers. Cloud computing can help meet that challenge through resource allocation and virtualization.

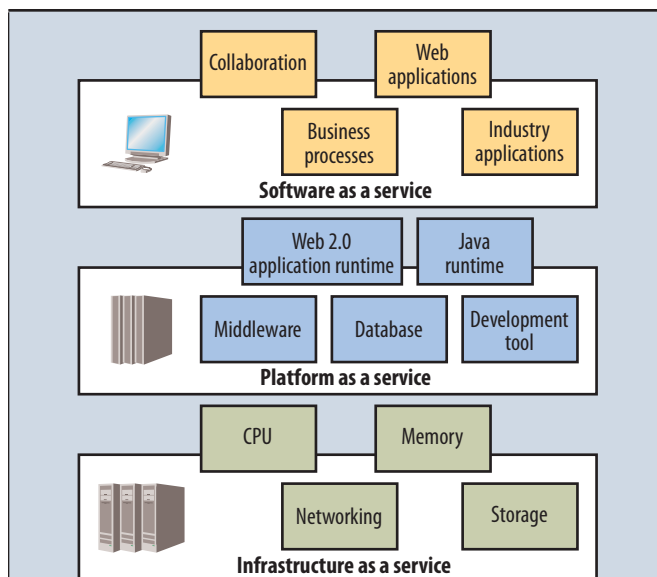


Figure 1. Cloud computing architecture. The top layer allows users and application developers to access services that third parties host and manage. The second layer consists of computing platforms and development services, and the third layer provides the computing resources that users can tap on demand.

On the other hand, cloud computing struggles both with providing interoperability across different clouds and with the rapid development of, and adaptation to, ever-changing business environments and requirements. SOA's standard interfaces and protocols could help address this interoperability challenge, while its dynamic service discovery and composition can provide the capabilities needed for dynamic adaptation in cloud computing environments.

Figure 2 shows the concept of developing applications using SOA and delivery through the cloud. Service providers could publish SaaS, PaaS, IaaS, and software

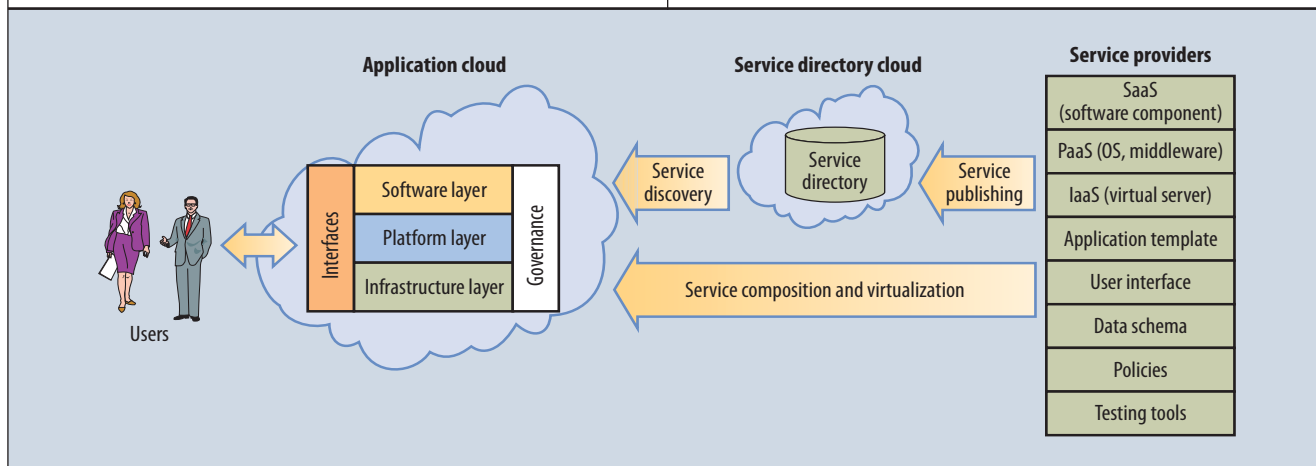


Figure 2. The SOSE vision. Service providers publish their services to a federated cloud directory. Application developers discover and compose the services to build an application, which is delivered to users through the cloud.

COVER FEATURE

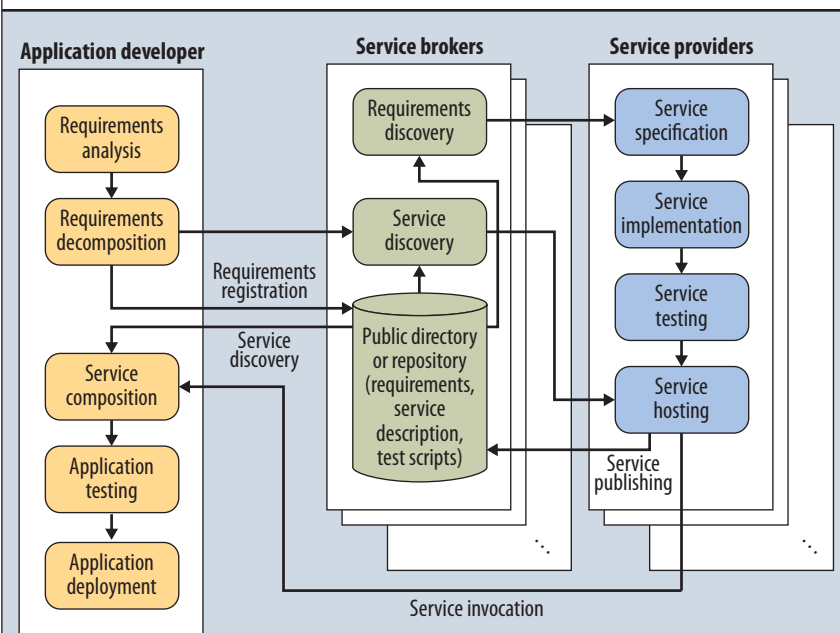


Figure 3. Interactions among an application developer, service brokers, and service providers in SOSE. The challenge for developing applications using SOA is to cope with the distributed nature of both the stakeholders and their activities, which can be in different organizations and locations.

artifacts, such as application templates, user interfaces, data schema, policies (including security policies), and testing tools in a service directory cloud. This federated service directory cloud could enable application developers to dynamically discover services in multiple distributed servers and compose these services using SOA and virtualization technologies.

Development with a service-oriented architecture

SOSE applies SOA to software development life-cycle stages, producing a cycle that includes not only the traditional requirements specification, design, and test phases, but also service implementation, discovery, and composition. Application development in SOA is different from software development approaches such as object-oriented programming, component-based software development, and aspect-oriented programming. The construction of an application from smaller software components in other software development methodologies is static and manual and depends on the components' technology and platform. In contrast, service composition in SOA is automated through standard protocols and interfaces, and thus does not depend on a specific technology or platform. In addition, service development, service publishing, and service composition (or application building) are parallel processes in SOA.

The challenge for developing applications using SOA is in addressing its distributed nature. Not only are the ser-

vices under development distributed among different machines in various locations, but the development process is also distributed because the application developers, service brokers, and service providers work independently in different locations. Hence, these three stakeholders must collaborate through well-defined standards and interfaces. Figure 3 shows some key tasks and interactions among these individuals.

As in other software development methodologies, SOSE starts with requirements engineering. During this phase, the application developer develops a business model; works with the customer to analyze, clarify, and refine requirements; designs a workflow for the business model; and decomposes requirements.

The application developer then sends each decomposed part of the requirements to the service broker to find available services that satisfy these requirements parts. After successfully

discovering all the needed services to satisfy each part, the application developer selects the needed services for all requirements parts and composes them into an application, essentially the business model workflow.

If no services are available for some parts, the application developer can register them in the service broker's directory and wait until the needed services are available. From the service providers' view, service development is similar to what happens in other software development processes, except that services must also comply with standard protocols and interfaces.

Software development in SOSE is highly flexible because SOA makes it possible to publish and reuse not only software services, but also numerous application development artifacts. Application developers can publish business models, application templates (workflow structures), requirements, services, application interfaces, testing tools, testing scripts, and policies in a service broker's directory, making them available for reuse. This flexibility facilitates the rapid development of large-scale distributed applications.

Delivery through cloud computing

Software engineering must address not only the software development processes, but also the effective delivery of the developed software to users, which includes software deployment and maintenance. However, SOA does not address how a developed application is to be delivered to users or how service providers will effectively manage

the applications during runtime. Cloud computing can help SOSE ensure effective application delivery by providing

- easy application deployment and maintenance for service providers through service virtualization,
- interfaces to facilitate users' access to and use of applications, and
- QoS management for service providers through dynamic resource virtualization and allocation.

These features illustrate the power of combining SOA-based development with cloud computing delivery.

APPLICATION DEVELOPMENT CHALLENGES

Achieving the vision of application development that Figure 2 depicts requires new approaches to effective virtualization and interoperability among SaaS, PaaS, and IaaS. It also requires revisiting software engineering issues, some of which are not new, but they are more severe in the context of services and cloud computing. We have identified seven areas that pose major challenges for application development using SOSE.

Confidentiality and integrity

In cloud computing, users have little control over data processing and storage, which is on remote machines that various service providers own and operate. Because this data is unencrypted, there is a risk that service providers or malicious users could disclose or alter it. Although techniques exist for confidentiality protection, they are not applicable to services and cloud computing systems because they are designed to protect data from malicious parties outside the systems. Services and cloud computing systems have many service providers inside the systems.

Thus, existing techniques for access control, identity management, end-to-end data confidentiality, and integrity assurance systems are not suitable. Although research is already addressing software engineering techniques for data confidentiality and integrity protection for services and cloud computing systems,^{3,4} more work is needed in this area.

Service reliability and availability

Because users' businesses rely heavily on third-party service providers, there are serious concerns about how threats to service reliability and availability—from a service provider's unstable economic status to natural disasters and cyberattacks—could affect a service and consequently a cloud user's business. To alleviate these threats, service and cloud users should check their data backup plan, system robustness, contingency and recovery plans, end-of-service support, and incident history before using a particular service.

Cyberattacks are a particularly serious threat. Services and cloud computing systems rapidly and flexibly provide massive computing resources according to users' demands. For the users, the computing capabilities and resources often appear to be unlimited, since they are available for purchase at any time and in any quantity. However, cyberattackers also can buy huge amounts of computing resources, enabling them to launch more powerful cyberattacks. Attackers have already used the Amazon EC2 and Google AppEngine clouds, for example.^{5,6}

To address this problem, services and cloud service providers need effective software engineering techniques to monitor and detect malicious user activities, as well as for strict user authentication and access control.

Security in a multitenant environment

In multitenancy a single software instance runs on a server that accommodates multiple users, or tenants. In a multitenant architecture, an application virtually partitions its data and configuration, and each user works with a customized virtual application instance.

Because users' businesses rely heavily on third-party service providers, there are serious concerns about how threats to service reliability and availability could affect a service and consequently a cloud user's business.

Services and cloud computing systems have multitenancy because multiple users share the application and a set of hardware. Security vulnerabilities are a major issue. Service providers use hypervisors that mediate access between virtual machines and hardware, but some hardware, such as CPU caches and GPUs, is not designed to offer strong isolation properties for a multitenancy architecture. Even virtual machine hypervisors can have flaws that allow one user's virtual machine to gain inappropriate control over others.⁷ Recently, attackers have exploited numerous hypervisor vulnerabilities to influence other users' operations or to gain unauthorized data access.

Addressing these vulnerabilities requires developing software engineering techniques for securing multitenancy architectures in services and cloud computing systems, such as techniques for isolating and monitoring virtual machines.

Unknown risk profile

In services and cloud computing systems, users have limited access to information about the internal system architecture, software versions, configurations, operations, and security practices of service providers. This

COVER FEATURE

limited access might enhance usability, but it also has serious implications for risk management. Risk management in software engineering ensures that the application developers identify and analyze threats to the application development process and that they use appropriate strategies to mitigate and control risks, such as failing to complete projects within the specified schedules and budget constraints and not meeting user requirements.

Because application developers lack information about the internal systems beneath the virtualized abstraction layer, they might not be able to conduct appropriate risk management. To address this problem, developers should ask service providers for three items:⁸

- partial or full disclosure of software design or infrastructure details;
- disclosure of applicable logs and data, such as network intrusion logs, anomaly detection logs, and security events logs; and
- disclosure of details of security policies and enforcement mechanisms.

Having these items will not eliminate risk, but the information should lead to much more effective risk management.

Quality-of-service monitoring

In services and cloud computing systems, managing a variety of QoS requirements is extremely difficult because numerous application developers are dynamically composing services over networks to form multiple workflows, and various providers with different techniques and policies are managing the services. Consequently, the QoS features of all services are tightly interrelated, and there are tradeoffs among them.

Features like throughput and service delay rely on system resource allocation at the applications' runtime. Often, the same server hosts multiple services, which compete for the server's CPU time, memory, and network bandwidth. In addition, service compositions, server resource status, workflow priorities, and QoS requirements are usually changing dynamically at runtime.

For these reasons, satisfying the QoS requirements of multiple workflows requires having effective techniques to adaptively allocate system resources to each service. To manage multiple QoS properties for such systems, services and cloud computing systems need situational awareness, context analysis and QoS estimation, and optimal resource allocation.⁹⁻¹¹

Mobile computing

Services are available over networks, and users or programs on a range of devices—desktops, laptops, smartphones, tablets, and PDAs—can access the services on the networks at any time or in any location through standard

protocols. Because identity theft and service hijacking are major threats, mobile services and cloud computing providers need rigorous software engineering techniques to secure ubiquitous access to services and data.

Legal issues

Those who use services and cloud computing systems do not know their data's exact physical locations because data processing and storage is often at unspecified geographic locations, both domestic and foreign. Legally, each location has a different jurisdiction. Service providers in foreign countries might not always guarantee regulatory compliance, such as protecting privacy, backing up data, or providing an audit trail. They might not be willing to assume liability for security incidents or for the failure to meet data backup requirements or to provide audit trails. They also might not protect intellectual property according to compliance standards.¹²

Application developers who establish SLAs with service providers need to check if the providers will commit to storing and processing data in specific jurisdictions, and if they will make a contractual commitment to comply with all regulatory requirements and liabilities in publishing and managing applications.

Although services computing and cloud computing have great promise in meeting the increasingly severe requirements of dynamic application development and use, fully realizing this potential requires some kinds of application development structure. Software engineering can help combine these computing paradigms and harness their considerable advantages for application development. Although many challenges remain in moving this idea from vision to implementation, the benefits of such an environment should serve to motivate the software engineering research that can meet those challenges. **□**

Acknowledgments

The work described in this article was partially supported by the National Science Foundation under grant CCF-0725340.

References

1. Y. Chen and W.-T. Tsai, *Service-Oriented Computing and Web Data Management: From Principles to Development*, Kendall Hunt, 2010.
2. K. Channabasavaiah, E. Tuggle, and K. Holley, "Migrating to a Service-Oriented Architecture," 16 Dec. 2003; www.ibm.com/developerworks/webservices/library/ws-migratesoa.
3. S.S. Yau and H.G. An, "Confidentiality Protection in Cloud Computing Systems," *Int'l J. Software Informatics*, vol. 4, no. 4, 2010, pp. 351-365.

4. Y. Zhu et al., "Dynamic Audit Services for Outsourced Storage in Clouds," to be published in *IEEE Trans. Services Computing*, 2011.
5. L. Whitney, "Amazon EC2 Cloud Service Hit by Botnet, Outage," *CNET News*, 11 Dec. 2009; http://news.cnet.com/8301-1009_3-10413951-83.html.
6. "Google Cloud Platform Used for Botnet Control," *Info Security*, 10 Nov. 2009; www.infosecurity-us.com/view/5115/google-cloud-platform-used-for-botnet-control.
7. C. Li, A. Raghunathan, and N. Jha, "Secure Virtual Machine Execution under an Untrusted Management OS," *Proc. IEEE 3rd Int'l Conf. Cloud Computing (CLOUD 10)*, IEEE CS Press, 2010, pp. 172-179.
8. Cloud Security Alliance, "Top Threats to Cloud Computing V1.0," Mar. 2010; <https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf>.
9. S.S. Yau et al., "Automated Situation-Aware Service Composition in Service-Oriented Computing," *Int'l J. Web Services Research*, vol. 4, no. 4, 2007, pp. 59-82.
10. S.S. Yau et al., "Rapid Development of Adaptable Situation-Aware Service-Based Systems," *Web Services Research for Emerging Applications: Discoveries and Trends*, L.-J. Zhang, ed., Information Science Reference, IGI Global, 2010, pp. 104-139.
11. S.S. Yau et al., "Toward Development of Adaptive Service-Based Software Systems," *IEEE Trans. Services Computing*, vol. 2, no. 3, 2009, pp. 247-260.
12. B.T. Ward and J.C. Sipior, "The Internet Jurisdiction Risk of Cloud Computing," *Information Systems Management*, vol. 27, no. 4, 2010, pp. 334-339.

Stephen S. Yau is the director of Arizona State University's Information Assurance Center and a professor of computer science. His research interests include software engineering, cybersecurity, distributed computing systems, service-based computing, and cloud computing systems. Yau received a PhD in electrical engineering from the University of Illinois at Urbana-Champaign. He is a Life Fellow of IEEE and a Fellow of the American Association for the Advancement of Science. Contact him at yau@asu.edu or at <http://dpse.asu.edu/yau>.

Ho G. An is a PhD student in the School of Computing, Informatics and Decision System Engineering at Arizona State University. His research interests include services and cloud computing, security, and privacy. An received an MS in computer science from Arizona State University. Contact him at ho.an@asu.edu.

cn Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.

New Software Engineering & Programming Titles from Morgan Kaufmann



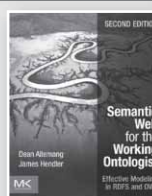
Engineering a Compiler, 2nd Edition

Keith Cooper & Linda Torczon
ISBN: 9780120884780 | \$89.95
A classic introduction to compiler construction fully updated with new techniques and practical insights.



An Introduction to Parallel Programming

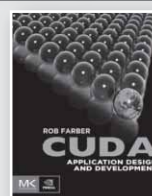
Peter Pacheco
ISBN: 9780123742605 | \$79.95
The first true undergraduate text in parallel programming, covering OpenMP, MPI, and Pthreads.



Semantic Web for the Working Ontologist, 2nd Edition

Effective Modeling in RDFS and OWL

Dean Allemang & James Hendler
ISBN: 9780123859655 | \$54.95
The bestselling practitioner's guide to the semantic web, updated with the latest developments in technologies for building useful and reusable models and applications.



CUDA Application Design and Development

Rob Farber
ISBN: 9780123884268 | \$49.95
A roadmap for developers facing the challenge of developing applications to effectively use GPUs with CUDA to achieve efficiency and performance goals.



Tcl/Tk, 3rd Edition A Developer's Guide

Cliff Flynt
ISBN: 9780123847171 | 69.95
Want to take your programming to the next level? Get *Tcl/Tk: A Developer's Guide, Second Edition*. — Cameron Laird, Vice President of Phaseit, Inc.

COMING SOON!

MK
MORGAN KAUFMANN

mkp.com

Scan this QR code to view all MK's Software Engineering & Programming Titles!



Prices subject to change.

COVER FEATURE



Software Engineering— Missing in Action: A Personal Perspective

David Lorge Parnas, *Middle Road Software*

Although a huge number of articles have been written about software development and many interesting ideas have been proposed, researchers and practitioners have failed to create a new engineering discipline focused on building software-intensive systems.

My professional career began about the time the term “Software Engineering” came into use. Before that, people were using terms like “programming” or “software development” to talk about work that today might be labeled software engineering.

The new terminology was introduced because some people noticed two things:

- Software was, even in the 1960s, beginning to be a major bottleneck. It was common to find the hardware ready and working when the software was still incomplete and unreliable. Software was usually over budget, behind schedule, and not fit for its intended use.
- Many of those who were developing software began to realize that they were doing something quite different from what they had been taught to do. They had been trained to be scientists or mathematicians and to add to our knowledge. Now, they were creating artifacts for others to use.

The founders of the “Software Engineering” movement noted that Engineers had been taught how to do their job; in contrast, people who were developing software had learned to do very different jobs. Engineers, although far from perfect, did not have as many problem projects as software developers had.

Those who founded the movement hoped that many of the software development problems that we were experiencing would go away if software development became a new field of engineering. This was not a jurisdictional dispute; the issues raised were about education and regulation.

WHAT MAKES ENGINEERING WORK?

Engineers, and their work, are not perfect, but they often succeed in building functional, reliable products within a fairly predictable timeframe and close to estimated cost. This, at least, is what the founders of the movement believed when they decided to use older engineering disciplines as a model for the rapidly growing software development profession.

The secret of the success and good reputation that engineers enjoy is simple:

- They have been taught how to do their job.
- They have been taught the basic mathematics (theory) and science they need to perform their work.
- They have been taught to work according to strict rules and to understand that if they do not follow those rules, they might be found to have been negligent and lose their license to work as Engineers.

It is important to note that Engineers are not just taught rote procedures tied to current technology; they are taught the basic mathematics and science that will allow them to understand and use new technology when it becomes available. Their education permits them to understand the assumptions behind standard procedures and, therefore, to know when new developments justify new procedures.

EDUCATION AND LICENSING

There is a “core body of knowledge” associated with each engineering discipline that comprises mathematics, scientific knowledge, guidelines, and regulations that all who practice that discipline are expected to know. In most jurisdictions, this body of knowledge is agreed upon at a state or national level, and academic programs are periodically evaluated to make sure that all elements of the core are taught effectively. Exams after the end of the university program are used to confirm that graduates have learned the core principles and are qualified to work in the profession.

In some jurisdictions, those who did not receive a degree from an accredited engineering program can be licensed if they pass an extensive set of exams that test their comprehension of the core body of knowledge. In most jurisdictions, graduates must work as an apprentice to a licensed engineer for several years before they are finally licensed to work on their own. License holders elect a body that maintains and enforces the regulations. Legislation gives the body the power to license practitioners and to enforce the rules.

This approach, which is also used in other disciplines such as law and medicine, has led to the success that the founders of the software engineering movement envisioned. In most jurisdictions, these professions are self-regulating.


In spite of all the positive things we can say about the engineering profession, it is important not to have an unrealistic view of it. Not all good designers are Engineers or behave like Engineers, and not all Engineers are good designers. The licensing bodies enforce *minimal* standards and discipline practitioners who do not meet those standards.

Professional Engineers have been taught the scientific principles, mathematics, procedures, rules, and regulations appropriate to their discipline. They have been taught to use this information when designing. They also have undergone a period of supervised internship or apprenticeship before they are fully licensed as a professional. However, many things can go wrong:

- Some Engineers never deeply understood what they were taught; they understood it just well enough to pass the exams.

- Some forget or ignore what they did learn.
- Some do not understand the principles behind what they learned and do not apply them correctly in unusual situations.
- Some deliberately take shortcuts or neglect the rules.

Further, engineering science and principles do not tightly constrain designers. Using the established principles of the discipline, Engineers can produce either routine, uncompetitive designs or brilliant designs. Engineering principles allow creating a product that does exactly what is required or one that does more. For example, the product can meet only current needs or it can grow as user needs change.



In spite of all the positive things we can say about software engineering, it is important not to have an unrealistic view of it.

Some brilliant designers never had a formal education in their area of expertise, but they are intuitive and have a good artistic sense. They also are likely to be careful people who pay attention to detail. Often they are not formally recognized as Engineers, and perhaps they do not need to be. However, such brilliant, intuitive designers are rare. In most cases, we cannot rely on them to provide the products we use every day. Some work must be done by less gifted people who benefit from having an engineering education and working in a regulated profession.

A few of the useful products that we depend on today are beautiful, highly functional designs. Far more began as poor designs but were improved through a lengthy period of prerelease testing, followed by beta testing and post-delivery revisions. Almost every piece of software that I encounter contains evidence of oversights either caused or exacerbated by a lack of discipline.¹

Engineer or technologist?

Engineering educators must clearly distinguish between technology, which changes rapidly and has many arbitrary facts, characteristics, and scientific principles, which remain usable throughout the Engineer's career. A good education teaches future Engineers how to use fundamental science to understand new technologies.

Engineer or application specialist?

Engineers are usually educated to work in broad disciplines and are not restricted to narrow fields of practice. Graduates are civil engineers, not road engineers or bridge engineers. Engineers usually become specialists through experience, but their education allows them to change.

COVER FEATURE

Engineer or scientist?

Engineering curricula often share some content with science and mathematics programs, but the educational goals are different. The engineering student must learn how to use science and mathematics to build things, while the science student must learn how to add new knowledge to previously known information.

When a new engineering field is developing, it can sometimes be difficult to distinguish it from an existing science field. For example, when electrical engineering was new, some universities tried to keep it in physics. Physics departments claimed that physicists learned everything anyone needed to know to design electrical systems. Nonetheless, these fields are now clearly distinguished. Some people educated as engineers might end up doing research and working as scientists, but their focus is usually on applicable science. Often, they work together with pure scientists to develop recent scientific advances into useable technologies.

Engineers are usually educated to work in broad disciplines and are not restricted to narrow fields of practice.

THE STRUCTURE OF THE PROFESSIONS

Scientists have attempted to partition the body of knowledge accumulated about the world into distinct areas such as physics, biology, and chemistry; these are further divided into narrower areas such as hydraulics, thermodynamics, human physiology, and organic chemistry. The borders between these areas are not always clear, but the basic distinctions are. This structuring is essential to the work of scientists who often confine their research to a very narrow area. Specialization allows a scientist to know an area well and to use that knowledge to extend our understanding of the area.

Engineering is not partitioned in the same way. Engineers from a specific discipline are expected to be responsible for developing a class of products; to do that, they are required to know material from several areas of science. At the start of a career, it is not easy to predict what knowledge they will require, so an engineering education must be broad; even the required core is broad. There is significant overlap in the requirements for the various disciplines. An engineering discipline is characterized by a selection of topics from science and mathematics, but it does not have an exclusive claim to those topics.

PROGRAMMER VERSUS SOFTWARE ENGINEER

When the term “Software Engineering” was introduced, many asked a simple question, “How is software engineer-

ing different from programming?” Some of those who asked that question were skeptical and wondered if the term had been invented to attract more attention and funding. Others were asking the question rhetorically to suggest that there was no such field.

The best response to this question was provided by British computer scientist Brian Randell, who described software engineering as “the multiperson development of multiversion programs.” This pithy phrase implies everything that educators should be teaching to future software developers. It should go without saying that a software engineer must be able to program, but that is not enough.

However, Randell’s description is not sufficient to determine the core body of knowledge from the point of view of those who license Engineers. Pure software knowledge is not enough for the development of many software products. Just as those who grant licenses require extensive overlap between mechanical and chemical engineering, they would expect a licensed software engineer to know much more than software design.²

WHY BOTH MISSING IN ACTION AND LOST?

The goal of those who introduced the term “Software Engineering” has not been achieved, and in fact we seem to have lost sight of it. The gap between the computer science research world and software development practices continues to grow:

- Industry is aware of the need for improvement and sporadically forms new groups and initiatives that attempt to bring about change re what practitioners do. Most mainstream academics do not get involved. Often, they are too busy playing the publication numbers game.³
- On the academic side, we see new notations, formalisms, proof methods, and design approaches. These gain little traction with industry because they do not appear to address the practitioner’s problems. Rather than show a better, more efficient way to do things, they call for additional work that has no obvious benefit.⁴

The gap between research and practice is not strictly between academia and industry. Larger companies have in-house research groups whose work looks much like academic research—they interact at least as much with external researchers as with internal developers. Other companies have their own internal methods specialists, but the developers often view them as theoreticians. It is this gap between academic research and developer problems that leads me to say that the “Software Engineering” discipline is “missing in action.”

University researchers and educators, even those who claim to be in “Software Engineering,” are rarely involved in establishing a regulated profession. In fact, sometimes they actively resist it.⁵

SOFTWARE ENGINEERING MEETS ...?

Each of the other articles in this special issue focuses on a specific area of research or application area and discusses its relation to software engineering. We must consider an obvious question: “If software development has not become an engineering discipline, how can they talk about it meeting another area?” In fact, none of these articles do that; each one confirms my position that software engineering, as originally envisioned, does not yet exist.

Theory

Manfred Broy’s message in “Can Practitioners Neglect Theory and Theoreticians Neglect Practice?” is that we cannot have an engineering discipline without “theory.” In traditional engineering, theory refers to a set of assumptions about the physics of the situation and a mathematical analysis of the implications of those assumptions. In computer science, there is no physics involved; theory is all mathematics.

Today, more than half a century after the term was coined, an article arguing that software engineering needs mathematics is evidence that we do not yet have such a field. In the traditional engineering disciplines, professors do sometimes discuss the mathematics to be included in a curriculum, but they do not discuss whether they need mathematics—they discuss which mathematics and how much mathematics. In software, it is also necessary to explain how mathematics can be used to improve product quality. In fact, some doubt that it has any relevance at all.⁶

If we want to establish software development as an engineering profession, we definitely need to discuss the role that mathematics can play in the field and exactly what mathematics must be taught. There have been some proposals, but there has not been enough discussion to reach any agreement.^{7,8}


Open source software

Brian Fitzgerald’s “Open Source Software: Lessons from and for Software Engineering” offers an interesting cautionary tale for software developers. Proponents have advanced OSS as a “silver bullet” that can ameliorate many of the problems that software developers encounter. However, OSS is a business model, not a design or engineering method. OSS is a way to motivate and control developers; it offers a different method for recouping investments. Consequently, an OSS product can be reliable or unreliable, changeable or difficult to change, and so on. A software development effort can use OSS ideas or reject

them independently based on whether the work is done by professional engineers or not.

Evolutionary computation

In “Software Engineering Meets Evolutionary Computation,” Mark Harman begins by reminding us that software evolves and refers to early research that investigated how software grew and changed as a result of modifications and additions. The bulk of this article deals with techniques that use the idea of evolution in a different way: the application of algorithms that mimic the genetic mutation process. These are interesting algorithms that can be useful in many situations, including their application to some software project management problems. Such algorithms could be a part of the core of software engineering knowledge, but they are never mentioned in many educational programs. Currently, it is not clear whether this approach would be accepted as



University researchers and educators, even those who claim to be in software engineering, are rarely involved in establishing a regulated profession.

a part of the core knowledge required of software engineers. However, if we had established an engineering discipline, it would be clear.

This particular programming approach does pose one problem for professional engineers, who are responsible for assuring that their product is fit for its intended use. Providing such assurance is difficult, though not impossible, if the product’s performance depends on future evolution.

Space applications

In “Software Engineering for Space Exploration,” Robyn Lutz provides an excellent introduction to the role of software in space exploration and describes the problems that programmers and engineers have had to solve in that application area.

This article illustrates why we need to develop a software engineering discipline in general rather than granting degrees in narrower fields such as space software engineering, aircraft software development, or game design. The problems that Lutz describes arise in many software applications. The software field has developed into a set of cliques, each with its own terminology and technologies. These differences in terminology conceal the common principles and lead to duplication, confusion, and some unnecessary disagreements.

Establishing a core body of knowledge would enable and enhance the transfer of ideas and technology between

COVER FEATURE

various application areas. It would also reduce the unnecessary differences in terminology between suppliers.

Service-oriented software and cloud computing

In “Software Engineering Meets Services and Cloud Computing,” Stephen S. Yau and Ho G. An describe an architecture for an important class of Web-based systems. Although the applications are new and modern high-speed communication networks make the distributed structure practical, the architectural approach is reminiscent of others that work well in other applications. The reader might want to compare the service-oriented approach with the output-oriented approach described in numerous reports.⁹⁻¹¹

Although a huge number of articles have been written about software engineering and many interesting ideas have been proposed, researchers and practitioners have failed to create a new engineering discipline focused on building software-intensive systems.

While each of the other articles in this special issue claims to discuss the intersection of software engineering with some other research area, they support my position:

- If we are still writing papers arguing that we need some theory or mathematics to be a profession, rather than arguing about specific areas of theory that a software engineer should know, we have not established a profession.
- If we find that individual application areas are discovering common problems and solving them with their own terminology and specialized techniques, we have not yet established a profession.
- If we do not consistently distinguish between engineering problems, management problems, technology problems, and business-plan issues, we have not yet established a profession.
- If we are, as many others have noted, a field that is dominated by fads with clever acronyms that cause a flurry of interest and then fade away, we have not yet established a profession.
- If we continue to treat software engineering as a “grab bag” research area rather than as a regulated profession, we have lost sight of the original goal.

Those who began the software engineering movement were prescient. They seem to have anticipated today’s heavy dependence on software; they must have recognized that software engineering should become one of the many established engineering disciplines. Unfortunately, we who came after them underestimated the difficulty of achieving that goal and have lost sight of it.

If we want to establish a discipline of engineering that specializes in software-intensive systems, the first step is

to agree on a core body of knowledge. Thus far, the various efforts to establish a body of knowledge have been too inclusive. They have tried to collect every belief and fact about software development rather than identify a small core of solid knowledge that all software engineers must master.

In my experience, establishing a core will not be easy. Unless we are vigilant, the core will continue to expand and, consequently, lose relevance because it is too large. **■**

Acknowledgments

I thank Carl Chang and David Weiss for their thought-provoking comments on an earlier version of this paper.

References

1. D.L. Parnas, “Risks of Undisciplined Development,” *Comm. ACM*, Oct. 2010, pp. 25-27.
2. D.L. Parnas, “Software Engineering Programmes Are Not Computer Science Programmes,” *Ann. Software Eng.*, vol. 6, 1998, pp. 19-37.
3. D.L. Parnas, “Stop the Numbers Game,” *Comm. ACM*, Nov. 2007, pp. 19-21.
4. D.L. Parnas, “Really Rethinking ‘Formal Methods,’” *Computer*, Jan. 2010, pp. 28-34.
5. D.L. Parnas, “Licensing Software Engineers in Canada,” *Comm. ACM*, Nov. 2002, pp. 96-98.
6. D.L. Parnas, “How Engineering Mathematics Can Improve Software,” *Proc. Int’l Conf. Eng. Reconfigurable Systems and Algorithms* (ERSA 2011); <http://ersaconf.org/ersa11/#hdisplay>.
7. D.L. Parnas and M. Soltys, *Basic Science for Software Developers*, SQRL report no. 7, Software Quality Research Laboratory, Dept. of Computing and Software, McMaster University, 2002; www.cas.mcmaster.ca/sqrl/sqrl_reports.html.
8. D.L. Parnas, “Mathematics of Computation for (Software and Other) Engineers,” *Bull. European Assoc. Theoretical Computer Science*, Oct. 1993, pp. 249-259.
9. K.L. Heninger, “Specifying Software Requirements for Complex Systems: New Techniques and Their Application,” *IEEE Trans. Software Eng.*, Jan. 1980, pp. 2-13.
10. D.L. Parnas, “Precise Documentation: The Key to Better Software,” *The Future of Software Engineering*, S. Nanz, ed., Springer, 2010, pp. 125-148.
11. Z. Liu, D.L. Parnas, and B. Trancón y Widemann, “Documenting and Verifying Systems Assembled from Components,” *Frontiers in Computing Science in China*, June 2010, pp. 151-161.

David Lorge Parnas is professor emeritus at McMaster University, Canada, and the University of Limerick, Ireland, as well as president of Middle Road Software. Parnas received a PhD in electrical engineering from Carnegie Mellon University and honorary degrees from ETH Zurich, the University of Louvain, the University of Italian Switzerland, and the Technical University of Vienna. Contact him at parnas@mcmaster.ca.



Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.

IEEE computer society

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.

MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

OMBUDSMAN: To check membership status or report a change of address, call the IEEE Member Services toll-free number, +1 800 678 4333 (US) or +1 732 981 0060 (international). Direct all other Computer Society-related questions—magazine delivery or unresolved complaints—to help@computer.org.

CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

AVAILABLE INFORMATION: To obtain more information on any of the following, contact Customer Service at +1 714 821 8380 or +1 800 272 6657:

- Membership applications
- Publications catalog
- Draft standards and order forms
- Technical committee list
- Technical committee application
- Chapter start-up procedures
- Student scholarship information
- Volunteer leaders/staff directory
- IEEE senior member grade application (requires 10 years practice and significant performance in five of those 10)

PUBLICATIONS AND ACTIVITIES

Computer: The flagship publication of the IEEE Computer Society, *Computer*, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.

Periodicals: The society publishes 13 magazines, 18 transactions, and one letters. Refer to membership application or request information as noted above.

Conference Proceedings & Books: Conference Publishing Services publishes more than 175 titles every year. CS Press publishes books in partnership with John Wiley & Sons.

Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.

Technical Committees: TCs provide professional interaction in more than 45 technical areas and directly influence computer engineering conferences and publications.

Conferences/Education: The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.

Certifications: The society offers two software developer credentials. For more information, visit www.computer.org/certification.

NEXT BOARD MEETING

13–14 Nov., New Brunswick, NJ, USA

EXECUTIVE COMMITTEE

President: Sorel Reisman*

President-Elect: John W. Walz*

Past President: James D. Isaak*

VP, Standards Activities: Roger U. Fujii†

Secretary: Jon Rokne (2nd VP)*

VP, Educational Activities: Elizabeth L. Burd*

VP, Member & Geographic Activities: Rangachar Kasturi†

VP, Publications: David Alan Grier (1st VP)*

VP, Professional Activities: Paul K. Joannou*

VP, Technical & Conference Activities: Paul R. Croll†

Treasurer: James W. Moore, CSDP*

2011–2012 IEEE Division VIII Director: Susan K. (Kathy) Land, CSDP†

2010–2011 IEEE Division V Director: Michael R. Williams†

2011 IEEE Division Director V Director-Elect: James W. Moore, CSDP*

*voting member of the Board of Governors †nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 2011: Elisa Bertino, Jose Castillo-Velázquez, George V. Cybenko, Ann DeMarle, David S. Ebert, Hironori Kasahara, Steven L. Tanimoto

Term Expiring 2012: Elizabeth L. Burd, Thomas M. Conte, Frank E. Ferrante, Jean-Luc Gaudiot, Paul K. Joannou, Luis Kun, James W. Moore

Term Expiring 2013: Pierre Bourque, Dennis J. Frailey, Atsuhiko Goto, André Ivanov, Dejan S. Milojicic, Jane Chu Prey, Charlene (Chuck) Walrad

EXECUTIVE STAFF

Executive Director: Angela R. Burgess

Associate Executive Director; Director, Governance: Anne Marie Kelly

Director, Finance & Accounting: John Miller

Director, Information Technology & Services: Ray Kahn

Director, Membership Development: Violet S. Doan

Director, Products & Services: Evan Butterfield

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928
Phone: +1 202 371 0101 • Fax: +1 202 728 9614

Email: hq.ofc@computer.org

Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314
Phone: +1 714 821 8380

Email: help@computer.org

MEMBERSHIP & PUBLICATION ORDERS

Phone: +1 800 272 6657 • **Fax:** +1 714 821 4641 • **Email:** help@computer.org

Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan

Phone: +81 3 3408 3118 • **Fax:** +81 3 3408 3553

Email: tokyo.ofc@computer.org

IEEE OFFICERS

President: Moshe Kam

President-Elect: Gordon W. Day

Past President: Pedro A. Ray

Secretary: Roger D. Pollard

Treasurer: Harold L. Flescher

President, Standards Association Board of Governors: Steven M. Mills

VP, Educational Activities: Tariq S. Durrani

VP, Membership & Geographic Activities: Howard E. Michel

VP, Publication Services & Products: David A. Hodges

VP, Technical Activities: Donna L. Hudson

IEEE Division V Director: Michael R. Williams

IEEE Division VIII Director: Susan K. (Kathy) Land, CSDP

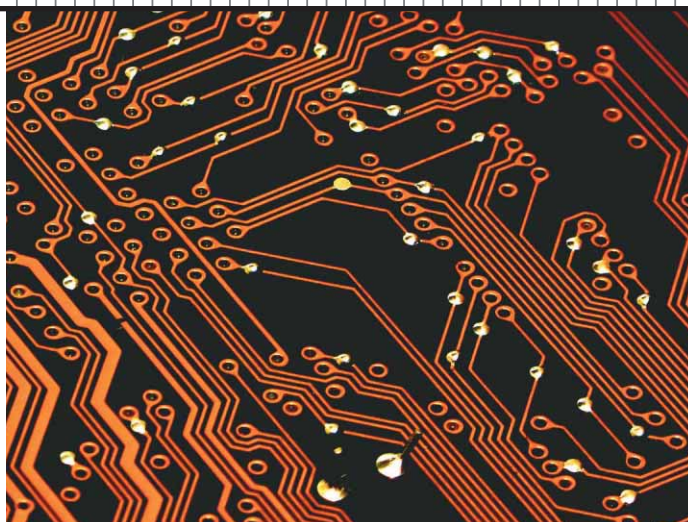
President, IEEE-USA: Ronald G. Jensen



revised 24 August 2011

RESEARCH FEATURE

Display Power Management That Detects User Intent



Jae Min Kim, Minyong Kim, Joonho Kong, Hyung Beom Jang, and Sung Woo Chung
Korea University

In a proposed display power-management scheme, the laptop detects when the user is not looking at the screen, boosting low-power operation to 50 percent and increasing energy savings by up to 13 percent.

With the growth of ubiquitous computing has come an increased reliance on laptops over desktops, and display power management (DPM) that prolongs battery life continues to be a critical issue in maintaining that trend.

Although display power consumption varies across laptops, it tends to hover around 30 percent of the device's total power consumption.¹ This percentage is high in deference to users' demands for high-quality images, videos, and so on, but too often this high power consumption continues even when the user is not actively looking at the screen. Contrary to popular belief, screensavers do little to reduce display power consumption, and the computational overhead to run a saver can even increase system power use.

Nothing saves power as much as simply turning off the display. Thus, an obvious area for DPM improvement is the creation of a mechanism that can detect user intent, turning the display off not only when the user is gone but also when the display is not in use.

As the "Display Power Management and User Presence" sidebar describes, several studies have looked at this problem. A typical operating system provides timeout-based DPM, which turns off the display after a sustained lack of input. Some recent studies have proposed DPM with a finer-grained detection, such as searching for pixels that match the user's skin color² or using ultrasound to detect presence. Most of these

schemes, however, look primarily at power reduction when the user is away from the display.

In contrast, we have developed a scheme that refines detection by identifying states of user presence from retrieved webcam images, such as detecting if the user is attentive or inattentive. To guard against false detections, we include an intermediate state, *weakly attentive*, that allows the face-detection algorithm³ to retrieve another webcam picture a few seconds later and recheck for a frontal face image.

In an experiment to investigate our DPM scheme's power efficiency and the degree of user irritation in employing it, we found that our scheme activates the laptop's low-power mode nearly 50 percent of the time, versus 30 to 40 percent with two other DPMS: FaceOff, a DPM approach that looks at skin-colored pixels to judge user presence, and a timeout-based DPM technique that monitors user input to judge user presence. Our scheme typically reduces average system-wide energy consumption by 5 percent, a reduction that is up to 13 percent more than timeout-based DPM approaches, with our scheme irritating users less than once an hour on average.

DETECTING USER STATE

Our scheme recognizes four states that correspond to the user's physical state, the last three of which are finer-grained states of user presence:

DISPLAY POWER MANAGEMENT AND USER PRESENCE

Among existing DPM approaches, the most popular appears to be timeout-based DPM, which monitors user input to determine user intent. When there is no user input for a fixed time (users typically set the interval to 5 minutes), timeout-based DPM determines that the user is away and turns off the display. As soon as user input occurs, it turns the display back on.

Other DPM approaches, such as FaceOff,¹ detect user presence by processing images from a webcam.² If image processing determines that no user is at the computer, FaceOff turns off the display. The scheme recognizes three user states:

- *Interactive*. The user is interacting with the computer in some way. The webcam does not take pictures.
- *Present*. The user is in front of the computer.
- *Away*. The user is out of the webcam's sensing range.

Another proposed DPM technique uses ultrasonic sonar waves to determine the user's attention level.³ A laptop running this scheme emits inaudible ultrasonic sonar waves and records the waves reflected from a person or object in front of it. After 10 seconds of analyzing the recorded data, the scheme can determine user presence. The scheme is very accurate, and its developers maintain that it can also differentiate select user positions and

actions, although they did not evaluate detection at this level. In other work, they compared the energy efficiency of sonar-based and timeout-based DPM and the schemes' relative degree of user irritation.⁴

The sonar-based scheme and FaceOff have basically the same policy, which turns the display off when the user is not present. They differ only in the detection method used.

Unlike timeout-based DPM, FaceOff, or sonar-based DPM, our DPM scheme turns the display off when the user is not looking at it. This represents a more precise detection level than DPM approaches that turn the display off only if the user is absent.

References

1. A.B. Dalton and C.S. Ellis, "Sensing User Intention and Context for Energy Management," *Proc. Conf. Hot Topics in Operating Systems (HotOS 03)*, Usenix, 2003, pp. 151-156.
2. R.W. Picard, *Affective Computing*, MIT Press, 1997.
3. S.P. Tarzia et al., "Sonar-Based Measurement of User Presence and Attention," *Proc. Int'l Conf. Ubiquitous Computing (UbiComp 09)*, ACM Press, 2009, pp. 89-92.
4. S.P. Tarzia et al., "Display Power Management Policies in Practice," *Proc. Int'l Conf. Autonomic Computing (ICAC 10)*, IEEE CS Press, 2010, pp. 51-60.

- *Away*. The user is away from the laptop.
- *Interactive*. The user is interacting with the display in some way, such as typing text or clicking on an icon.
- *Attentive*. The user is looking at the display but not interacting with it.
- *Inattentive*. The user is not looking at the display.

A webcam detects the attentive or inattentive states by checking for the presence or absence of a frontal face image. It detects the away state by checking for a certain percentage of skin-colored pixels.

Our scheme also recognizes the *weakly attentive* state, which is a bridge between the attentive and inattentive states. It has no mapping to a physical user state, but rather serves to minimize detection errors when a user might glance away from the display for only a second, perhaps to rest his eyes. In the weakly attentive state, the face-detection algorithm checks another webcam picture (taken three seconds later) for a frontal face image to be sure that the user is indeed inattentive. Obviously, a user could easily become annoyed if the display switches to low-power mode when he has just looked away for a second. The weakly attentive state guards against such irritation.

Figure 1 shows how user states transition. Applying these state transitions to physical user states, the user starts in the away state (bottom left of the figure and moving clockwise). When the user returns to the laptop and begins entering data, he transitions from the away to the interactive state. He might then review that data but not input anything additional, transitioning to the atten-

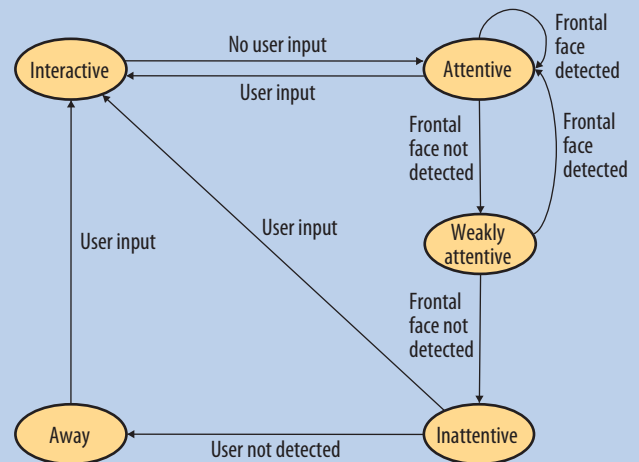


Figure 1. State transition diagram for the proposed DPM scheme. Keyboard or mouse activity indicates the interactive state, while a webcam detects if the user is looking at the display, not looking at the display, or not present. The weakly attentive state has no corresponding physical user state, but rather is an opportunity for the face-detection algorithm to recheck the presence or absence of a frontal face image.

tive state. After the review, he might look away to take a phone call or think about his work, causing a transition to the inattentive state. Finally, he might leave the laptop to consult with a colleague, causing a transition from the inattentive to the away state.

The user can also go directly from an inattentive to interactive state when he turns around to look at the display and open a file, for example, or he could move from the

RESEARCH FEATURE

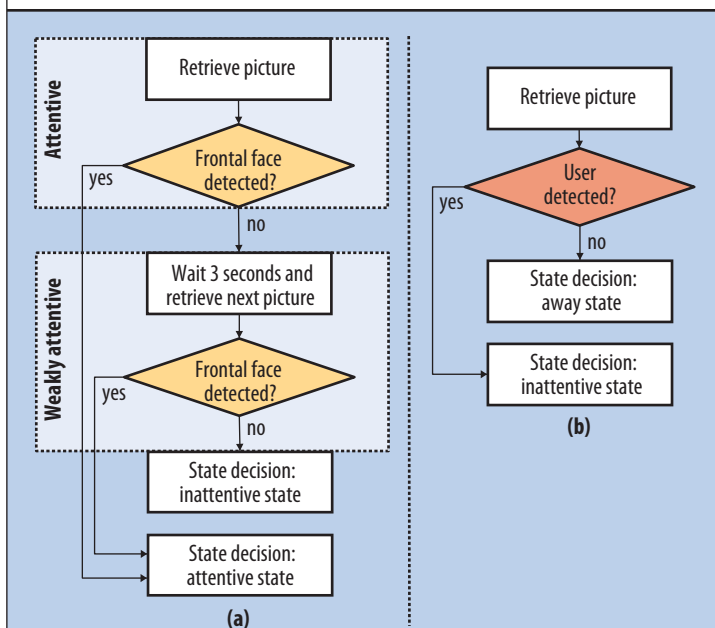


Figure 2. Flowchart for detecting the finer-grained states of user presence. (a) To determine if the user is attentive or inattentive, the scheme uses a robust real-time face-detection algorithm (yellow diamond). (b) To determine if the user is inattentive or away from the laptop, the scheme uses skin-color detection (red diamond).

attentive to interactive state when he is ready to enter more text after reading what he just wrote.

Any interaction with the display makes it obvious that the user is in the interactive state. Judging the other states is more complex. Figure 2 shows how our scheme concludes that a user is in the attentive, inattentive, or away state. To decide if the user is attentive or inattentive, as Figure 2a shows, the detection scheme first retrieves a picture and then uses robust real-time face detection to decide if the picture is a frontal face image. The scheme retrieves the picture and runs the frontal-face-detection algorithm at a predefined time, t_{interval} , where t is the abbreviation of time.

If the algorithm detects a frontal face image, the scheme concludes that the user is in the attentive state. If the algorithm fails to detect a frontal face image, the scheme retrieves another picture and reruns the algorithm. At this point, the user is “in” the weakly attentive state, although this does not reflect a physical state because it is impossible to half look away from a display and still be attentive. This is why we consider weakly attentive to be an intermediate state, not a state of user presence per se.

If the face-detection algorithm still indicates no frontal image, the scheme concludes that the user is in the inattentive state. Likewise, if the algorithm detects a frontal image, the scheme concludes that the user is in the attentive state.

As Figure 2b shows, the scheme uses a skin-color detection algorithm to decide if the user is inattentive or away from the laptop.² If a certain number of skin-colored pixels

appear in the retrieved picture, the scheme rules that the user is present but inattentive. Similarly, if the skin-colored pixels dip below a certain threshold, the scheme concludes that the user is not present.

IMPLEMENTATION ELEMENTS

To implement our DPM scheme, we use a system that consists of one main coordinating function and four subfunctions: message hooking control, camera control, image processing, and power management.

The message hooking control function continuously checks to determine if the operating system has sent a user input message and sends the results to the main function. When the main function receives a “no message” status, it sends a request to the camera control function to retrieve a webcam image, which then goes to the image-processing function. The image-processing function then runs either the frontal face or skin-color detection algorithm, depending on the last user state.

When the last user state was attentive, the image-processing function runs the face-detection algorithm.³ Because this algorithm uses Haar basis functions with the integral function, it can rapidly compute Haar-like features that are in proportion to image size. To maintain an acceptable processing overhead, we fix image size as 160×120 pixels. The image-processing function checks the facial image size, and when it finds a face large enough to fit a full front face, it determines that the user is looking at the display (attentive state). For our evaluation, we defined the size threshold for a frontal face as more than 50×50 pixels, but the optimal value depends on the user’s facial dimensions and distance from the laptop.

When the last user state was inattentive, the image-processing function runs the skin-color detection algorithm, which examines each pixel to determine if it is skin-colored.² If the ratio of skin-colored pixels equals or exceeds a certain threshold (we used 20 percent, but other values might be optimal, depending on the lighting conditions, skin-color tone, and so on), the image-processing function determines that a user is present (inattentive state).

Once the algorithm identifies the user’s state, the image-processing function sends the results to the main function, which then calls the power management function to change the power mode according to the predefined policy. Whenever the main function receives a “message sent” status from the message hooking control function, it calls on the power management function to change the power mode to normal.

EXPERIMENTAL RESULTS

We evaluated three DPM schemes on a Samsung NT-R20 laptop with a 14.1-inch diagonal, wide-aspect-ratio LCD

LOGGING DATA

To ensure a fair comparison, our log-collection program collects log data for the three DPM approaches—timeout-based DPM, FaceOff, and our DPM approaches—simultaneously. Each scheme determines its own user state by monitoring user input or executing image processing. Whenever any scheme determines a state transition, the log-collection program records the elapsed time and user state.

Each scheme turns off the display when it determines that the laptop should run in low-power mode. Any user input that one scheme detects can interfere with the others, since they are running simultaneously. For example, if only one scheme is running in the low-power mode, the schemes running in the normal mode should ignore the first user input because the input will turn the laptop back to the normal power mode.

For the schemes already in normal power mode, the time from the last input should keep increasing without being reset to zero.

In the user input scenario in Figure A, the first user input (A) occurs when all three schemes are running in the normal power mode. Thus, all the schemes reset their time from input A to zero. After two minutes, our scheme detects an inattentive user state and turns the display off. However, timeout-based DPM and FaceOff stay in the normal power mode, because the user input does not meet their conditions for going to low-power mode. For timeout-based DPM, only 2 minutes (not the required 5 minutes) have passed since input A. For FaceOff (which turns the display off only if the user is absent), conditions for turning off the display are not met because the user is present.

After another minute (3:00), user input B occurs. Timeout-based DPM and FaceOff ignore this input because the user would not

have made it if the monitor was already on. Thus, time from the last input remains at 3 minutes for the timeout-based DPM and FaceOff, but resets to zero for our scheme.

After another 2 minutes without any user input (5:00), timeout-based DPM turns the display off, since 5 minutes have passed from input A. Input C occurs 3 sec later, when the user moves to turn the display back on. In essence, timeout-based DPM has caused an irritation event. FaceOff and our scheme ignore input C.

We analyzed log data in this way to determine energy consumption as well irritation events for each scheme.

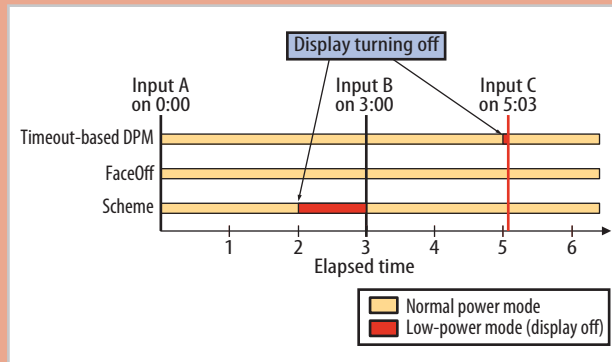


Figure A. Possible data collection scenario. A user is reading a document for most of the 6 minutes being logged, making only three inputs (A, B, and C). Because the schemes have different detection thresholds, analyzing user input is important.

display and a webcam that consumes an average of 1.5 W. The laptop is configured with the Intel Core 2 dual processor running at 2.0 GHz with the Windows XP operating system (home edition).

Our experiment compared timeout-based DPM, FaceOff, and our scheme. We first analyzed our scheme with various t_{interval} values in terms of resulting performance overhead to determine an acceptable t_{interval} . We then conducted a user study to evaluate energy consumption and irritation level while using each scheme. Our study involved 20 volunteers, mostly students and office staff, each of whom regularly uses a laptop.

We loaded all three DPM schemes on each volunteer's laptop along with a log-collection program that would record the laptop's energy consumption for each scheme without user intervention. We asked volunteers to run the schemes and the log-collection program simultaneously for 5 to 6 hours for one day. The "Logging Data" sidebar explains how we logged and interpreted data on the three schemes running simultaneously.

Performance overhead

We hypothesized that our scheme would reduce power consumption while minimizing the slowdown of any user programs. In the attentive and inattentive states, the image-

processing function executes every t_{interval} . If that value is too small, the execution time of other applications increases because of image-processing overhead. If the value is too large, the scheme might lag in transitioning to low-power mode.

To arrive at the optimal t_{interval} value, we ran compression for 280 Mbytes of data at the same time that we ran our scheme with varying t_{interval} . Table 1 shows data compression time with different t_{interval} values. Without our scheme ($t_{\text{interval}} = \text{infinity}$), data compression took 120 seconds.

With a 1-sec t_{interval} value, data compression took 151 sec—representing an unacceptable performance overhead of 31 sec, or approximately 25.8 percent. However, with a 15-sec value, overhead shrank to 6 sec, or 5 percent, and values greater than 15 sec showed insignificant

Table 1. Data compression time with different t_{interval} values.

T_{interval} (sec)	Compression time (sec)
1	151
15	126
30	125
60	125
∞	120

RESEARCH FEATURE

Table 2. Average power consumption by user state for various DPM schemes.

DPM scheme	Display state	Power consumption (W)	User state
Timeout-based DPM	On	25.8	Interactive (predefined period not met since last user input)
	Off	18.9	Away
FaceOff ($t_{\text{interval}} = 15 \text{ sec}$)	On	26.1	Interactive or present
	Off	18.9	Away
Our scheme ($t_{\text{interval}} = 15 \text{ sec}$)	On	26.1	Interactive or attentive
	Off	19.2	Inattentive
	Off	18.9	Away

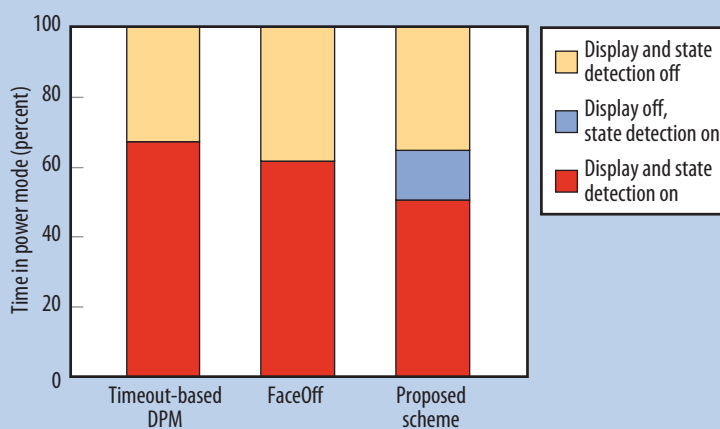


Figure 3. Time spent in the normal power mode by user state. The red area represents percentage of time in normal mode when the laptop is on. The graph shows that the display is off for a longer period with the proposed scheme. The blue area represents low-power mode when the user is in the inattentive state.

performance overhead. Consequently, we settled on a 15-sec t_{interval} for our experiment.

Of course, the optimal t_{interval} value might be different for other scenarios, since it depends on the laptop and the applications running on it. Also, some users might experience a much smaller overhead, since image processing does not occur in the interactive state, thus eliminating a primary performance overhead source.

Low-power mode

We also measured the average power consumption when the laptop is in a low-power mode. Measurements reflect average total laptop (system-wide) power for 10 minutes, according to the X4-Life Inspector II (www.x4-life.de/produkte/haus-garten/inspector-ii).

For this part of the experiment, we assumed that our DPM scheme would turn the display on when the user is in the interactive or attentive state and turn it off when the user is in the inattentive or away state. In the experimental context, we equate the weakly attentive state with the attentive state.

We could save even more energy by controlling the power of other components, such as the disk drive, but to ensure a fair comparison, our scheme considers only the display in on and off states. Both timeout-based DPM and FaceOff turn the display on when the user is interactive or present and turn it off when the user is away.

Table 2 shows the average power consumption for power modes that correspond to their user states. With timeout-based DPM, the laptop consumes 25.8 W; with our scheme, it consumes 26.1 W. The 0.3 W difference is from using the webcam and image processing to distinguish between the attentive and inattentive states.

Energy savings with users

To estimate how much energy the three DPM schemes consume in various user states, we used a simple model to analyze log data. The consumption from applications running on the laptop varies little with the DPM scheme, so we did not consider that source.

We calculated the total energy consumed for a certain period at a given state as the product of time and the average power consumption in a particular mode (normal or low power). Total energy consumption, E , is thus

$$\sum_{i=1}^n P_i \times T_i,$$

where P_i is the average power consumption in power mode i , T_i is the total time spent in the power mode i , and n is the number of power modes. After the model calculates each state's energy consumption, it sums the calculated values to get total energy consumption.

For example, if the user state is interactive and attentive for the first 10 minutes, inattentive for the next 10 minutes, and away for the last 10 minutes, the total energy consumption is then

$$26.1 \text{ W} \times 600 \text{ sec} + 19.2 \text{ W} \times 600 \text{ sec} + 18.9 \text{ W} \times 600 \text{ sec},$$

which totals 38,520 J (values from Table 2).

In our evaluation, timeout-based DPM changes the display to low-power mode when there is no user input for 5 minutes (typical user setting).¹ We use a 15-sec t_{interval} for FaceOff and our proposed scheme.

Figure 3 shows the average time spent in each power mode across three schemes for 20 users. The red region of each bar represents the time in the normal power mode

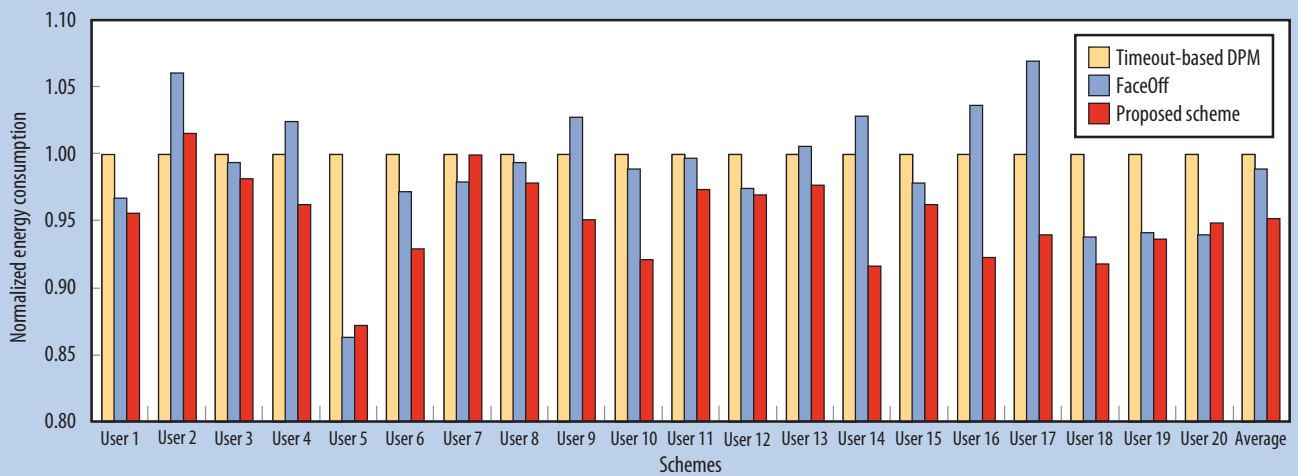


Figure 4. Comparison of total (systemwide) energy consumption according to scheme for the 20 users and on average (last three bars). The proposed scheme tended to save more energy relative to timeout-based DPM and FaceOff. Certain use patterns, such as frequently looking away from the monitor, resulted in higher energy savings with the proposed scheme. All energy consumption is normalized to that of timeout-based DPM.

(display on). The red and blue regions represent time in low-power mode (display off). The green region shows only our scheme because it is the only one that looks at attentiveness.

As the figure shows, timeout-based DPM runs in low-power mode for 32.7 percent of the laptop's operation—the least time of all the schemes—since it takes a full five minutes to move to low-power mode. Because our scheme moves the laptop to low-power mode when it detects an inattentive user, it runs in low-power mode for 49.3 percent of the laptop's operation. Laptop users running our scheme tend to be in low-power mode more than 10 percent longer, even relative to FaceOff's duration of 38.3 percent. The difference is primarily because our scheme can detect the inattentive user state, but FaceOff cannot.

Figure 4 shows the total energy consumption for all three DPM schemes, according to log results. For each user, we normalized all results to the energy consumption of timeout-based DPM. As this figure shows, our scheme saves as much as 13 percent more energy (user 5), with an average savings of 5 percent, relative to timeout-based DPM. Relative to FaceOff, our scheme again saves up to 13 percent more energy (user 17), with an average savings of 4 percent.

Energy savings tend to vary depending on use pattern. When a user frequently looks away from the display, for example, our scheme has more potential to save energy relative to FaceOff, which looks only at interactive, present, and away states.

Measuring user irritation

Automated power management can irritate users, particularly if the display goes off when the user needs it to be on. In our user study, we looked at irritation level in terms

of *irritation events*. An irritation event occurs when the user must interact with the laptop to turn on the display within three seconds after DPM has turned it off.

Figure 5 shows the irritation events per hour for each scheme. Although timeout-based DPM and FaceOff have a lower average of irritation events per hour (0.1 and 0.5 event, respectively), our scheme still has a reasonable average (approximately 0.8 event).

In three cases (users 13, 14, and 17), our scheme gave rise to more than one irritation event per hour, which we believe is due to the webcam angle's influence on the accuracy of the face-detection algorithm. When the webcam is not properly positioned, the algorithm can fail to detect a frontal face (attentive state) even when the user is facing the display.

The skin-detection algorithm, on the other hand, is less affected by webcam angle, and since FaceOff turns the display off according to the results of this algorithm, user irritation is lower in some cases. In other cases, however, FaceOff causes *more* irritation events than our scheme because lighting conditions and skin color influence the accuracy of the skin-color detection algorithm more than they do the accuracy of the face-detection algorithm that we use to determine attentiveness.

Adjusting the webcam angle dynamically with special hardware and software should substantially reduce user irritation while running our scheme.

Our user-aware DPM scheme can save power and prolong laptop battery life by determining if a user is looking at or away from the display. Because most

RESEARCH FEATURE

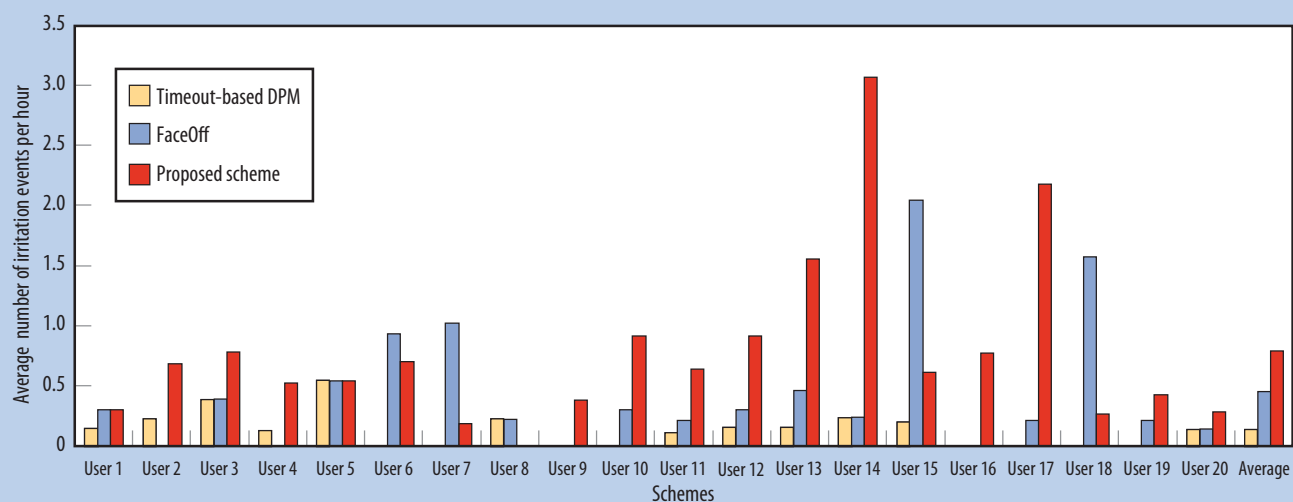


Figure 5. Comparison of user irritation events per hour, according to DPM scheme for the 20 users and on average. The proposed scheme had more irritation events than timeout-based DPM and FaceOff, but average irritation is still less than once an hour.

laptops now come with webcams, implementing our scheme requires no additional hardware.

Our scheme reduces the total energy consumption by up to 13 percent, relative to timeout-based DPM and FaceOff, but there is room for improvement if we adopt a more accurate and power-efficient frontal-face-detection algorithm. With the current algorithm, environmental objects or passersby can interfere with detection. Consequently, our scheme might not conclude that the user is in fact inattentive or away from the display. A more accurate detection algorithm could avoid such a situation, saving even more energy.

We are also researching how to lower video quality when the user is not attentive, which can reduce CPU or GPU power consumption. Our current efforts focus on tablets and smartphones. **C**

Acknowledgments

We thank Gokhan Memik and Stephen P. Tarzia for helpful comments on the foundational ideas for this article. We also thank the anonymous reviewers for their useful feedback.

This research was supported by the Basic Science Research Program of the National Research Foundation of Korea, funded by the Ministry of Education, Science and Technology under contract 2011-0004917, and the Ministry of Knowledge Economy, Korea, Information Technology Research Center support program, supervised by the National IT Industry Promotion Agency under contract NIPA-2011-C1090-1121-0010.

References

1. S.P. Tarzia et al., "Display Power Management Policies in Practice," *Proc. Int'l Conf. Autonomic Computing (ICAC 10)*, IEEE CS Press, 2010, pp. 51-60.
2. M.J. Jones and J.M. Rehg, "Statistical Color Models with Application to Skin Detection," *J. Computer Vision*, Jan. 2002, pp. 81-96.

3. P.A. Viola and M.J. Jones, "Robust Real-Time Face Detection," *J. Computer Vision*, May 2004, pp. 137-154.

Jae Min Kim is a PhD student in the Division of Computer and Communication Engineering, Korea University. His research interests include user-aware design, computer architectures, and low-power design. Kim received a BSc in computer science from Korea University. He is a student member of IEEE. Contact him at joist@korea.ac.kr.

Minyong Kim is a master's student in the Division of Computer and Communication Engineering, Korea University. His research interests include system-on-chip design, user-aware design, and design verification methods. Kim received a BSc in computer science from Korea University. He is a student member of IEEE. Contact him at mkim05@korea.ac.kr.

Joonho Kong is a postdoctoral researcher in the Division of Computer and Communication Engineering, Korea University. His research interests include fault tolerance, computer architectures, and temperature-aware design. Kong received a PhD in computer science from Korea University. He is a member of IEEE. Contact him at luisfigo77@korea.ac.kr.

Hyung Beom Jang is a PhD student in the Division of Computer and Communication Engineering, Korea University. His research interests include 3D stacking, computer architectures, and low-power design. Jang received an MSc in computer science from Korea University. He is a student member of IEEE. Contact him at kuphy01@korea.ac.kr.

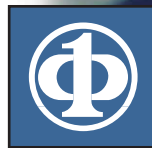
Sung Woo Chung is an associate professor in the Division of Computer and Communication Engineering, Korea University. His research interests include low-power design, temperature-aware design, and user-aware design. Chung received a PhD in electrical engineering and computer science from Seoul National University. He is a member of IEEE and the IEEE Computer Society. Contact him at swchung@korea.ac.kr.



Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.

COMPUTER SOCIETY CONNECTION

UTA's Yale Patt Wins First IEEE CS Rau Award



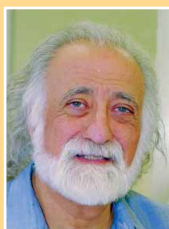
Yale N. Patt, an electrical and computer engineering professor at the University of Texas at Austin, has been selected as the recipient of the inaugural IEEE Computer Society B. Ramakrishna Rau Award. The IEEE Computer Society established the award in 2010 in memory of the late Bob Rau, a senior research scientist at HP Labs. The Rau Award recognizes significant accomplishments in the field of microarchitecture and compiler code generation.

For more than four decades, Patt has combined an active research program with consulting and teaching. His award citation reads, "For significant contributions and inspiring leadership in the microarchitecture community with respect to teaching, mentoring, research, and service."

The award, which comes with a \$2,000 honorarium and a certificate, will be presented at the ACM/IEEE International Symposium on Microarchitecture (MICRO), set for 3-7 December in Porto Alegre, Brazil.

A PIONEER IN MICROARCHITECTURE

In 1965, Patt introduced the WOS module, the first complex logic gate implemented on a single piece of silicon. In 1984, he and students Wen-mei Hwu, Steve Melvin, and Mike Shebanow introduced HPS, a high-performance microarchitecture that exploits instruction-level parallelism. Five years after



Yale Patt leads key research projects in microarchitecture.

that, Patt and student Tse-Yu Yeh introduced the two-level branch predictor, which provides much improved accuracy.

Patt's current research focuses on the potential challenges of 2018-era microprocessors, which are slated to contain more than 30 billion transistors. This includes breaking the abstraction layers that separate the problem statement in natural languages (like English) from the electronic circuits that actually execute the program. Among Patt's projects is ACMP, a reconfigurable heterogeneous multicore microprocessor. He is also working on improving the interface between the processor core and the DRAMs, creating GPUs for non-graphics processing, establishing effective prefetching in a multicore environment, and making more effective use of the runtime system for performance.

OTHER ACHIEVEMENTS

Patt holds the Ernest Cockrell Jr. Centennial Chair in Engineering at the University of Texas at Austin. He earned a BS from Northeastern

University and an MS and PhD from Stanford University, all in electrical engineering.

With former student Sanjay Patel, Patt coauthored *Introduction to Computing Systems: From Bits and Gates to C and Beyond*, a textbook now in its second edition.

Patt was the recipient of the 1995 IEEE Emanuel R. Piore Medal "for contributions to computer architecture leading to commercially viable high-performance microprocessors," the 1996 IEEE Computer Society/ACM Eckert-Mauchly Award "for important contributions to instruction-level parallelism and superscalar processor design," the 1999 IEEE Computer Society W. Wallace McDowell Award for engineering and education contributions to the high-performance microprocessor industry, and the 2005 IEEE Charles Babbage Award. Patt is a Fellow of both IEEE and the ACM.

"For Yale to receive the first Rau Award is an honor not only for Yale, but also for our electrical and computer engineering department here at the University of Texas at Austin," said Ahmed Tewfik, department chair. "Yale is one of the top college educators in our nation in general, and computer engineering educators more specifically. He regularly teaches other university professors across the world how to present computer architecture to undergraduate and graduate students. Many of his innovations are embedded in the microprocessors that we all use in our laptops, tablets, and

COMPUTER SOCIETY CONNECTION

smartphones. This award recognizes his many contributions over the years to research and education in computer engineering.”


B. RAMAKRISHNA ‘BOB’ RAU

B. Ramakrishna Rau was among the computer architecture professionals whom Patt respected most. “Bob was one of the giants in our community who left us long before his time. Frankly, I did not expect I would be the first to get this award since I can think of at least two other people who

are very deserving of it, including one who is my student. I can only say it is humbling to receive this award that bears his name,” Patt said.

Rau, who died in 2002, managed HP Labs’ Compiler and Architecture Research group. He started HP Labs’ research program in very long instruction word (VLIW) and instruction-level parallel (ILP) processing when he joined the facility in 1989, resulting in the development of the explicitly parallel instruction computing style of architecture that is

the basis for the IA-64. A cofounder of Cydrome, which developed one of the first VLIW minisupercomputers, Rau taught at the University of Illinois at Urbana-Champaign, authored dozens of articles on VLIW computing, coauthored a book on ILP, and held 15 patents. He also was a recipient of the Eckert-Mauchly Award and a fellow of both IEEE and the ACM.

To learn more about Computer Society awards, including the Rau Award, visit www.computer.org/awards. 

Computer Society Certifies Software Professionals

In a world where software is pervasive, the need for skilled, competent, software development professionals is greater than ever. In response to this growing need, the IEEE Computer Society offers two certification programs for computing professionals.

The Certified Software Development Professional credential is intended for mid-career software professionals looking to advance in their careers. The Certified Software Development Associate credential is intended for graduating software engineers and entry-level software professionals.

In 2007, a panel of experts developed a set of exam specifications for the CSDP and CSDA via a job analysis process, an industry-accepted systematic procedure for identifying and validating a job’s performance domain and the knowledge and skills that are necessary to perform that job.

CSDP EXAMINATION

The CSDP certification is the only software development certification that features all of the components of a professional certification. The

program requires exam-based testing that demonstrates mastery of a body of knowledge (BOK); an extensive experience base in performing the work or profession being certified; and continuing professional education, measured and relevant to the BOK.

The CSDP examination covers the following knowledge areas, with the approximate percentage of questions that are covered in the exam:

- I. Software Requirements 11%
- II. Software Design 11%
- III. Software Construction 9%
- IV. Software Testing 11%
- V. Software Maintenance 5%
- VI. Software Configuration Management 5%
- VII. Software Engineering Management 8%
- VIII. Software Engineering Process 7%
- IX. Software Engineering Methods 4%
- X. Software Quality 7%
- XI. Software Engineering Professional Practice 5%
- XII. Software Engineering Economics 5%
- XIV. Mathematical Foundations 3%


- XV. Engineering Foundations 4%

CSDA EXAMINATION

The CSDA credential is a software development certification that is intended for recent software engineering graduates or entry-level software development professionals. First launched in May 2008, the CSDA exam covers many of the same knowledge areas as the CSDP, but in differing percentages.

EXAM REFRESH UNDER WAY

More than three dozen holders of the Certified Software Development Professional credential, representing academic and industrial institutions from around the world, are participating in a project to develop and test the new CSDA examination, which is expected to debut in January 2012. More than 500 new items have been written and reviewed to date. Beta testers will begin taking the exam later this year.

Find more information about IEEE Computer Society certification programs, including registration forms, at www.computer.org/certification. 

Larson Best Paper Scholarship Seeks Applications

In 1983, the Larson family established the Lance Stafford Larson Award in memory of their son, who died in an electrical accident while an undergraduate at the University of Maryland. The Larson family, which includes IEEE past-president Robert Larson, created this award to encourage students to develop excellence in their communication skills and to motivate them toward achievement in the field of computer science. One \$500 award is given each year to the first-place winner. First-, second-, and third-place winners also receive a certificate of commendation and a writing implement. All undergraduate students who are IEEE Computer Society members can compete for the award.

The Lance Stafford Larson Award is presented for the best student paper. Only papers concerning computer-related subjects of no more than 20 pages are eligible. Papers will be judged on technical content, writing skill, and overall presentation. A minimum GPA of 3.0 is required.

To apply, complete the application at www.computer.org/portal/web/studentactivities/larson by 31 October and return to John Daniel (jw.daniel@computer.org).

UPE Student Award Recognizes Academic Excellence

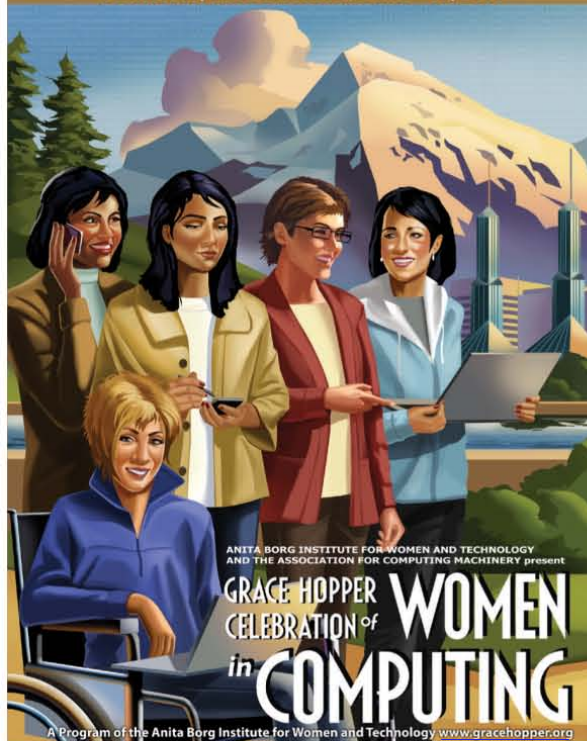
Upsilon Pi Epsilon, an international honor society for the computing and information disciplines, encourages academic excellence for students in computing fields. The UPE Student Award for Academic Excellence, presented in cooperation with the IEEE Computer Society, aims to emphasize the importance of academic achievement to our future computer professionals. Up to four awards of \$500 each are given each year to competition winners, who also receive a certificate of commendation and a periodical subscription for one year. UPE award criteria require the same minimum grade point average that UPE membership requires, which is currently a minimum of 3.0 out of a possible 4.0.

Student winners of the Computer Society's Richard Merwin or UPE/CS award within the previous 13 months are not eligible for this award. All other full-time graduate and undergraduate students who are IEEE Computer Society members are eligible. Judging is based on academic achievement, extracurricular activities related to the computing discipline, and letters of recommendation.

The application process for the UPE Student Award for Academic Excellence is described at www.computer.org/portal/web/studentactivities/upe. Submissions are due by 31 October.

REGISTRATION NOW OPEN FOR THE 2011 GRACE HOPPER CELEBRATION OF WOMEN IN COMPUTING

PORTLAND, OREGON NOVEMBER 9 - 12, 2011



One of the largest technical conferences for women in the world features:

- Keynote addresses by Sheryl Sandberg, COO of Facebook and Shirley Jackson, President of Rensselaer Polytechnic Institute
- Over 600 speakers, leaders in their technical fields, representing industry, academia and government
- A Career Fair with recruiters from over 70 leading technology companies and academic institutions. Come find your next great career opportunity.
- Workshops to develop skills on leadership and networking, for every level from undergraduate to executive level

Register Today!

Go to www.gracehopper.org for more details



ANITA BORG INSTITUTE
FOR WOMEN AND TECHNOLOGY

Come Join Us in Portland, Oregon for the 2011 Grace Hopper Celebration

CALL AND CALENDAR

CALLS FOR ARTICLES
FOR COMPUTER

Computer seeks submissions for an April 2012 special issue on interaction beyond the keyboard.

Interaction with computers has become an integral part of daily life for most people. As computing technologies proliferate, simple user interfaces and ease of use become key success factors for a wide range of products.

Although the keyboard and mouse are still the dominant user interfaces in home and office environments, with the massive increase in mobile device usage and the many new interaction technologies available, the way we interact with computers is becoming richer and more diverse. Touch-enabled surfaces, natural gestures, implicit interaction, and tangible user interfaces mark some of these trends.

Authors are encouraged to submit original research that describes groundbreaking new devices, methods, and approaches to human-computer interaction in a world of ubiquitous computer use. Suitable topics include interactive surfaces and tabletop computing, tangible interaction and graspable user interfaces, and user interfaces based on physiological sensors and actuators.

Direct inquiries to guest editor Albrecht Schmidt of the University of Stuttgart at albrecht@computer.org.

Paper submissions are due **1 November**. For author guidelines and information on the electronic submission process, visit www.computer.org/portal/web/peerreviewmagazines/computer.

Computer seeks submissions for a September 2012 special issue on



modeling and simulation of smart and green computing systems.

Sustainable and efficient utilization of available energy resources is perhaps the fundamental challenge of the current century. Academic and industrial communities have invested significant resources in developing new solutions to address energy-efficiency challenges in several areas including IT and telecommunications, green buildings and cities, and the smart grid.

Modeling and simulation methodologies are necessary for the comprehensive performance evaluation that precedes costly prototyping activities for such complex, large-scale systems. This special issue aims to disseminate the latest advances in modeling and simulation of smart and green computing systems, which are critical from the perspective of sustainable economic growth and environmental conservation.

Topics of interest include modeling and simulations of energy-efficient computing systems, green commu-

nications systems, and smart grid applications. For author guidelines and information on how to submit a manuscript electronically, visit www.computer.org/portal/web/peerreviewmagazines/computer.

Articles are due by **1 March 2012**. Visit www.computer.org/portal/web/computingnow/cocfp9 to view the complete call for papers.

CALLS FOR ARTICLES FOR
IEEE CS PUBLICATIONS

IEEE Computer Graphics and Applications plans a September/October 2012 special issue titled "Biomedical Applications: From Data Capture to Modeling."

Today's broad array of image and data-capture tools is dramatically changing the understanding of biological processes. Imaging modalities like computed tomography and magnetic resonance imaging let users visualize and track complex biological processes. Motion capture can help researchers to understand how animals move.

Just as calculus helped physicists understand and model the mechanical world, computers can help model complex biological systems for researchers to use in reasoning and making predictions about them. Computer graphics techniques and algorithms—from modeling to animation—make this possible.

SUBMISSION INSTRUCTIONS

The Call and Calendar section lists conferences, symposia, and workshops that the IEEE Computer Society sponsors or cooperates in presenting.

Visit www.computer.org/conferences for instructions on how to submit conference or call listings as well as a more complete listing of upcoming computer-related conferences.

EVENTS IN 2011-2012

November

6-10 ASE 2011
 6-13 ICCV 2011
 7-9 ICTAI 2011
 12-18 SC 2011
 21-23 NCCA 2011

December

5-8 E-Science 2011
 7-9 ICPADS 2011
 11-14 ICDM 2011
 18-21 HiPC 2011

January 2012

4-7 HICSS 2012
 9-11 WACV 2012

HICSS 2012

Now in its 45th year, the Hawai'i International Conference on System Sciences is one of the longest-standing continuously running scientific conferences. This conference brings together researchers in a friendly atmosphere conducive to the free exchange of scientific ideas. Unique characteristics of the conference include a matrix structure of tracks and themes that highlight research into a rich mixture of computer-based applications and system technologies.

The conference is organized into minitracks, symposia, workshops, and tutorials. Some tutorials are actually advanced seminars or in-depth surveys for those who already have a significant background in the area under discussion.

HICSS 2012 takes place **4-7 January** in Manoa, Hawai'i. Visit www.hicss.hawaii.edu for complete conference details.

This special issue is dedicated to multidisciplinary efforts in building, verifying, and understanding biological models. The guest editors seek contributions that address biology problems ranging from biochemistry through computational anatomy.

Articles are due by **14 January 2012**. Visit www.computer.org/portal/web/computingnow/cgacfp5 to view the complete call for papers.

CALENDAR

NOVEMBER 2011

6-10 Nov: ASE 2011, 26th IEEE/ACM Int'l Conf. on Automated Soft-

ware Eng., Lawrence, Kansas; www.continuinged.ku.edu/programs/ase

6-13 Nov: ICCV 2011, 13th Int'l Conf. on Computer Vision, Barcelona, Spain; www.iccv2011.org

7-9 Nov: ICTAI 2011, 23rd IEEE Int'l Conf. on Tools with Artificial Intelligence, Boca Raton, Florida; www.cse.fau.edu/ictai2011

12-18 Nov: SC 2011, ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage, and Analysis, Seattle; <http://sc11.supercomputing.org>

21-23 Nov: NCCA 2011, First IEEE Symp. on Network Cloud Computing and Applications, Toulouse, France; <http://sites.google.com/site/ieeencca2011>

DECEMBER 2011

5-8 Dec: E-Science 2011, 7th Int'l Conf. on e-Science, Stockholm; www.escience2011.org

7-9 Dec: ICPADS 2011, IEEE Int'l Conf. on Parallel and Distributed Systems, Tainan, Taiwan; <http://conf.ncku.edu.tw/icpads>

11-14 Dec: ICDM 2011, IEEE Int'l Conf. on Data Mining, Vancouver, Canada; <http://webdocs.cs.ualberta.ca/~icdm2011/index.php>

18-21 Dec: HiPC 2011, IEEE Int'l Conf. on High-Performance Computing, Bangalore, India; www.hipc.org

JANUARY 2012

4-7 Jan: HICSS 2012, 36th Hawai'i Int'l Conf. on System Sciences, Manoa, Hawai'i; www.hicss.hawaii.edu

9-11 Dec: WACV 2011, Workshop on the Applications of Computer Vision, Breckenridge, Colorado; www.wacv2012.org

COMPUTING THEN

Learn about computing history and the people who shaped it.

<http://computingnow.computer.org/ct>

CAREER OPPORTUNITIES

LEAD DEVELOPER F/T (Poughkeepsie, NY) Must have Bach deg or the foreign equiv in Electronic Engg, Engg, Electronics & Comm or related with five (5) yrs of progressive exp leading a 5 member team & be proficient in the application analysis, design, development, testing through full product development life cycle, and release process. Responsible for analyzing and collecting requirements, designing, developing, testing and deploying Java/J2EE systems using Web logic/JBoss/Tomcat ,Web Services (SOAP) , WSDL, Apache Axis 1.2, Eclipse/MyEclipse, Agentis, Rational Rose, JDK 1.4 and above, EJB, JMS, MQ Series, JSP, Servlets, Spring framework, XML, FOP, Struts, Log4j, Oracle 9i/10g, DB2, VSTS, Teamprise. Manage, review, and approve application design and changes, also ensure proper documentation of the product. Provide training on technologies to the team members. Send resume: Indotronics Int'l Corp., Recruiting (CBS), 331 Main St, Poughkeepsie, NY 12601.

LEAD MDMS ENGINEER in Glen Ellyn, IL to upgrade the Master Data Management Service (MDMS) from Version 7.0 to Version 9.0 for a business info project

for ISG's client, Caterpillar, Inc & to guide the transformation, leading the modeling, testing, & implmnt of the upgrade, & to engage in ETL processes such as extracting, loading, data mapping, XML, data modeling, composite transactions, behavior extensions, & de-duplication & dsgn web services out of IBM MDM. Req'd: Master's deg in Comp Sci, Comp Engg or Bachelor's deg in Comp Sci, Comp Engg + 5 yrs of continuous, progressive exp as Comp Software Engr/Lead MDMS Dvlpr. Mail resumes to Joselito Salas, Innovative Systems Group, Inc., 799 Roosevelt Rd. Building 4, Ste 109, Glen Ellyn, IL 60137. Ref# 110302450B. No Calls. No recruiters. Job applicants only.

BUSINESS SYSTEMS ANALYST. Analyze business needs and recommend solutions to procedure, work flow, and policy problems for life insurance administration systems throughout the Software Development Life Cycle (SDLC). Conduct business specification reviews with stakeholders. Conduct design activities for life insurance including process mapping, testing, readiness using process mapping tools and testing tools (Quick Test Pro, Test Director,

or Quality Center). Direct applications to: M.Gagne, Mailstation F110, Massachusetts Mutual Life Insurance Company, 1295 State Street, Springfield, MA 01111; Please Reference Job ID: 50346603.

SIEMENS PLM SOFTWARE INC. has an opening in Peoria, IL for Software Product Consultant to design, develop & implement solutions for business process around Teamcenter products. Requires Bachelor's degree & 10 yrs exp in Software Development or Implementation. Send resumes to PLMCareers@ugs.com. Job code UG591 must be referenced in email subject line. EOE.

PROGRAMMER ANALYST: Analysis, research, dsgn, write specs, maintain, enhance & dvlp using Oracle Apps 11i/R12 (AP, AR, GL, FA, CM, INV, BOM, WIP & iStore), Toad, Discoverer 4i/9i, Workflow, UNIX, SQL loader, Forms & Report Builder 6i, FSG, XML, BPEL, SOA, OBIEE, AOL, Unix Informatica, Cognos, PL/SQL, VB, ASP, ASP.NET, C#, SQL Server, Java, OAF, HTML & DHTML. Freq travel reqd. Req's MS Comp Sci, Eng or rel. Mail re-

Baylor University

Assistant, Associate or Full Professor of Computer Science

The Department of Computer Science seeks a productive scholar and dedicated teacher for a tenure-track position beginning August, 2012. The ideal candidate will hold a terminal degree in Computer Science or closely related field, demonstrate scholarly capability and an established and active independent research agenda in one of several core areas of interest, including, but not limited to, game design and development, software engineering, computational biology, machine learning or large-scale data mining. A successful candidate will also exhibit a passion for teaching and mentoring at the graduate and undergraduate level. For position details and application information please visit: <http://www.baylor.edu/hr/index.php?id=81302>

The Department: The Department offers a CSAB-accredited B.S. in Computer Science degree, a B.A. degree with a major in Computer Science, a B.S. in Informatics with a major in Bioinformatics, and a M.S. degree in Computer Science. The Department has 13 full-time faculty members, over 250 undergraduate majors and approximately 30 master's students. We are currently seeking approval to offer a dual Ph.D. degree in cooperation with a well-established partner institution. Interested candidates may contact any faculty member to ask questions and/or visit the web site of the School of Engineering and Computer Science at <http://www.ecs.baylor.edu>.

The University: Chartered in 1845 by the Republic of Texas, Baylor University is the oldest university in Texas and the world's largest Baptist University. It is situated on a 500-acre campus next to the Brazos River and annually enrolls more than 14,000 students in over 150 baccalaureate and 80 graduate programs. Baylor's mission is to educate men and women for worldwide leadership and service by integrating academic excellence and Christian commitment within a caring community. Baylor is actively recruiting new faculty with a strong commitment to the classroom and an equally strong commitment to discovering new knowledge as Baylor aspires to become a top tier research university while reaffirming and strengthening its distinctive Christian mission as described in Baylor 2012 (www.baylor.edu/vision/).

Application Procedure: Applications, including detailed curriculum vitae, a statement demonstrating an active Christian faith, and contact information for three references should be sent to: Chair Search Committee, Department of Computer Science, Baylor University, One Bear Place #97356, Waco, TX 76798-7356.

Appointment Date: Fall 2012. For full consideration, applications should be received by January 1, 2012.

Baylor is a Baptist university affiliated with the Baptist General Convention of Texas. As an Affirmative Action/Equal Employment Opportunity employer, Baylor encourages minorities, women, veterans, and persons with disabilities to apply.

FACULTY POSITION

CONCORDIA UNIVERSITY MONTREAL, CANADA

The Department of Electrical and Computer Engineering is seeking excellent **tenure-track candidates at the Assistant or Associate Professor ranks in the area of Computer Engineering (Software)**. Applicants must possess expertise and research interests in cloud-computing, virtualization, and quality of service engineering and monitoring. Knowledge of software engineering techniques is required.

Applicants must hold a Ph.D. degree in Computer Engineering, Electrical Engineering, Computer Science or Software Engineering. The language of instruction at Concordia is English; however, knowledge of French is an asset. Detailed applications should include, a CV, teaching and research statements, and names of three referees, and shall be accepted in electronic form (PDF) until positions are filled.

Applications should be sent to Dr. W. E. Lynch, Chair, at blynch@ece.concordia.ca. Further details on these positions are available at www.ece.concordia.ca.

All qualified candidates are encouraged to apply; however, Canadian citizens and permanent residents of Canada will be given priority. Concordia University is committed to employment equity.





Think about impacting 1 out of every 2 people online—in innovative & imaginative ways that are uniquely Yahoo! We do just that each & every day, & you could too. After all, it's big thinkers like you who will create the next generation of Internet experiences for consumers & advertisers across the globe. Now's the time to show the world what you've got. Put your ideas to work for over half a billion people. Yahoo! currently has multiple position openings for various levels & types in the following locations:

Santa Clara/Sunnyvale, CA:

* **Applied Scientist** (Req.#40720). Working with the Yahoo! Labs organization, deliver both fundamental and applied scientific leadership through published research and new engineering products.

* **Architect** (Req.#40721). Build deliverables on time with available resources & in compliance with principles such as modularity, scalability, testability, availability, operability, security, & global deployability.

* **Business Systems Analyst** (Req.#40723). Work closely with customers to identify business roadmap and requirements. Analyze science, engineering, business, and all other data processing problems for application to electronic data processing systems.

* **Data Insights Manager** (Req.#40724) Responsible for extracting, tracking, reporting & analyzing site performance & quality metrics from the web that assist decision-making for several departments (e.g., marketing, advertising & sponsorship sales, product development, finance, technology, & ecommerce).

* **Director, Product Management** (Req.#40725). Manage a suite of products or a large complex product with multiple interdependencies.

* **Engineering Manager** (Req.#40727). Plan, direct, or coordinate software engineering activities for software enhancements & new products that provide behind-the-scenes functionality for global online network of integrated services.

* **Interaction Designer** (Req.#40764) Deliver conceptual models, task flows, site-maps, navigation models and wireframes for the interface and specifications for development that clarify understanding of user actions.

* **Manager, Database Administration** (Req.#40729). Manage a team of full-time employees with responsibility for planning computerized databases, including base definition, structure, documentation, sizing, capacity planning & protection to meet the needs of Yahoo! businesses.

* **Manager, Service Engineering** (Req.#40728). Manage a team of full-time employees with responsibility focusing on the availability, performance, scalability & maintainability of Yahoo!'s Production Applications & Services.

* **Manager, Software Development Engineering, Applications** (Req.#40730). Directs the activities of a software applications development function for software application enhancements & new products.

* **Manager, Storage Engineering** (Req.#40733). Manage a team of full-time employees with responsibility for overseeing design, planning, installation, configuration, security, & usage of Yahoo! production mass storage subsystems, storage networks & backups.

* **Network Engineer** (Req.# 40734). Responsible for overall strategy, network architecture, platform management, service provider management, implementation & operations of Yahoo!'s worldwide production network or internal data communications systems.

* **Paranoid** (Software Engineer) (Req.#40755) Works closely with others to protect Yahoo! data & resources by helping define & build systems that remain dependable in the face of bugs, malice or bad luck.

* **Performance Engineer** (Req.#40736). Support developer with optimization & troubleshooting of Yahoo! properties.

* **Product Manager** (Req.#40737). Responsible for the definition & ongoing management of a product or family of products at all stages of the product lifecycle including defining a product concept & requirements.

* **Project/Program Manager** (Req.#40740). Responsible for the overall planning & coordination of a project/program from inception to completion.

* **Research Engineer** (Req.#40741). Work alongside our world-class scientists to ensure state-of-the-art scientific excellence in Yahoo! products, responsible for implementation & development including data processing & analysis on several types of projects, such as creation of large-scale research prototypes.

* **Research Scientist** (Req.#40742). Work in the Yahoo! Labs organization to deliver both fundamental & applied scientific leadership through published research & new technologies powering the company's products.

* **Search Editor** (Req.#40743). Responsible for improving the quality of all search products & representing the interests & expectations of users in evaluating product content quality.

* **Service Engineer** (Req.# 40744). Focused on the availability, performance, scalability & maintainability of Yahoo!'s Production Applications & Services.

* **Software Development Engineer, Applications** (Req.# 40745). Design, build, implement, modify, test, debug & deploy applications for customer or partner facing products or internal users.

* **Software Development Engineer, Systems** (Req.# 40746). Design, build, implement, modify, test, debug & deploy software systems, large-scale infrastructure platforms & network services that are the foundation of multiple Yahoo! properties & applications.

* **Software Quality Assurance Engineer** (Req.# 40747). Responsible for developing quality standards for Yahoo! products, certification & execution of software test plans & analysis of test results.

* **Data Insights Analyst** (Req.#40749). Responsible for extracting, tracking, reporting and analyzing site performance and quality metrics from the web that assist decision-making for several departments

* **Storage Engineer** (Req.#40763). Oversee design, planning, installation, configuration, security, & usage of Yahoo! production mass storage subsystems, storage networks & backups.

* **Systems Administrator** (Req.#40750). Determines network/system requirements for designing and developing hardware and software solutions.

* **Technical Operations Specialist** (Req.#40751). Provide technical support to the Yahoo! Network and partners on new inventory program set-up, property changes, and a number of other mission-critical display advertising support areas.

* **Visual Designer** (Req.#40752) Responsible for translating brand attributes & business goals into design & layout solutions for digitally delivered applications & content.

San Francisco, CA

* **Software Development Engineer, Applications** (Req.#40718). Design, build, implement, modify, test, debug & deploy applications for customer or partner facing products or internal users.

* **Software Development Engineer, Systems** (Req.# 40719). Design, build, implement, modify, test, debug & deploy software systems, large-scale infrastructure platforms & network services that are the foundation of multiple Yahoo! properties & applications.

* **Architect** (Req.42249). Build deliverables on time with available resources & in compliance with principles such as modularity, scalability, testability, availability, operability, security, & global deployability.

Burbank, CA:

* **Product Manager** (Req.#40759). Responsible for the definition & ongoing management of a product or family of products at all stages of the product lifecycle including defining a product concept & requirements.

* **Service Engineer** (Req.# 40760). Focused on the availability, performance, scalability & maintainability of Yahoo!'s Production Applications & Services.

* **Software Development Engineer, Applications** (Req.# 40690). Design, build, implement, modify, test, debug & deploy applications for customer or partner facing products or internal users.

* **Software Development Engineer, Systems** (Req.# 40691). Design, build, implement, modify, test, debug & deploy software systems, large-scale infrastructure platforms & network services that are the foundation of multiple Yahoo! properties & applications.

* **Software Quality Assurance Engineer** (Req.# 40692). Responsible for developing quality standards for Yahoo! products, certification & execution of software test plans & analysis of test results.

Santa Monica, CA:

* **Experience Designer** (Req.#40753) Creates visual design concepts that support the brand as well as the business goals.

* **Interaction Designer** (Req.#40761) Deliver conceptual models, task flows, site-maps, navigation models and wireframes for the interface and specifications for development that clarify understanding of user actions.

* **Product Manager** (Req.#40754). Responsible for the definition & ongoing management of a product or family of products at all stages of the product lifecycle including defining a product concept & requirements.

* **Software Development Engineer, Applications** (Req.# 40756). Design, build, implement, modify, test, debug & deploy applications for customer or partner facing products or internal users.

* **Software Development Engineer, Systems** (Req.# 40757). Design, build, implement, modify, test, debug & deploy software systems, large-scale infrastructure platforms & network services that are the foundation of multiple Yahoo! properties & applications.

New York, NY:

* **Research Scientist** (Req.#40709). Work in the Yahoo! Labs organization to deliver both fundamental & applied scientific leadership through published research & new technologies powering the company's products.

* **Software Development Engineer, Applications** (Req.# 40710). Design, build, implement, modify, test, debug & deploy applications for customer or partner facing products or internal users.

* **Software Development Engineer, Systems** (Req.# 40711). Design, build, implement, modify, test, debug & deploy software systems, large-scale infrastructure platforms & network services that are the foundation of multiple Yahoo! properties & applications.

Richardson, TX

* **Product Manager** (Req.#40712). Responsible for the definition & ongoing management of a product or family of products at all stages of the product lifecycle including defining a product concept & requirements.

* **Software Development Engineer, Applications** (Req.#40713). Design, build, implement, modify, test, debug & deploy applications for customer or partner facing products or internal users.

* **Software Development Engineer, Systems** (Req.# 40715). Design, build, implement, modify, test, debug & deploy software systems, large-scale infrastructure platforms & network services that are the foundation of multiple Yahoo! properties & applications.

* **Software Quality Assurance Engineer** (Req.# 40716). Responsible for developing quality standards for Yahoo! products, certification & execution of software test plans & analysis of test results.

* **Storage Engineer** (Req.#40717). Oversee design, planning, installation, configuration, security, & usage of Yahoo! production mass storage subsystems, storage networks & backups.

We offer competitive salaries & comprehensive benefits packages. Please submit resume & cover letter (in WORD.doc format) to yahoo-caljobs@yahoo-inc.com & indicate applicable Req.#. EOE

Microsoft Corporation currently has the following openings (job opportunities available at all levels, e.g., Principal, Senior and Lead levels):

Cambridge, MA

Program Managers: Coordinate program development of computer software applications, systems or services. <http://bit.ly/MSJobs-ProgMgr>

Charlotte, NC

Program Managers: Coordinate program development of computer software applications, systems or services. <http://bit.ly/MSJobs-ProgMgr>

Support Engineers / Escalation Engineers / Support Escalation Engineers: Provide technical support on issues experienced with Microsoft technologies. <http://bit.ly/MSJobs-Support>

Senior Technical Account Manager: Coordinate with other Microsoft groups to assist customer in the development, use, support and implementation of Microsoft solutions. www.jobs-microsoft.com/job/go/1429974/

Chevy Chase, MD

Product Manager: Develop or oversee development of the marketing plan for a product or product line. <http://bit.ly/MSJobs-Marketing>

Systems Analyst (Services Manager): Responsible for the delivery of design, planning, and implementation services that provide IT solutions to customers and partners. www.jobs-microsoft.com/job/go/1384128/ and www.jobs-microsoft.com/job/go/1330764/

Downers Grove, IL

Senior Consultant: Deliver design, planning, and implementation services that provide IT solutions to customers and partners. www.jobs-microsoft.com/job/go/1446629/

PFE Manager: Responsible for hiring, developing and retaining a team of skilled Premier Field Engineers to ensure customers are satisfied with Premier Field Engineering support. www.jobs-microsoft.com/job/go/1446578/

Fargo, ND

Support Engineers / Escalation Engineers / Support Escalation Engineers: Provide technical support on issues experienced with Microsoft technologies. <http://bit.ly/MSJobs-Support>

Partner Consultant (Technical Lead): Manage partner support for a delivery team. www.jobs-microsoft.com/job/go/1435864/

Fort Lauderdale, FL

Solution Sales Specialist: Develop customized sales solutions to support team deliverables and the overall marketing vision. www.jobs-microsoft.com/job/go/1413446/

Marketing Manager (IW BG LEAD): Define the business & marketing strategy to successfully manage the portfolio of company products. www.jobs-microsoft.com/job/go/1410200/

Houston, TX

Dynamics ERP Solutions Specialist: Deliver total solution map and vision to prospects/customers/partners. www.jobs-microsoft.com/job/go/1330754/

New York, NY

Price Display Manager: Design, deploy, and support tools to implement APS display pricing strategy. www.jobs-microsoft.com/job/go/1409805/

Vertical Analytics Manager: Develop actionable insights through application of statistical theory and methods to analyze consumer digital media usage data. www.jobs-microsoft.com/job/go/1440235/

Irving, TX

Support Engineers / Escalation Engineers / Support Escalation Engineers: Provide technical support on issues experienced with Microsoft technologies. <http://bit.ly/MSJobs-Support>

Plano, TX

Engagement Manager: Manage the deployment of computer products and services using knowledge of software applications and C++ and C#. www.jobs-microsoft.com/job/go/1199150/

Mountain View, CA

Program Managers: Coordinate program development of computer software applications, systems or services. <http://bit.ly/MSJobs-ProgMgr>

Researchers: Conducting research and lead research collaborations that yield new insights, theories, analyses, data, algorithms, and prototypes. <http://bit.ly/MSJobs-Research>

Business Development & Strategy Analyst: Design and implement framework for executing Corporate/Field partner selection, segmentation, and engagement in domestic and international subsidiaries. www.jobs-microsoft.com/job/go/1443931/

Sr. Network Services Engineer: Responsible for designing and configuring online network services and devices. www.jobs-microsoft.com/job/go/1323597/

Technical Account Manager: Provide support delivery management of Speech at Microsoft Team issues and services designed to improve customer IT operational health in assigned account(s). www.jobs-microsoft.com/job/go/1446561/

Technology Architect: Manage the sales, discovery, and design phases of computer software deployments, with broad focus around enterprise, industry, platform, and solutions. www.jobs-microsoft.com/job/go/1440033/

Senior Network Engineer: Responsible for developing a technical roadmap for major

online services network infrastructure. www.jobs-microsoft.com/job/go/1421852/

Redmond, WA

Program Managers: Coordinate program development of computer software applications, systems or services. <http://bit.ly/MSJobs-ProgMgr>

Product Manager: Develop or oversee development of the marketing plan for a product or product line. <http://bit.ly/MSJobs-Marketing>

Product Marketing Manager: Lead strategy and implementation for taking products to market and optimizing return on investment. <http://bit.ly/MSJobs-Marketing>

Support Engineers / Escalation Engineers / Support Escalation Engineers: Provide technical support on issues experienced with Microsoft technologies. <http://bit.ly/MSJobs-Support>

Service Engineers, Service Operations Engineers, and Systems/Operations Engineers: Plan, architect, deploy and/or support complex client/server or database software systems. <http://bit.ly/MSJobs-SysOps>

User Experience Designers: Develop user interface and user interaction designs, prototypes and/or concepts for business productivity, entertainment or other software or hardware applications. <http://bit.ly/MSJobs-UX>

Premier Field Engineers: Provide customers with reliable technical solutions to complex integration problems associated with business solutions. Requires travel throughout U.S. up to 100%. <http://bit.ly/MSJobs-Support>

Build Engineer: Design, implement, automate, configure, execute and maintain software build and deployment infrastructure and processes. www.jobs-microsoft.com/job/go/1446654/

Build Engineer 2: Implement and maintain software build processes. Develop, debug, and maintain build tools. www.jobs-microsoft.com/job/go/1364572/

Business Process Manager: Coordinate program development of computer software applications, systems, or services. www.jobs-microsoft.com/job/go/1432463/

Compliance Engineer: Assimilate regulatory information (worldwide) on electronic products affecting hardware. www.jobs-microsoft.com/job/go/1364437/

Compliance Engineer: Responsible for the EMC and Radio & Safety Compliance for all Microsoft hardware. www.jobs-microsoft.com/job/go/1418179/

Creative Director: Responsible for managing and directing game design, art and audio features for computer software games and experiences, as well as cloud-based services. www.jobs-microsoft.com/job/go/1441878/

Developer Evangelist: Provide sales technical support, responses to proposals, and/or evangelism to partners and/or customers, working with sales and product development teams. www.jobs-microsoft.com/job/go/1446512/

Game Designer: Responsible for creating innovative game-play solutions as a member of Microsoft Games Studio. www.jobs-microsoft.com/job/go/1446540/

General Manager: Responsible for engineering for the Syndication Partner Program. www.jobs-microsoft.com/job/go/1418452/

General Manager Xbox Live: Align multiple game development studios to be consistent with corporate policy. www.jobs-microsoft.com/job/go/1245297/

Hardware Engineer: Design, implement and test computer hardware. www.jobs-microsoft.com/job/go/1446604/

Integration Systems Analyst: Analyze data processing applications for health care data processing systems. www.jobs-microsoft.com/job/go/1434128/

ISV Developer Evangelist: Provide sales technical support, responses to proposals, and/or evangelism to partners and/or customers, working with sales and product development teams. www.jobs-microsoft.com/job/go/1446541/

IT Test Engineer: Responsible for orchestrating and drive test automation framework implementation. www.jobs-microsoft.com/job/go/1446657/

Lead Worldwide Communities Security: Responsible for the delivery of design, planning, and implementation services that provide IT Security, Rights Management and Information Protection solutions to customers and partners. www.jobs-microsoft.com/job/go/1436078/

Lighting Artist: Work closely with Art, Cinematic and Programming Teams to create industry leading visuals across a variety of environments, characters and cinematic sequences. www.jobs-microsoft.com/job/go/1446564/

Operations Program Manager: Lead a diverse global team to define, develop, implement, and execute global programs, services, processes and/or systems. www.jobs-microsoft.com/job/go/1446751/

Operations Program Manager: Operational Management for applications, processes, offerings and tools in maintenance or run mode. Responsible for detailed process modeling, business rule identification, and evaluation of process performance levels. www.jobs-microsoft.com/job/go/1446584/

Partner Skills Development Manager: Develop and execute partner skills development marketing and communications strategy worldwide. www.jobs-microsoft.com/job/go/1404383/

Principal Development Lead: Responsible for leading team of software developers to develop software applications for an entire product or large feature area. www.jobs-microsoft.com/job/go/1446834/

Principal Director Software Development Engineer in Test: Design, develop, and deliver world class learning solutions using a variety of media, tools and vendors. Travel domestically and internationally as required. www.jobs-microsoft.com/job/go/1443920/

Procurement Business Partner US: Engage and deliver global procurement strategies within an assigned geography or business division. www.jobs-microsoft.com/job/go/1434383/

Product Manager: Research and develop solutions to improve Microsoft profitability and generate value. www.jobs-microsoft.com/job/go/1313446/

Reliability Engineer: Coordinate program development of hardware applications, systems or services, working with development and product planning teams on standard or complex problems. www.jobs-microsoft.com/job/go/1446622/

Security/Engagement Manager: Investigate vulnerabilities in S/W products. www.jobs-microsoft.com/job/go/1247034/

Senior Artist: Act as the technical art expert, handling all aspects of producing prototypes and samples. www.jobs-microsoft.com/job/go/1434199/

Senior CATM Specialist: Microsoft's Customer Advocacy & Technology Management (CATM) team is looking for world-class talent to fill the position of Infrastructure Senior CATM Specialist (SCS). www.jobs-microsoft.com/job/go/1260660/

Senior Industrial Engineer: Oversee the end design of Xbox hardware and accessories programs. www.jobs-microsoft.com/job/go/1237710/

Senior Manager-WEB RPTG/ANALYSIS: Coordinate program development of computer software applications, systems or services. www.jobs-microsoft.com/job/go/1290726/

Senior Marketing Manager: Responsible for developing or overseeing development of the marketing plan for a product or product line. www.jobs-microsoft.com/job/go/1404355/

Senior Network Engineer: Responsible for developing a technical roadmap for major online services network infrastructure. www.jobs-microsoft.com/job/go/1446496/

Service Engineer: Analyze computer systems and services for various business processes. www.jobs-microsoft.com/job/go/1435971/

Service Engineer: Manage teams of Service Engineers and Systems Engineers to provide database, application and hardware support for existing and new technologies in the pre-production and production stages of product implementation. www.jobs-microsoft.com/job/go/1435938/

Service Operations 2 Network Engineer: Design and implement complex connectivity to the corporate intranet, partner networks, and Internet according to established security, design and deployment standards and best practices. www.jobs-microsoft.com/job/go/1432649/

Services Support Manager Lead: Manage a team of Service Managers to provide technical service and support to enterprise customers, partners and/or others on mission critical issues experienced with Microsoft technologies. www.jobs-microsoft.com/job/go/1426930/

SO 2 Systems Engineer: Review requirements, functional specifications, and high and low-level design for new releases. www.jobs-microsoft.com/job/go/1366421/

Software Development Engineer: Responsible for developing computer software applications, systems or services. Position requires travel to various unanticipated locations throughout the U.S. www.jobs-microsoft.com/job/go/1269275/

Software Test Engineer: Define test plans and influence the design and content of release deliverables. www.jobs-microsoft.com/job/go/1423551/

Solutions Planning & Quality Manager: Responsible for designing and planning the architecture of the CRM sales application built on Microsoft Dynamics CRM 4.0 technology. www.jobs-microsoft.com/job/go/1446623/

Sr. OE System Lead: Leading support teams at the front line. www.jobs-microsoft.com/job/go/1444193/

Sr. Release Product Manager: Project manage the release of online services products with cross team engagement. www.jobs-microsoft.com/job/go/1366404/

Sr. SE Security Engineer: Responsible for work with the customer/partners teams to provide business-to-technology mapping for software development engagements. www.jobs-microsoft.com/job/go/1435837/

Sr. Sourcing Manager: Design, execute, and implement category sourcing strategies for the acquisition of technical skills and related support services in Redmond, Washington; India; and China. www.jobs-microsoft.com/job/go/1404492/

Technical Account Manager: Provide support delivery management of Premier issues and services designed to improve customer IT operational health in assigned account(s). www.jobs-microsoft.com/job/go/1446509/

Technical Account Manager: Provide customers with reliable technical solutions to complex integration problems associated with business solutions. Requires travel throughout U.S. up to 100%. www.jobs-microsoft.com/job/go/1446495/

Technical Artist: Responsible for the creation of visual concepts, effects, animation, or other visual images. www.jobs-microsoft.com/job/go/1428537/

Test Engineer 2: Responsible for creating test strategies, test plans, test cases and test data. www.jobs-microsoft.com/job/go/1446838/

Traffic Quality Engineer 3: Solve complex technical issues, identifying trends and fraudulent patterns in data and code. www.jobs-microsoft.com/job/go/1434221/

San Francisco, CA

Program Managers: Coordinate program development of computer software applications, systems or services. <http://bit.ly/MSJobs-ProgMgr>

Product Manager: Develop or oversee development of the marketing plan for a product or product line. <http://bit.ly/MSJobs-Marketing>

Technical Account Manager: Responsible for support issues for premier customers. www.jobs-microsoft.com/job/go/1446616/

Tampa, FL

Technical Account Manager: Provide valuable guidance around operations and optimization of Microsoft's Premier Support customers' IT infrastructure through quality Service Delivery Management. www.jobs-microsoft.com/job/go/1446844/

Tempe, AZ

Consultant I - Deliver design, planning, and implementation services that provide IT solutions to customers and partners. www.jobs-microsoft.com/job/go/1446571/

Waltham, MA

Support Engineers / Escalation Engineers / Support Escalation Engineers: Provide technical support on issues experienced with Microsoft technologies. <http://bit.ly/MSJobs-Support>

Multiple job openings are available for each of these categories. To view all opportunities, detailed job descriptions and minimum requirements, and to apply to specific job opportunities, visit the website address listed for each job category. Microsoft is an equal opportunity employer and supports workplace diversity.

sumes to Nitya Software Solutions Inc., 9690 South 300, Ste 319, Salt Lake City, UT 84070.

TEXAS STATE UNIVERSITY-SAN MARCOS, Department of Computer Science, Assistant Professor, Associate Professor, or Professor.

Applications are invited for a faculty position at the rank of Assistant, Associate, or full Professor to start on September 1, 2012. Consult the department's recruiting page, <http://www.cs.txstate.edu/recruitment/>, for job duties, qualifications, application procedures, and information about the university and the department. Texas State University-San Marcos will not discriminate against any person in employment or exclude any person from participating in or receiving the benefits of any of its activities or programs on any basis prohibited by law, including race, color, age, national origin, religion, sex, disability, veterans' status, or on the basis of sexual orientation. Texas State University-San Marcos is a member of the Texas State University System.

UNIVERSITY OF CHILE.

The Department of Industrial Engineering at the University of Chile invites applications for a faculty position in the area of Technology Management broadly understood. Applicants should possess or be close to completion of a PhD in Information Technology, Information Systems, Technological Innovation, or related fields. They should also exhibit potential for quality research, contribution to relevant applications, and successful teaching at undergraduate and graduate levels. Applications should include cover letter, curriculum vitae, samples of scholarly work and three reference letters should be sent by email to tics@dii.uchile.cl. The deadline for receiving the applications is December 15th, 2011.

TENNESSEE STATE UNIVERSITY, Assistant Professor - Computer Science.

The Computer Science Department at Tennessee State University is seeking outstanding candidates at the assistant professor level for a tenure-track position. Areas of special emphasis include high-performance and cloud computing, bioinformatics, computer networks, and cyber security. The position requires a Ph.D. in computer science, computer engineering, or a related area with outstanding academic credentials. Please apply at <https://jobs.tnstate.edu>.

MICROSOFT CORPORATION

currently has the following openings in Aliso Viejo, CA; Boulder, CO; Cambridge, MA; Durham, NC; Fargo, ND; Hauppauge, NY; Madison, WI; Mountain View, CA; Redmond, WA; St. Paul, MN; and Wash-

ington, D.C. (job opportunities available at all levels, e.g., Principal, Senior and Lead levels): Software Development Engineers, Software Development Engineers in Test, Test Leads: Responsible for developing or testing computer software applications, systems or services. <http://bit.ly/MSJobs-SDE>. Multiple job openings are available for each of these categories. To view all opportunities, detailed job descriptions and minimum requirements, and to apply to specific job opportunities, visit the website address listed for each job category. Microsoft is an equal opportunity employer and supports workplace diversity.

VIRGINIA TECH, www.cs.vt.edu, Artificial Intelligence/Machine Learning Faculty Position. The Department of Computer Science at Virginia Tech invites applications for a full-time tenure-track position at any rank from candidates with expertise in artificial intelligence having specific emphasis on machine learning or probabilistic reasoning. The department plans on making multiple hires over multiple years in this area. Candidates should have a record, appropriate to the desired rank, of scholarship, leadership, and collaboration in computing and interdisciplin-

ary areas; demonstrated ability to contribute to teaching at the undergraduate and graduate levels in AI and related subjects; sensitivity to issues of diversity in the campus community; and the skills to establish and grow a multidisciplinary research group. Salary for suitably qualified applicants is competitive and commensurate with experience. Applications must be submitted online to <https://jobs.vt.edu> for posting #0110872. Applicant screening will begin December 15, 2011 and continue until the position is filled. Inquiries should be directed to Dr. Dennis Kafura, kafura@cs.vt.edu. Virginia Tech is an Equal Opportunity/Affirmative Action Institution. See <http://www.cs.vt.edu/FacultySearch> for more details.

VIRGINIA TECH, www.cs.vt.edu, Data-Intensive Computing Faculty Position. The Department of Computer Science at Virginia Tech invites applications from candidates in data-intensive computing for a full-time tenure-track position at any rank. Examples of research foci in the area of data-intensive computing include, but are not limited to, streaming and sensor data management, scientific databases, networked data management, query languages, workflow/provenance modeling, in-

formation integration, and database designs for modern architectures such as the cloud and pervasive appliances. The successful candidate must be able to conduct an active research program in the management and processing of massive data, such as arise in social networks, biology, GIS, astronomy, text, and/or other emerging large-scale applications. Candidates should have a record, appropriate to their rank, of scholarship, leadership, and collaboration in computing and interdisciplinary areas; demonstrated ability to contribute to teaching at the undergraduate and graduate levels in data-related subjects (e.g., database design and system architectures); sensitivity to issues of diversity in the campus community; and the skills needed to establish and grow a multidisciplinary research group. Salary for suitably qualified applicants is competitive and commensurate with experience. Applications must be submitted online to <https://jobs.vt.edu> for posting #0110874. Applicant screening will begin December 15, 2011 and continue until the position is filled. Inquiries should be directed to Dr. Lenwood S. Heath, heath@vt.edu. Virginia Tech is an Equal Opportunity/Affirmative Action Institution. See <http://www.cs.vt.edu/FacultySearch> for more details.

Boston University

Department of Electrical & Computer Engineering


 BOSTON
UNIVERSITY

The Department of Electrical & Computer Engineering (ECE) at Boston University (BU) is seeking candidates for anticipated faculty positions in Computer Engineering. All areas and ranks will be considered, with particular interest in entry-level candidates in software, security, and computer systems. The Department is seeking to foster growth in the broad, interdisciplinary topics of energy, health, information systems, and cyberphysical systems. Candidates with research interests that transcend the traditional boundaries of ECE are strongly encouraged to apply. Joint appointments with other BU departments and with the Division of Material Science & Engineering and Division of Systems Engineering are possible for candidates with appropriate experience and interests.

Qualified candidates must possess a relevant, earned PhD, and have a demonstrable ability to teach effectively, develop funded research programs in their area of expertise, and contribute to the tradition of excellence in research that is characteristic of the ECE Department. Self-motivated individuals who thrive on challenge and are eager to utilize their expertise to strengthen an ambitious program of departmental enhancement are desired. Women, minorities, and candidates from other underrepresented groups are especially encouraged to apply and help us continue building an exceptional 21st century university department.

ECE at BU is a world-class department with excellent resources that is steadily gaining national and international prominence for its exceptional research and education record. ECE is part of BU's rapidly growing and innovative College of Engineering, and currently consists of 40 faculty members, 200 graduate students, and 250 BS majors. Outstanding collaboration opportunities are available with nationally recognized medical centers and universities/colleges, nearby research centers, and industry throughout the Boston area.

Beyond its research and academic activities, BU has a lively, urban campus situated along the banks of the Charles River in Boston's historic Fenway-Kenmore neighborhood. The campus and surrounding areas offer limitless opportunities for recreational activities, from world-class art and performances to sporting events and fine dining.

Please visit <http://www.bu.edu/ece/facultysearch> for instructions on how to apply. Application deadline is December 31, 2011. The review of applications will begin on October 1, 2011. Therefore, applicants are encouraged to apply early. Boston University is an Equal Opportunity/Affirmative Action Employer.

VIRGINIA TECH, www.cs.vt.edu, Assistant Professor Computer Science (Compilers/Systems Software Engineering). The Department of Computer Science at Virginia Tech invites applications for a full-time tenure-track position at the Assistant Professor rank from candidates with a research focus in the area of compilers, programming languages, or software engineering. The department plans to strategically grow its research presence in these areas over the coming years. Preference is given to candidates whose research efforts lie in applied domains with relevance to computer systems software, particularly for emerging architectures such as many-core processors and GPGPUs. Candidates should have a doctoral degree in Computer Science or a cognate area, a record of significant research achievement and publication, a coherent research and teaching plan showing the potential to secure research funding, build a research program in their area of specialty, and contribute to the department's graduate/undergraduate teaching mission, and sensitivity to issues of diversity in the campus community. The position is part of a coordinated cluster hire of a total of five positions on both the Blacksburg and National Capital Region campuses of Virginia Tech that is intended to synergistically increase research momentum in computer systems, software engineering, security,

and cybersecurity by the Department of Computer Science (CS) and the Bradley Department of Electrical and Computer Engineering (ECE). The successful candidate will join the Department of Computer Science on the Blacksburg campus. Applications must be submitted online to <https://jobs.vt.edu> for posting #0110873. Applicant screening will begin December 15, 2011 and continue until the position is filled. Inquiries should be directed to Dr. Godmar Back, gback@cs.vt.edu. Virginia Tech is an Equal Opportunity/Affirmative Action Institution. See <http://www.cs.vt.edu/FacultySearch> for more details.

THE UNIVERSITY OF MINNESOTA – TWIN CITIES invites applications for faculty positions in Electrical and Computer Engineering in the areas of computer engineering; power and energy systems; nanofabrication, including medical devices and biosciences; and communications/networking. Women and other underrepresented groups, and those with interdisciplinary interests, are especially encouraged to apply. An earned doctorate in an appropriate discipline is required. Rank and salary will be commensurate with qualifications and experience. Positions are open until filled, but for full consideration, apply at <http://www.ece.umn.edu/> by December 1, 2011. The University of Minnesota is an equal opportunity employer and educator.

UNIVERSITY AT BUFFALO, The State University of New York. Faculty Positions in Computer Science and Engineering. Tak Associate/Assistant Professor position: The CSE Department invites excellent candidates in all core areas of Computer science and Engineering to apply for an opening at the associate/assistant professor level. This is an endowed position with five years research funding of \$100K per year. Assistant Professor positions: The CSE Department also invites excellent candidates in all core areas of Computer science and Engineering to apply for openings at the assistant professor level. The department is affiliated with successful centers devoted to biometrics, bioinformatics, biomedical computing, cognitive science, document analysis and recognition, high performance computing, and information assurance. Candidates are expected to have a Ph.D. in Computer Science/Engineering or related field by August 2012, with an excellent publication record and potential for developing a strong funded research program. Applications should be submitted by December 31, 2011 electronically via <http://www.ubjobs.buffalo.edu/>. The University at Buffalo is an Equal Opportunity Employer/Recruiter.

University of Illinois at Urbana-Champaign

The Department of Electrical and Computer Engineering invites applications for faculty positions at all levels and in all areas of electrical and computer engineering, particularly in the areas of control and communications, circuits, energy and power systems, nanoelectronics, nanophotonics, and computing. Applications are encouraged from candidates whose research programs are in traditional as well as nontraditional and interdisciplinary areas of electrical and computer engineering. The department is engaged in exciting new and expanding programs for research, education, and professional development, with strong ties to industry.

Applicants for positions at the assistant professor level must have an earned Ph.D. or equivalent degree, excellent academic credentials, and an outstanding ability to teach effectively at both the graduate and undergraduate levels. Successful candidates will be expected to initiate and carry out independent research and to perform academic duties associated with our B.S., M.S., and Ph.D. programs. Senior level appointments with tenure are available for persons of international stature.

Faculty in the department carry out research in a broad spectrum of areas and are supported by world-class facilities and programs for international work, including the Coordinated Science Laboratory, the Information Trust Institute, the Micro and Nanotechnology Laboratory, the Beckman Institute for Advanced Science and Technology, and several industrial centers. The department has one of the leading programs in the United States, granting approximately 350 B.S. degrees, 100 M.S. degrees, and 60 Ph.D. degrees annually.

In order to ensure full consideration by the Search Committee, applications must be received by December 15, 2011. Salary will be commensurate with qualifications. Preferred starting date is August 16, 2012, but is negotiable. Applications can be submitted by going to <http://jobs.illinois.edu> and uploading a cover letter, CV, research statement, and teaching statement, along with names of three references. For inquiry, please call 217-333-2301 or email ece-recruiting@illinois.edu.

Illinois is an Affirmative Action /Equal Opportunity Employer and welcomes individuals with diverse backgrounds, experiences, and ideas who embrace and value diversity and inclusivity (www.inclusiveillinois.illinois.edu).

STANFORD UNIVERSITY

Department of Computer Science Faculty Openings

The Department of Computer Science at Stanford University invites applications for tenure-track faculty positions at the junior level (Assistant or untenured Associate Professor). We give higher priority to the overall originality and promise of the candidate's work than to the candidate's sub-area of specialization within Computer Science.

We are seeking applicants from all areas of Computer Science, spanning theoretical foundations, systems, software, and applications. We are also interested in applicants doing research at the frontiers of Computer Science with other disciplines, especially those with potential connections to Stanford's main multidisciplinary initiatives: Energy, Human Health, Environment and Sustainability, the Arts and Creativity, and the International Initiative. Interdisciplinary candidates whose research combines other fields of engineering or mathematics with computer science may be considered for a joint appointment in the Institute for Computational and Mathematical Engineering (<http://icme.stanford.edu/>).

Applicants must have completed (or be completing) a Ph.D., must have demonstrated the ability to pursue a program of research, and must have a strong commitment to graduate and undergraduate teaching. A successful candidate will be expected to teach courses at the graduate and undergraduate levels, and to build and lead a team of graduate students in Ph.D. research. Further information about the Computer Science Department can be found at <http://cs.stanford.edu>. The School of Engineering website may be found at <http://soe.stanford.edu>.

Applications should include a curriculum vita, brief statements of research and teaching interests, and the names and contact information of at least four references. Applications should be sent to: <http://soe-apps.stanford.edu/FacultyApplyCS>. Questions should be directed to, Search Committee Chair, c/o Laura Kenny-Carlson, via electronic mail to search@cs.stanford.edu.

The review of applications will begin on November 28, 2011, and applicants are strongly encouraged to submit applications by that date; however, applications will continue to be accepted at least until February 15, 2012.

Stanford University is an equal opportunity employer and is committed to increasing the diversity of its faculty. It welcomes nominations of and applications from women and members of minority groups, as well as others who would bring additional dimensions to the university's research and teaching missions.

University of Maryland, College Park PROFESSOR and DIRECTOR Center for Bioinformatics and Computational Biology

The University of Maryland invites applications for Director of the Center for Bioinformatics and Computational Biology. Candidates are expected to be prominent scholars with publications and research experience at the interface of biological science and computing. Their primary responsibility will be to lead a nationally visible research program complementing existing strengths in computational genomics, proteomics, and molecular evolution. They will also be expected to promote the CBCB, and help build collaborative relationships, both on and off-campus. Information about the Center can be found at www.cbc.umd.edu. Collectively, the CBCB faculty spans the fields of computer science, mathematics and statistics, biology, and biochemistry. The Center is housed in contiguous space and has access to significant high-end computing infrastructure through the University of Maryland Institute for Advanced Computer Studies. CBCB faculty members are also affiliated with at least one other campus academic unit appropriate to their interests. There is ample potential for collaboration with other organizations in the area, such as the NIH, the JCVI, and the Smithsonian Institution. For more information contact the search chair, Thomas D. Kocher (tdk@umd.edu). To apply, send a letter of application, curriculum vitae, and names of three references, following the instructions at <http://cbc.umd.edu/hiring/>. Review of applications will begin November 15, 2011.

The University of Maryland is an affirmative action, equal opportunity employer. Women and minorities are encouraged to apply.

CLASSIFIED LINE AD SUBMISSION DETAILS

Rates are \$400.00 per column inch (\$500 minimum). Eight lines per column inch and average five typeset words per line. Free online listing on careers.computer.org with print ad. Send copy at least one month prior to publication date to: Marian Anderson, Classified Advertising, Computer Magazine, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314; (714) 816-2139; fax (714) 821-4010.

Email: manderson@computer.org

Baylor University Assistant or Associate Professor of Computer Science

Chartered in 1845 by the Republic of Texas, Baylor University is the oldest university in Texas and the world's largest Baptist University. Baylor's mission is to educate men and women for worldwide leadership and service by integrating academic excellence and Christian commitment within a caring community. Baylor is actively recruiting new faculty with a strong commitment to the classroom and an equally strong commitment to discovering new knowledge as Baylor aspires to become a top tier research university while reaffirming and strengthening its distinctive Christian mission as described in Baylor 2012 (www.baylor.edu/vision/). The combination of teaching, research and service has made Baylor one of the best universities for faculty, according to the Chronicle of Higher Education <http://chronicle.com/article/Great-Colleges-to-Work-For/128312/>.

The Department of Computer Science seeks a productive scholar and dedicated teacher for a tenure-track position beginning August, 2012. All specializations will be considered. Game/simulated environments, mobile computing, and graphics are of particular interest. The successful candidate will hold a terminal degree in Computer Science or a closely related field, demonstrate scholarly capability in his or her area of specialization, and exhibit a passion for teaching and mentoring at the graduate and undergraduate level. For position details and application information please visit: <http://www.ecs.baylor.edu>.

The Department: The Department offers a CSAB-accredited B.S. in Computer Science degree, a B.A. degree with a major in Computer Science, a B.S. in Informatics with a major in Bioinformatics, and a M.S. degree in Computer Science. We are currently seeking approval to offer a dual Ph.D. degree in cooperation with a well-established European institution. The Department has 15 full-time faculty, over 370 undergraduate majors and 30 master's students. The Department's greatest strength is the faculty's dedication to the success of the students and each other. Interested candidates may contact any faculty member to ask questions and/or visit the web site of the School of Engineering and Computer Science at <http://www.ecs.baylor.edu>.

The University: Baylor University, situated on a 500-acre campus next to the Brazos River. It annually enrolls more than 14,000 students in over 150 baccalaureate and 80 graduate programs through: the College of Arts and Sciences; the Schools of Business, Education, Engineering and Computer Science, Music, Nursing, Law, Social Work, and Graduate Studies; plus Truett Seminary and the Honors College. For more information see <http://www.baylor.edu>.

Application Procedure: Please submit a letter of application, current curriculum vitae, and transcripts. Include names, addresses, and phone numbers of three individuals from whom you have requested letters of recommendation to: Jeff Donahoo, Ph.D., Search Committee Chair, Baylor University, One Bear Place #97356, Waco, Texas 76798-7356. Materials may be submitted to: Jeff_Donahoo@baylor.edu

Appointment Date: Fall 2012. For full consideration, applications should be received by January 1, 2012. However, applications will be accepted until the position is filled.

Baylor is a Baptist university affiliated with the Baptist General Convention of Texas. As an Affirmative Action/Equal Employment Opportunity employer, Baylor encourages minorities, women, veterans, and persons with disabilities to apply.

Apple is looking for qualified individuals for following 40/hr/wk positions. To apply, mail your resume to 1 Infinite Loop 84-REL, Attn: AEC Staffing-SA, Cupertino, CA 95014 with Req# and copy of ad. Job site & interview, Cupertino, CA. Principals only. EOE.

DW/BI Framework Development Engineer (Req #8993126)

Responsible for developing and building Enterprise Data Warehouse (EDW) to support analytical and reporting needs of global Apple. Req's Bachelor's degree, or foreign equivalent, in Computer Science, or related degree plus six (6) years professional experience in job offered or in a related occupation. Professional experience must be post-baccalaureate and progressive in nature. Must have professional experience with: Framework for processing large data volumes, BW/Business Objects, Extractors, Experience with interfacing & integrating SAP solutions, Master data management solutions, Java based development, Security architectures, defining solution architecture for complex enterprise projects, project management using SDLC.

Technology Manager [Req #9140459]

Responsible for the team that implements interactive, rich media ads served over the air to mobile devices. Requires Bachelor's degree, or foreign equivalent, in Engineering, or related degree and 5 years professional experience in job offered or in a related occupation. Professional experience must be post-baccalaureate and progressive in nature. Also experience with UI and Web engineering as a tech manager; HTML5 (specifically media type tags); CSS3 coding (specifically animation and webkit technologies); JavaScript; DOM manipulation; interplay between JavaScript, HTML & CSS. Will have direct reports.

Sr. Network Engineer [Req #9321656]

Responsible for Apple's Data Center Network Infrastructure. Req's Bachelor's degree, or foreign equivalent, in Computer Science or related degree and five (5) years experience in job offered or related occupations. Professional experience must be post-baccalaureate progressive in nature. Must also have professional experience with: Network infrastructure implementation and support, including the configuration and support of Cisco routers and switches, firewalls, load balancers, and routing protocols; Cisco router and switch configuration; firewall design, implementation and rulesets; load balancer configuration; and common routing protocols including BGP and OSPF. Travel up to 10% domestically.

Design Engineer [Req #9320887]

Complete gate level design and place-and-route of high-speed Req's Master's degree, or foreign equivalent, in Electrical Engineering, Computer Engineering, or related plus One (1) years professional experience in job offered or in a related occupation. Must have professional experience with: Custom VLSI transistor-level design for high-speed, low-power register files, CAM arrays, and dynamic logic; physical verification of electrical circuits, including electrical rule checks, formal verification, electromigration, noise, power, and timing.

Senior Software Engineer [Req #9324278]

Design and implement new OpenGL 3D graphics and display device driver features for the Mac OS X operating system. Requires Master's degree, or foreign equivalent, in Computer Science, Computer Engineering, Engineering, or related and 3 years professional experience in job offered or in a related occupation. Must have academic background or professional experience with: graphics and 3D, 2D/display, and/or Media; C/C++ programming; development of Graphics driver software; kernel and application programming; problem solving, debugging and performance optimization; 3D/2D graphics including OpenGL, graphics architectures.

Apple is looking for qualified individuals for following 40/hr/wk positions. To apply, mail your resume to 1 Infinite Loop 84-REL, Attn: AEC Staffing-SA, Cupertino, CA 95014 with Req# and copy of ad. Job site & interview, Cupertino, CA. Principals only. EOE.

Senior Software Engineer [Req #9336748]

Work on web application development. Master's degree, or foreign equivalent, in Computer Science, Mathematics, Electrical Engineering, Information Technology, Computer Engineering or related plus six (6) years professional experience in job offered or in a related occupation Must have professional experience with: web application development; data structures and algorithms; Java; ORM products; component based web application frameworks; debugging; relational database; SQL; data modeling; MVC web framework.

Software Development Engineer [Req #9341259]

Contribute to the development of embedded software solutions and desktop simulators for our current and future products. Requires Bachelor's degree, or foreign equivalent, in Electrical Engineering, Applied Science, or related and 6 years professional experience in job offered or in a related occupation. Professional experience must be post-baccalaureate and progressive in nature. Also experience with software architecture; electronics hardware; debug of custom hardware; open communication protocols; designing custom protocols; network programming; developing embedded system software; security protocols; developing application software, hardware simulators and/or emulators, test tools, automated unit tests and software frameworks; and SQL/SQLite database programming.

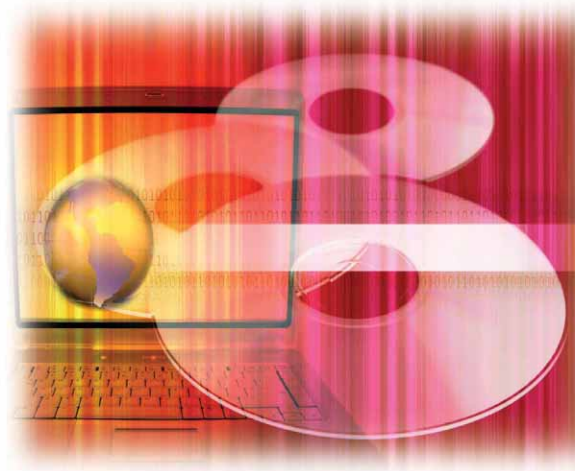
Software Engineer [Req #7694371]

Maintain, enhance, and add new functionalities to Apple compilers. Req.'s Bachelor's degree, or foreign equivalent, in Computer Science, Computer Engineering, Electrical Engineering, Mathematics, Physics, or related plus five (5) years professional experience in job offered or in a related occupation Professional experience must be post-baccalaureate and progressive in nature. Professional experience must be post-baccalaureate and progressive in nature. Must also have professional experience with: C and C++ programming languages; Compiler design and implementation; software testing; computer architecture; debug computer assembly code.

ENTERTAINMENT COMPUTING

Object Digitization for Everyone

Justin Clark, *Microsoft*



Object digitization lets users customize their gameplay experiences, bring the real world into the virtual world, and share more content with others.

Videogames increasingly offer users the opportunity to personalize the gaming experience with their own created 3D content. They often produce this content using editing tools built into the game. For example, recent editions of Microsoft's *Halo* series include a map editor, and with *Spore Creature Creator*, users can craft their own creatures for the popular Electronic Arts game.

Such tools allow many players to unleash their creativity and enrich their gameplay. However, mastering most of these types of tools is so difficult that only a subset of users create content with them, and they can generate only limited types of content in this way.

For more general content, such as arbitrary 3D models, textures, and animations, it often takes even skilled professionals days or perhaps weeks to create individual assets using expensive tools such as Autodesk 3ds Max (formerly 3D Studio Max) and Maya. The cost and skill requirements of these tools make them impractical for most users.

Despite these barriers, enabling all users to create their own content remains highly desirable, as it can lead

to entirely new types of experiences and a potentially vast increase in the amount of content available for the community of people using a product.

OBJECT DIGITIZATION

Toward this end, *object digitization* provides an attractive alternative that is both more powerful and simpler to use than most existing tools. The digitization process uses cameras and other input devices to measure 3D positions on the surfaces of real-world objects and automatically constructs 3D models of these objects.

One technique uses 3D laser scanners to project laser light onto an object and measures surface-point distances based on how long it takes the light to return to a camera. With this method, researchers have constructed highly detailed models of cultural artifacts for study and preservation, as in Stanford University's Digital Michelangelo Project (<http://graphics.stanford.edu/projects/mich>).

In another approach often used in the production of films and games, artists build real-life models of characters and props and then use coordinate measuring machines to digitize the models into virtual

versions. Typical CMMs infer 3D positions from the joint angles of a robot-arm-like touch probe that users manually place at various points on an object's surface.

These object digitization techniques are particularly useful when traditional modeling is not practical or cost-effective. However, the expensive hardware and meticulous user interaction of both methods make them infeasible for most people.

'KINECTING' WITH OBJECTS

Recent innovations in both hardware and software now enable anyone to do object digitization at home. On the hardware side, with Microsoft's Kinect's color and depth cameras, users can capture both an object's color and the shape of its surfaces, at a fraction of the cost of previous cameras. On the software side, researchers have made significant progress in solving the difficult problems of reconstructing 3D models based on input from this type of camera.

Digitizing objects

Microsoft developers have implemented an object digitization

ENTERTAINMENT COMPUTING

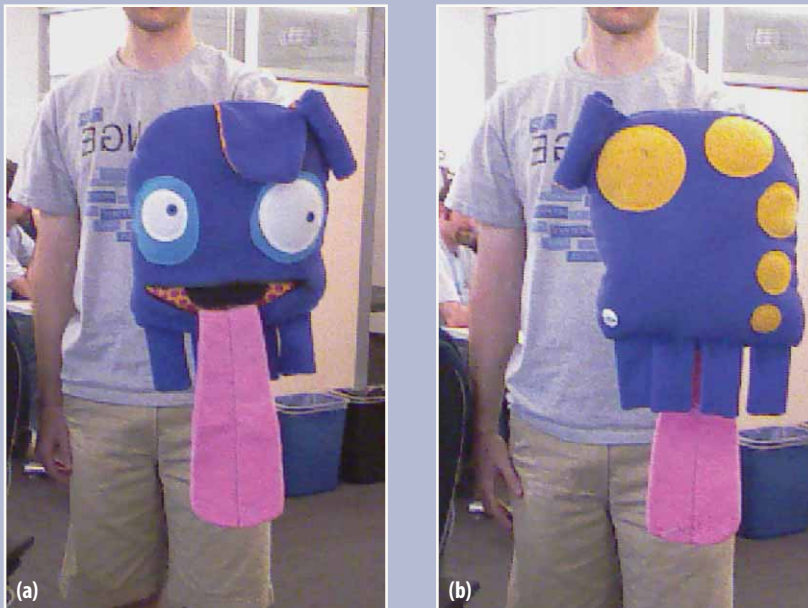


Figure 1. Images taken with a Microsoft Kinect camera show an object's (a) front and (b) back presented to the camera for object digitization.

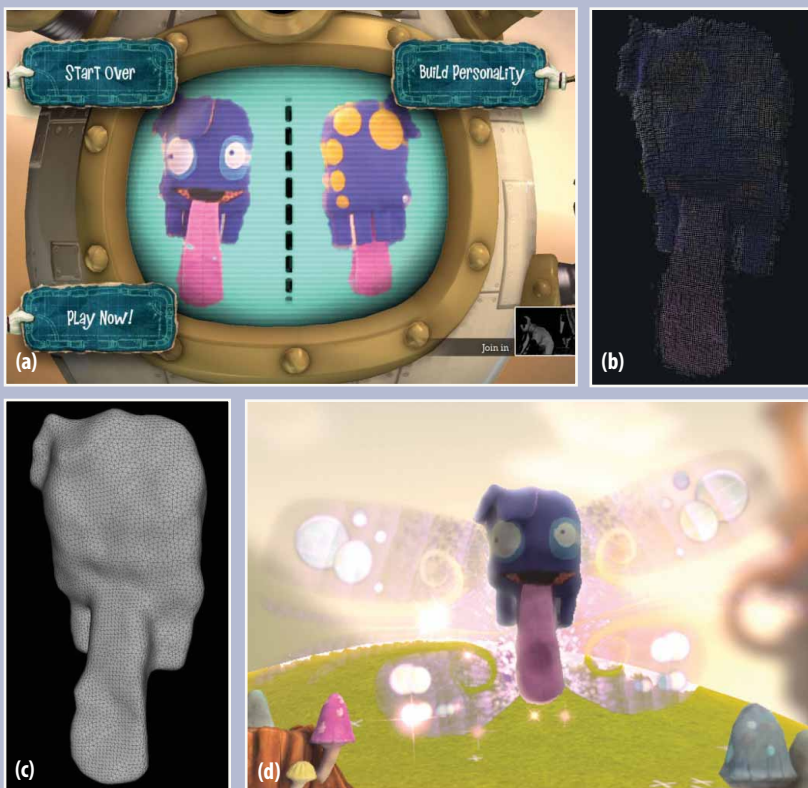


Figure 2. Converting camera images to a 3D model: (a) captured images with nontarget parts removed through segmentation; (b) point cloud created from camera data; (c) 3D surface fit over the point cloud; (d) stylized rendering and animation of the final digitized object

solution that is fast, inexpensive, and easy to use. This solution is demonstrated by the Build a Buddy and Googly Eyes applications available now in Kinect Fun Labs on the Xbox 360. With Build a Buddy, users can scan in a wide variety of objects, such as stuffed animals and favorite toys, and turn them into virtual buddies they can interact with. Googly Eyes brings these digitized objects to life using body poses and gestures.

These new applications employ a very simple digitization process. As Figure 1 shows, the user first holds the front of a real-world object up to Kinect, then turns the object around and presents the back side. After a little processing, the system constructs a 3D model that the user can play with. The entire process takes less than one minute.

From objects to 3D models

Although the user process is simple, going from camera images to a final 3D model requires significant computation. First, the system collects multiple images of an object's front and back from the Kinect camera. Because these images contain extraneous data, such as the user's body, a segmentation process, the result of which is shown in Figure 2a, removes the unwanted data.

The system then merges the images and converts them into patches of 3D points that represent the object's front and back surfaces. As Figure 2b shows, the system aligns the patches to form a single set of points on the surface of the virtual object, called a *point cloud*.

Next, a surface reconstruction algorithm fits a smooth 3D surface over the top of the point cloud, as Figure 2c shows. Performing surface reconstruction quickly is a difficult task enabled by new technology developed at Microsoft Research Asia (K. Zhou et al., "Data-Parallel Octrees for Surface Reconstruction," *IEEE Trans. Visualization and*

Computer Graphics, May 2011, pp. 669-681).

Finally, as Figure 2d shows, the Kinect system maps color data from the captured images onto the 3D surface and fits an animation skeleton into the virtual object so that it can bend, twist, and move around.

Build a Buddy and Googly Eyes represent only the tip of the iceberg of what object digitization can accomplish. The technology puts considerable power into users' hands—to customize their gameplay experiences, bring the real

world into the virtual world, and share more content with others.

What will happen when millions of users are creating and sharing a flood of 3D models with one another? What can they do with those models and how can we make the models richer, more dynamic, and easier to edit? And what will happen when this technology is combined with the 3D printing technology that is steadily approaching consumer-level viability?

A world of possibilities is waiting to be explored by users through innovative games and applications. **C**

Justin Clark is a software engineer at Good Science Studio at Microsoft. Contact him at juclark@microsoft.com.

Editor: Kelvin Sung, Computing and Software Systems, University of Washington, Bothell; ksung@u.washington.edu



Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.



is seeking

Business Systems Analyst, Sr. Staff

Irvine, CA • Reference Job Code: ENG7-IRCAHS

Req. BS (or foreign equiv.) in Bus. Administration, CIS, Comp. Eng or rel + 7 yrs of post-baccalaureate progressive exp. Responsible for analyzing complex business problems and resolving them using automated systems. Broadcom Corporation, Irvine, CA. F/T. Must have unrestricted U.S. work authorization.

Scientist, Senior Staff – Electronic Design

Chandler, AZ • Reference Job Code: ENG7-CHAZSA

Reqs PhD in EE, CE, or CS and 3 yrs exp. Architect, design and integrate digital IPs. Professional experience or academic knowledge must include: Microarchitecture design; C/C++ algorithms; Verilog/System Verilog; EDA tools for front-end design; DFT (Design for Test) methodologies. Up to 10% of domestic travel time is required.

Mail resumes to:
HR Operations Coordinator
5300 California Ave.
Bldg. 2, #22108B
Irvine, CA 92617



is seeking a

Engineer, Staff-Software

Chandler, AZ

Req. MS (or foreign equiv.) in Electrical Engg, CS, or Computer Engg. Conduct unit testing while ensuring compliance to standards. Up to 5% domestic travel req. F/T. Must have unrestricted U.S. work authorization.

Mail resumes to:
HR Operations Coordinator
5300 California Ave.
Bldg. 2, #22108B
Irvine, CA 92617
Must reference
job code ENG7-AZKK

INVISIBLE COMPUTING

From Data Analysis and Visualization to Causality Discovery

Min Chen, Anne Trefethen, René Bañares-Alcántara,
Marina Jirotko, and Bob Coecke
University of Oxford, UK

Thomas Ertl and Albrecht Schmidt
University of Stuttgart, Germany



As data becomes invisible, emerging technologies can help human analysts and decision makers understand, model, and visualize causal relationships.

Nowadays the public, and even most decision makers, rarely see raw data. People are used to statistical results and visualized data presented by scientists, data analysts, or news readers. As computers disappear into the background, raw data is also becoming invisible.

The reliance on data analysis and visualization to improve the usability of information and communications technology echoes Mark Weiser's vision of ubiquitous computing: "... only when things disappear in this way are we freed to use them without thinking and so to focus beyond them on new goals" ("The Computer for the 21st Century," *Scientific Am.*, Sept. 1991, pp. 94-104).

In general, the aim of data analysis and visualization is to help identify the causes of observed events. Integrating emerging technologies can facilitate causality discovery in numerous endeavors, including the sciences, engineering, medicine, the humanities, industry, business, and governance.

CAUSALITY REASONING

Causality is the fabric of our dynamic world. We all frequently attempt to reason about the causes of everyday events—for example, why do I have a headache, or what has upset Alice?—in order to better manage them. We also seek to understand the origins of numerous complex phenomena such as social divisions, economic crises, global warming, and terrorism. Some of the greatest scientific discoveries, from Newton's laws of motion to Darwin's theory of natural selection, involve causality.

Causality has been studied in the natural sciences, philosophy, probability and statistics, and computer science. Max Born, winner of the 1954 Nobel Prize in physics, observed that "chance has become the primary notion, mechanics an expression of its quantitative laws, and the overwhelming evidence of causality with all its attributes in the realm of ordinary experience is satisfactorily explained by the statistical laws of large numbers"

(*Natural Philosophy of Cause and Chance*, 1949).

Figure 1 illustrates Born's four levels of causality reasoning and the transitions between them. *Probabilistic causation* is a form of preliminary reasoning based on statistical correlation. "Overwhelming evidence" abstracts probabilistic causation to *logical causation*. When there is a scientific understanding of causality, *quantitative laws* of nature describe the functional relationship between measurable attributes of various events. According to Born, quantum mechanics offers the *fundamental understanding* of causation in physics. However, new scientific discoveries will continually redefine what is fundamental.

Humans analyze causality through observation, experimentation, and a priori knowledge. Today's technologies enable us to make observations and carry out experiments on an unprecedented scale, resulting in a deluge of data. This offers exciting opportunities to discover new causation relationships, but managing

such data also presents unparalleled challenges.

CAUSALITY DISCOVERY TECHNOLOGIES

Numerous technologies including visual analytics, data repositories and grids, computer-assisted workflow and process management, and quantum computing are improving the process of causality discovery. Figure 2 shows how these technologies could provide decision support in a typical organization and aid hypothesis generation and evaluation in a scientific investigation.

Visual analytics

Visual analytics (VA), a term coined by Jim Thomas and his colleagues at the National Visualization and Analytics Center (*Illuminating the Path: The Research and Development Agenda for Visual Analytics*, IEEE CS Press, 2005), has become the de facto standard process for integrating data analysis, visualization, and interaction to better understand complex systems.

VA rests on the following assertions:

- statistical methods alone cannot convey an adequate amount of information for humans to make informed decisions—hence the need for visualization;
- algorithms alone cannot encode an adequate amount of human knowledge about relevant concepts, facts, and contexts—hence the need for interaction;
- visualization alone cannot effectively manage levels of details about the data or prioritize different information in the data—hence the need for analysis and interaction; and
- direct interaction with data alone isn't scalable to the amount of data available—hence the need for analysis and visualization.

These four assertions apply to any causality discovery process

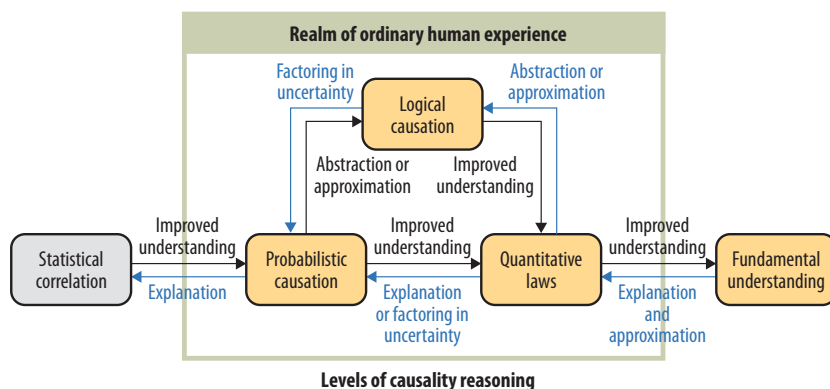


Figure 1. Max Born's four levels of causality reasoning.

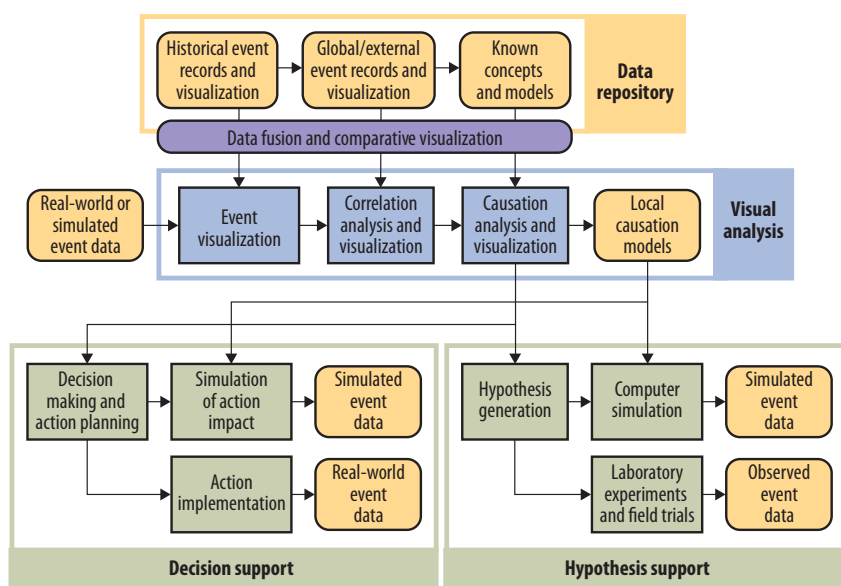


Figure 2. Workflow for causality discovery to provide decision support and aid hypothesis generation and evaluation.

for complex phenomena and environments.

Researchers have demonstrated the usefulness of VA in causality reasoning in several case studies, such as linking readability to the acceptance of research papers (D. Oelke et al., "Visual Readability Analysis: How to Make Your Writing Easier to Read," *Proc. 2010 IEEE Conf. Visual Analytics Science and Technology* [VAST 10], IEEE Press, 2010, pp. 123-130) and formulating cost-effective decision trees for facial expression classification (G.K.L. Tam et al., "Visualization of Time-Series Data in Parameter Space for

Understanding Facial Dynamics," *Computer Graphics Forum*, June 2011, pp. 901-910).

In recent years, many applications, including risk management, fault diagnosis, geohistorical sense-making, document analysis, financial planning, and decision formulation have exploited VA's potential for identifying probable causal factors. Figure 3 shows an example of a visual analytics tools developed by researchers at the University of Stuttgart for the IEEE VAST 2011 Challenge (<http://hcil.cs.umd.edu/localphp/hcil/vast11>). The tool identifies the ground zero location and affected regions of

INVISIBLE COMPUTING

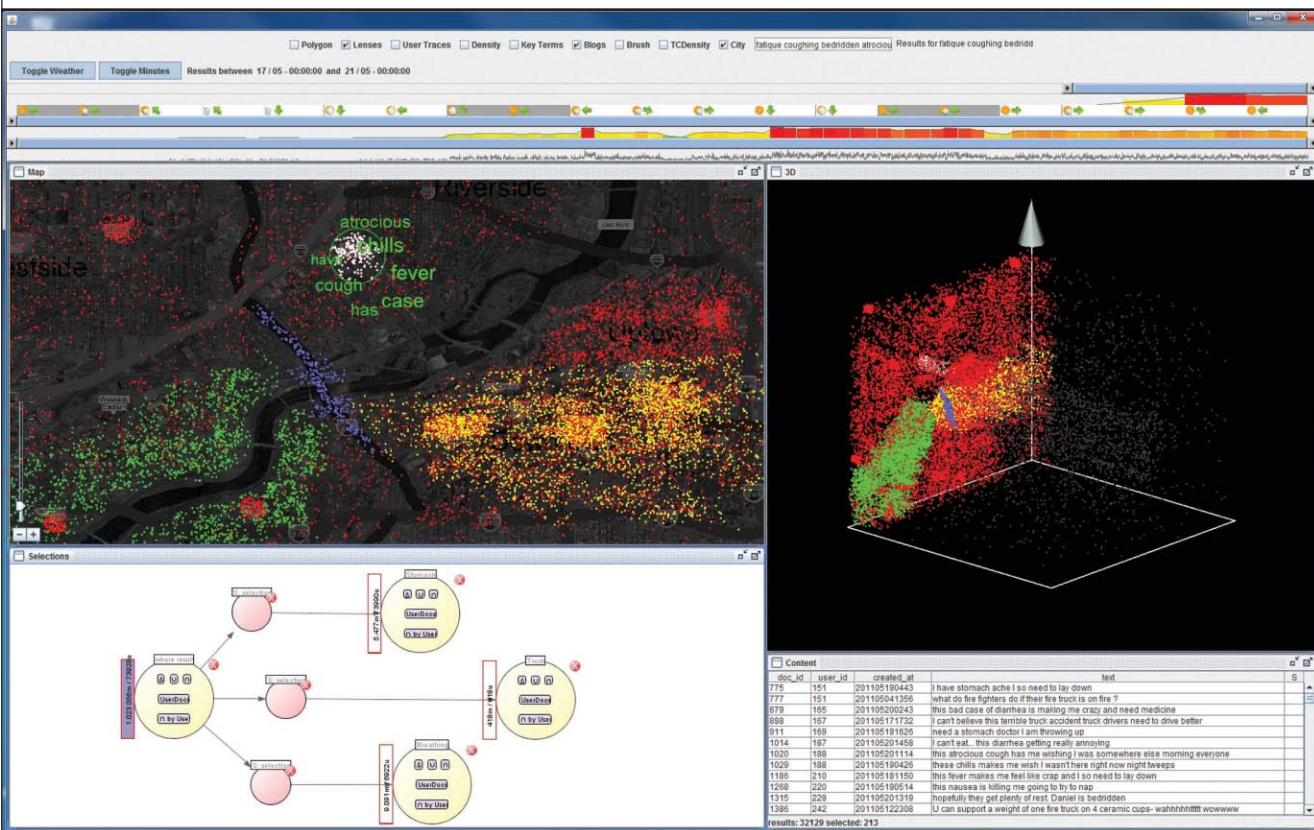


Figure 3. Visual analytics tool developed by researchers at the University of Stuttgart for the IEEE VAST 2011 Challenge. The tool identifies the ground zero location and affected regions of a simulated epidemic outbreak from multiple data sources, including about one million fictive microblog messages and relevant GPS data, map information, and satellite images.

a simulated epidemic outbreak from multiple data sources, including about one million fictive microblog messages and relevant GPS data, map information, and satellite images.

Data repositories and grids

The availability of huge amounts of data presents new opportunities for researchers in all types of fields to observe complex events and phenomena and to discover hidden relationships. Data-intensive computing has arguably become the new paradigm for scientific research (T. Hey, S. Tansley, and K. Tolle, eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, 2009), and advances in data repositories and grid technologies now make it possible to seamlessly collect and integrate data from different historical periods, geospatial locations, and sensory devices.

Workflow and process management

Researchers use computer systems and software tools extensively to manage workflows and processes in a broad range of applications including software development, business planning, manufacturing, and financial and human resource management. The distinction between software design and work practice is not as clear as before. In many applications, theoretical frameworks and national or international standards underpin workflow and process management.

Quantum computing

While quantum computing is still in its infancy, recent advances in the theoretical understanding of quantum computation could help researchers construct a more sophisticated and coherent theoretical framework for causality reasoning.

According to the four levels of causality reasoning illustrated in Figure 1, quantum physics (of a probabilistic nature) provides explanations for most quantitative laws in physical sciences. In quantum computing, it has been shown that a universal computer (of a logical nature) can be simulated on a universal quantum computer (of a probabilistic nature). Because probabilistic causation is also of a probabilistic nature, the theoretical concepts in quantum computing can potentially be applied to the transition from probabilistic causation to quantitative laws.

CHALLENGES

The development of causality discovery technology also faces many challenges.

First, organizations will need an *infrastructure* to provide the capability for data archiving, causality

analysis and visualization, and workflow and knowledge management. Creating such an infrastructure and training personnel to properly use and maintain it could be expensive.

In addition, *visualization literacy* must be dramatically improved. Most widely used visualization techniques were developed long before computers—for example, time-series plots were introduced about 1,100 years ago and pie charts about 300 years ago. Many researchers are unfamiliar with advanced, computer-based visualization techniques such as parallel coordinates and tree maps.

Whereas computer scientists have made great strides in computation scalability, *interaction scalability* presents a more serious challenge. Reducing the burden of interacting with a continual data deluge is an essential feature of causality discovery technology.

Scientific models typically omit some apparently minor causes and effects, but when applied to real-world situations these omissions produce *uncertainty*. There is a need for estimating, visualizing, and propagating uncertainty in the reasoning process as in scientific processes.

Perhaps the most difficult challenge is *transitioning to improved understanding*. In the sciences, such transitions often result in the most rewarding discoveries. While there is no magic recipe for making significant leaps in causality discovery, researchers could achieve incremental improvements by systematically reducing uncertainty, enumerating and evaluating hypotheses, and prompting users with the most promising results to stimulate innovative and creative thinking.

Today, scientific communities and businesses face mountains of often conflicting data that they must act upon quickly. Neither individual organizations nor society in general

can continue to rely on a few geniuses to make important discoveries about causal relationships.

While it's not technically feasible—at least within the next few decades—or even desirable for computers to analyze data and make important decisions on their own, it is possible to develop techniques and tools that help human analysts and decision makers understand, model, and visualize causal relationships. Causality discovery technology has the potential to fundamentally change the way in which scientists, engineers, doctors, school teachers, company managers, government officials, law enforcement personnel, and members of other professions discover and exploit these relationships. **□**

Min Chen is a professor of scientific visualization at the Oxford e-Research Centre, University of Oxford, UK. Contact him at min.chen@oerc.ox.ac.uk.

Anne Trefethen is director of the Oxford e-Research Centre, University of Oxford. Contact her at anne.trefethen@oerc.ox.ac.uk.

René Bañares-Alcántara is a lecturer in the Department of Engineering Science at the University of Oxford.

Contact him at rene.banares@eng.ox.ac.uk.

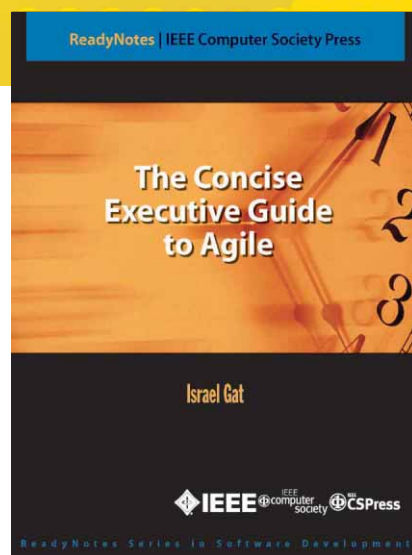
Marina Jirotko is a reader in requirements engineering, director of the Centre for Requirements Engineering, associate director of the Oxford e-Research Centre, and associate researcher of the Oxford Internet Institute, University of Oxford. Contact her at marina.jirotko@oerc.ox.ac.uk.

Bob Coecke is a professor of quantum foundations, logics, and structures in the Department of Computer Science, University of Oxford. Contact him at coecke@cs.ox.ac.uk.

Thomas Ertl is a professor of computer science at the University of Stuttgart, where he also heads the Institute for Visualization and Interactive Systems and the Visualization Research Center. Contact him at thomas.ertl@vis.uni-stuttgart.de.

Albrecht Schmidt, Invisible Computing column editor, heads the Human-Computer Interaction group in the Institute for Visualization and Interactive Systems at the University of Stuttgart. Contact him at albrecht@computer.org.

Editor: Albrecht Schmidt, University of Stuttgart, Germany; albrecht@computer.org



NEW from  **CSPress**

THE CONCISE EXECUTIVE GUIDE TO AGILE

by Israel Gat

Get the tools and principles you need to lead an Agile transformation at your organization in this short and practical handbook, delivered digitally right when you need it.

PDF edition • \$15 list / \$12 members • 21 pp.

Order Online:
COMPUTER.ORG/STORE

SOFTWARE TECHNOLOGIES

Guidelines for Software Development Effort Estimation

Dirk Basten and Ali Sunyaev
University of Cologne, Germany



An analysis of 32 accuracy factors yields a list of useful guidelines for improving software development effort estimation accuracy.

Inaccurate development effort estimation is one of the main problems that impact software projects. On average, the effort required to complete a project deviates 30 percent from the initial projection. Software experts don't expect such estimates to be absolutely accurate, but the consequences of inaccurate predictions can be severe and even fatal.

For example, underestimating the development effort can lead to understaffing, cost and schedule overruns, and poor-quality software, while overestimates can result in inefficiency and wasted resources, as projects tend to expand to fit the estimated effort.

Although previous studies analyzing the reasons for estimation inaccuracy provide helpful insights, developers would greatly benefit from guidelines that simultaneously address the numerous potential influences on estimation accuracy.

Toward that end, we surveyed more than 50 empirical studies published since 1990 to identify factors generally affecting the accu-

racy of software development effort estimates. We didn't consider specific estimation techniques, nor did we explore the reasons for deviations between estimates and actual results—for example, management removal of padding.

We first categorized factors affecting accuracy according to their role in development effort estimation in the project life cycle. These categories are the *process*, *estimator*, *project*, and *context*.

We then identified 32 factors, listed in Figure 1, that potentially improve or impede estimation accuracy—for example, use of historical data and lying, respectively. Not all of these factors can be controlled, at least in the short term. Nevertheless, knowledge about their influence will help professionals be aware of their impact.

Based on these factors, we developed a set of practical guidelines for each category.

PROCESS

Estimation process factors guide estimators' work on both single-proj-

ect and cross-project issues and thus should be addressed first.

To enable higher estimation accuracy

- ensure that accuracy is part of the estimator's evaluation—such pressure will lead to more carefully conducted estimates;
- collect as much information about estimation tasks as possible—analogs with similar projects are helpful; and
- include adequate feedback sessions in the estimation process to improve the estimator's performance.

ESTIMATOR

Estimators have different knowledge, skills, and behavior. These factors can't be controlled through the estimation process.

To select an adequate estimator

- ensure that the estimator has the proper technical and managerial expertise to understand the task and its context;

Process	Project
Accountability Assessment of developer skills and task complexity Careful examination of the estimate Estimate alteration Estimation development Estimation management Estimation strategy (top down versus bottom up) Historical data Initial risk analysis Learning Request format	Anchor effect Changes in personnel Clear project goals Development process Level of innovation Lying Neutrality and relevance of information Overlooked tasks Requirements specification Task complexity
Estimator	Context
Diligence Experience General degree of optimism Reference to the project Role Inconsistency of estimates	Client maturity Client priority Collaboration/communication Contract type (fixed price versus pay per hour) Project priority

Figure 1. Factors affecting software development effort estimates.

- appoint professionals to estimate the task they develop themselves—both the estimate and the development will benefit from this choice;
- select an estimator who is willing to learn—otherwise, feedback sessions will have less value; and
- try to exclude candidates with a high general degree of optimism as well as optimism in the estimation process.

PROJECT

Project factors are inherent in the development task, such as complexity, or surround the estimation process, such as lying. Neither the design of the estimation process nor the estimator can influence these factors.

To enable the highest possible accuracy

- ensure requirements are as precise as possible to make project goals clearer—uncertainty, overlooked tasks, and irrelevant and misleading information will all diminish estimation accuracy;
- avoid anchors—estimators will unconsciously focus on an

anchor and only subsequently alter the estimate; and

- split larger tasks into smaller ones, which can usually be estimated more accurately.

CONTEXT

It's difficult to control external factors related to or determined by the client, such as the bidding process or the contract type.

With respect to the project environment

- communicate frequently with the client to obtain direct feedback;
- clearly state why the project is important—a high priority will influence the client to exert more diligence; and
- use your expert knowledge to support inexperienced clients and improve the accuracy of the requirements and thus the estimates.

IMPLICATIONS AND LIMITATIONS

Some estimation accuracy factors can only be controlled to a limited degree—for example, one party can't dictate the contract type. Accordingly,

simple knowledge of these factors isn't sufficient to guarantee accurate estimates.

Nevertheless, our guidelines will

- sensitize software professionals to be cautious regarding diverse influences on the estimation process and thus have greater awareness of surrounding problems;
- serve as a starting point to develop checklists, thereby systematically eliminating reasons for inaccuracy;
- guide software professionals to a goal-oriented improvement of estimation accuracy—for example, by refining the estimation process using available resources if no experienced estimators are available; and
- improve the estimation process by controlling the various influences.

Because our guidelines are based on empirical studies, they can't be generalized to all software development projects. Practitioners should always determine what specific factors they need to control in their own case. Moreover, it isn't feasible

SOFTWARE TECHNOLOGIES

to control all factors on a project: the necessary resources probably aren't available. Therefore, it's important to perform a cost-benefit analysis of each potential controllable factor.

The findings we used to derive our guidelines weren't distinct in every case. Each company should therefore

review its own context and use our list of factors as a wrapper for project-specific checklists. For example, distinguishing between fixed-price and pay-per-hour contracts isn't meaningful if a company usually executes the former. In such cases, it's more reasonable to think about

contract design within the context of fixed-price projects.

There seem to be no objective criteria for ordering the accuracy factors in a prioritized list. However, we believe that estimator experience is one of the most important factors in achieving development effort estimation accuracy. Expert estimation is the most often-used technique, and even the creation of meaningful estimation models relies on expert input. At the same time, it's impractical to appoint the highest-qualified estimator on every project.

Although our research-driven guidelines can improve the software development estimation process and its result, users shouldn't expect highly accurate estimates solely using them. Many companies have their own guidelines and shouldn't ignore their knowledge and experience. In those cases, our guidelines should supplement existing processes. ■

Dirk Basten is a postdoctoral researcher in the Department of Information Systems and Systems Development at the University of Cologne, Germany. Contact him at basten@wiso.uni-koeln.de.

Ali Sunyaev is an assistant professor in the Department of Information Systems and Systems Development at the University of Cologne. Contact him at sunyaev@wiso.uni-koeln.de.

Editor: Mike Hinchey, Lero—The Irish Software Engineering Research Centre;
mike.hinchey@lero.ie

cn Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.



ALLIANCE
DATA SYSTEMS

ADS Alliance Data Systems, Inc has positions in the following locations:

Irving, TX

DATABASE DEVELOPER

Perform tasks related to the full life-cycle development of software for internal & client use; plan, analyze requirements, design solutions, develop code & testing the software; use Oracle10g to develop PL/SQL packages, Stored Procedures/Functions, SQL scripts, SQL*Loader & Unix shell scripts; & other duties/skills required. [Job ID# AD-TX11F-DDEV]

SENIOR DATABASE ADMINISTRATOR

Responsible for database backups, recoveries, patches, upgrades, design, development, testing, & performance tuning; work with advanced Shell script, Perl scripting, advanced Oracle SQL programming, Oracle RMAN, Toad; & other duties/skills required. [Job ID# AD-TX-SRDAD]

Wakefield, MA

SENIOR DATABASE DEVELOPER

Exp. in development and/or administrator exp. on DataStage PX; one or more Database/RDBMS; & other duties/skills required. [Job ID# AD-WK11J-SRDD]

INTERACTIVE DEVELOPER

Provide production support to include working with Microsoft .NET programming (C#); web application & web service development; SQL programming (Oracle or SQL Server); & Object Oriented Design & design patterns skills; & database development; & other duties/skills required. [AD-WK11J-ID]

Lafayette, CO

SENIOR DATABASE DEVELOPER

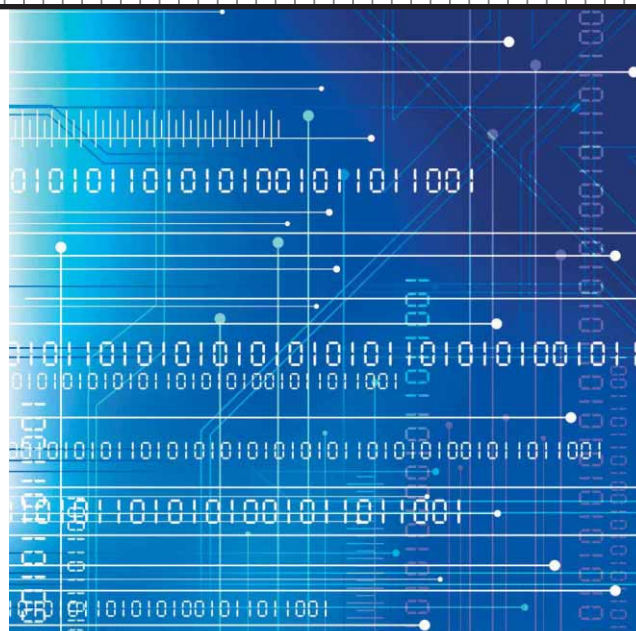
Develop & support complex customized marketing & loyalty database solutions using PL/SQL, .NET & other ELT/ETL tools; work with ETL in OLTP & data warehouse environments; RDBMS concepts; & other duties/skills required. [Job ID#: AD-CO11-SRDD]

Mail resume to Attn: S. Resler-HR Coordinator, Alliance Data, 601 Edgewater Dr., Wakefield, MA 01880 & note the Job ID #.

DISCOVERY ANALYTICS

Twitter Mood as a Stock Market Predictor

Johan Bollen and Huina Mao
Indiana University Bloomington



Behavioral finance researchers can apply computational methods to large-scale social media data to better understand and predict markets.

It has often been said that stock markets are driven by “fear and greed”—that is, by psychological as well as financial factors. The tremendous volatility of stock markets across the globe in recent years underscores the need to better understand the role that emotions play in shaping stock prices and other economic indices.

A stock market is a large-scale, complex information processing system that responds to a wide variety of socioeconomic factors. As such, its behavior is difficult to model and, consequently, to predict. The *efficient market hypothesis* (EMH) asserts that financial markets are “informationally efficient.” In other words, market prices tend to fully reflect all available information because investors rationally take this into account and act accordingly. This implies, among other things, that it’s impossible to achieve better-than-average market returns over long terms.

Behavioral finance has challenged this notion, asserting that investors don’t always act on the basis

of rational considerations and that their behavior is subject to particular psychological biases and emotions. Consequently, predicting market behavior requires understanding the factors that shape investors’ individual as well as collective behavior.

PREDICTING MARKET BEHAVIOR

Behavioral finance and investor sentiment theory have firmly established that investors’ behavior can be shaped by whether they feel optimistic (bullish) or pessimistic (bearish) about future market values. Theories of social mood and their role in market behavior, such as those proposed as early as the 1970s by Robert Prechter (“What’s Going On?,” *The Elliott Wave Theorist*, 3 Aug. 1979), are also enjoying increasing acceptance.

The recent financial crisis has reinforced the notion that investor sentiment shapes financial markets more than the EMH would allow. Consequently, the question is no longer whether social mood and investor

sentiment affect stock prices, as it was a few decades ago, but rather how we can best measure and model their effects.

Historically, surveys have been the most direct way to measure social mood and investor sentiment. For example, the Conference Board’s Consumer Confidence Index, the University of Michigan’s Consumer Sentiment Index, and Gallup’s Economic Confidence Index measure public sentiment about current and future economic trends. In addition, numerous investment services and organizations including Merrill Lynch, Investors Intelligence, and the American Association of Individual Investors conduct and publish investor sentiment polls.

Such surveys, however, have serious inherent disadvantages. Conducting them is expensive, thus limiting both the number of individuals they can cover and how often they can be repeated in a given time interval. Furthermore, explicit expressions of mood or sentiment could be less reliable than behavior-based indicators.

DISCOVERY ANALYTICS

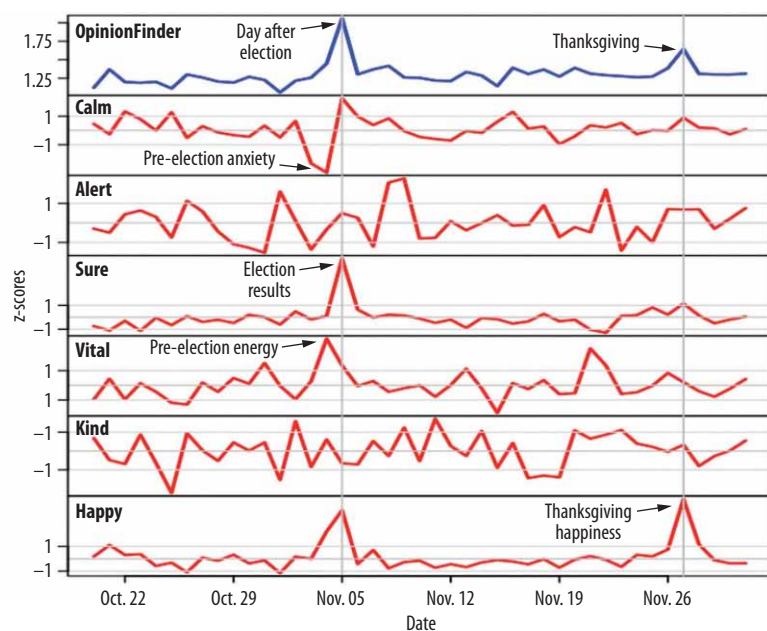


Figure 1. Fluctuation of public mood states from October to December 2008 as measured from large-scale Twitter data by GPOMS and OpinionFinder. Key events—the US presidential election and Thanksgiving—had an effect on public mood that the time series easily picked up.

The rise of social networking environments has opened a new frontier in the development of reliable, scalable, and rapid assessments of the public mood. Recent research efforts led by Alan Mislove (“Pulse of the Nation: U.S. Mood throughout the Day, as Inferred from Twitter,” www.ccs.neu.edu/home/amislove/twittermood) and Peter Dodds (“Temporal Patterns of Happiness and Information in a Global Social Network: Hedonometrics and Twitters,” <http://arxiv.org/abs/1101.5120>) have shown that analysis of geolocated and time-stamped tweets can lead to geographical models of changes in the public mood over time.

In our work, (J. Bollen, H. Mao, and X.-J. Zeng, “Twitter Mood Predicts the Stock Market,” *J. Computational Science*, Mar. 2011, pp. 1-8), we have leveraged sophisticated sentiment-tracking tools and large-scale Twitter data to assess global public mood states at short time intervals. We then used the resulting Twitter mood data to predict stock market values, in

particular the daily fluctuations of the Dow Jones Industrial Average (DJIA), with surprising results.

DATA COLLECTION AND VALIDATION

Our initial analysis in 2010 focused on the year 2008, which was marked by a US presidential election as well as the near-collapse of the world’s financial system, with a corresponding crash of global stock markets.

We obtained a collection of 9,853,498 public tweets posted by approximately 2.7 million users from 28 February to 19 December 2008. Each tweet record contained an anonymous identifier, the submission’s date-time stamp, the mechanism for submitting the tweet (mobile phone, Web, and so on), and the tweet’s content. As most Twitter users know, tweets are limited to only 140 characters—hence the term microblogging. After removing stop words and punctuation, we grouped all tweets submitted on the same date to generate daily time series. We made no

efforts to limit tweets to a particular geographical location or time zone.

We applied two emotion analysis tools to our daily collections of tweets:

- GPOMS, a now proprietary tool that measures six different dimensions of mood often ignored by traditional sentiment-tracking methods—calm, alert, sure, vital, kind, and happy; and
- OpinionFinder (OF; www.cs.pitt.edu/mpqa/opinionfinderrelease), a widely used sentiment-analysis tool that classifies texts in terms of their positive versus negative sentiment and is useful for cross-validation.

For each of the six GPOMS dimensions and OF analysis, we generated a daily time series of public mood fluctuations.

To qualitatively assess the time series’ face validity, we examined mood fluctuations in the two-month period from 5 October 2008 to 5 December 2008, during which the presidential election (4 November) and Thanksgiving (27 November) occurred. Our assumption was that these events had a variegated effect on the public mood that the series should easily pick up.

As Figure 1 shows, the GPOMS time-series data revealed increased nervousness before the election, high energy the day of the election, and public elation as well as assurance the day after the election. The OF results confirm a high degree of positive sentiment the day after the election, but due to its binary nature, the data doesn’t capture any other public mood changes. All seven time series indicate a coherent response to Thanksgiving: happy or positive, but not much more.

A statistical analysis further showed that the OF time series is significantly correlated with several GPOMS time series—namely, sure, vital, and happy—but not with calm, alert, and kind. Because the latter

dimensions might nevertheless play an important role in shaping public mood, we retained them in subsequent analyses.

DATA ANALYSIS

To determine whether public sentiment relates to stock market values, we first applied a Granger causality analysis to our seven public mood daily time series as well as a daily time series of DJIA closing values in the same period.

We then deployed a self-organizing fuzzy neural network (SOFNN) model to test the hypothesis that including public mood measurements can improve the accuracy of DJIA prediction models—in other words, we used the SOFNN model as a diagnostic tool to determine whether our public mood daily time series contained predictive information with respect to DJIA closing values. We chose 28 February to 28 November as the training period and 1 to 19 December 2008 as the test period.

Figure 2 illustrates the methodology and timeline of our approach.

As Figure 3 shows, public mood and DJIA daily time series frequently overlapped in the same direction. Calm had the highest Granger causality relation with the DJIA for time lags ranging from two to six days (p -values < 0.05). Changes in calm values on day $(t - 3)$ predicted a similar rise or fall in DJIA values (t) , indicating the predictive power of calm with regard to DJIA. Compared with the baseline model of historical DJIA values, adding calm reduced the mean absolute percentage error (MAPE) by 6.15 percent (1.94 to 1.83 percent) and improved direction accuracy by 18.3 percent (73.3 to 86.7 percent). The other four GPOMS mood dimensions and OpinionFinder didn't have a significant correlation with stock market changes.

In sum, public mood as measured along GPOMS dimensions, especially calm, matched shifts in DJIA values that occurred three to four days

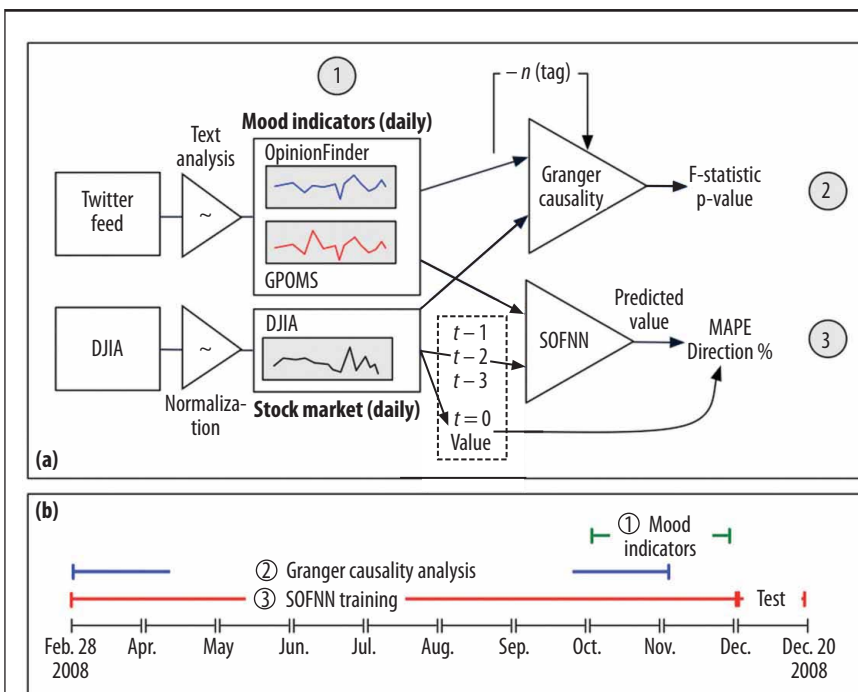


Figure 2. Assessing the correlation between public mood daily time series data extracted from Twitter and Dow Jones Industrial Average (DJIA) closing values: (a) methodology and (b) timeline.

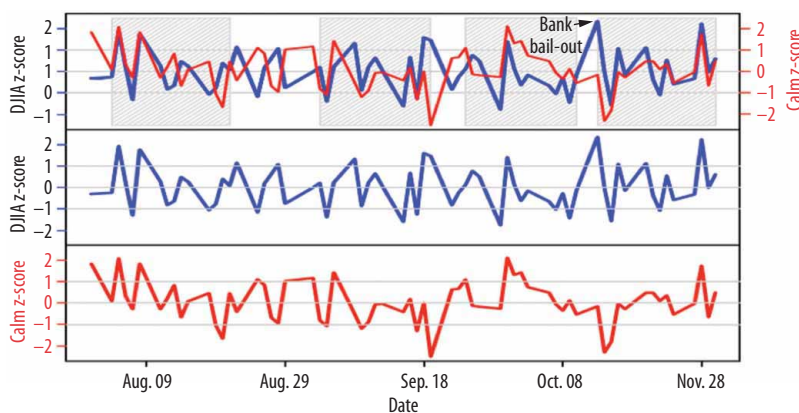


Figure 3. Daily time series of public mood overlaid with those of DJIA closing values. The latter are shifted back three days to highlight where public mood changes predict the DJIA values.

later. This result is consistent with a basic assumption of behavioral finance—emotions play a key role in financial markets.

Behavioral finance as a research field is still in its infancy, but it will increasingly become part of mainstream

finance. Exploiting emerging technologies, researchers can apply computational methods to social-media data to model behavior in financial markets more accurately and on a scale well beyond the limits of traditional controlled experiments.

Our analysis suggests some interesting directions for future research.

DISCOVERY ANALYTICS

First, we only studied the relationship between Twitter mood and the US stock market. Twitter offers a potentially rich source to track public sentiment in other countries—including the UK, Portugal, Japan, and Spain (English, Portuguese, Japanese, and Spanish are the most popular languages on Twitter)—to investigate the impact of mood on their stock markets.

Second, while our results strongly indicate a predictive correlation between Twitter mood and DJIA values, they offer no information on the causative mechanisms. This

remains a crucial area for future research.

Third, researchers might need to take into account social and cognitive effects in which individual agents are endowed with the ability to learn from past experiences and can adjust their trading behavior accordingly.

Finally, there must be more study of financial markets at the aggregate level.

Explaining which stocks are likely to be most affected by sentiment is also an interesting research question. **□**

Johan Bollen is an associate professor in the School of Informatics and Computing at Indiana University Bloomington. Contact him at jbollen@indiana.edu.

Huina Mao is a PhD student in the School of Informatics and Computing at Indiana University Bloomington. Contact her at huinmao@indiana.edu.

Editor: Naren Ramakrishnan, Dept. of Computer Science, Virginia Tech, Blacksburg, VA; naren@cs.vt.edu



is seeking a

Engineer, Sr. Staff- Software Systems

Chandler, AZ

Req. BS (or foreign equiv.) in CS, Computer Engg, or rel. Provide necessary tech support to internal/external customers for the applications involving USB device & host controllers. May req up to 10% domestic travel. F/T. Must have unrestricted U.S. work authorization.

Mail resumes to:
HR Operations Coordinator
5300 California Ave.
Bldg. 2, #22108B
Irvine, CA 92617
Must reference
job code ENG7-AZAP.



is seeking a

Sr. Staff Engineer - IC Design

Irvine, CA

Req. MS (or foreign equiv.) in Electrical Engg. Run functional vectors on RTL and gates to identify areas for improvement of power dissipation. Up to 5% domestic travel time req. F/T. Must have unrestricted U.S. work authorization.

Mail resumes to:
HR Operations Coordinator
5300 California Ave.
Bldg. 2, #22108B
Irvine, CA 92617
Must reference
job code ENG7-IRCAHJ.



is seeking a

Engineer, Staff II- Electronic Design

Tempe, AZ

Req. MS (or foreign equiv.) in Electrical or Computer Engg. Develop test plans and test cases to meet functional and code coverage requirements. May req. up to 10% domestic travel. F/T. Must have unrestricted U.S. work authorization.

Mail resumes to:
HR Operations Coordinator
5300 California Ave.
Bldg. 2, #22108B
Irvine, CA 92617
Must reference
job code ENG7-AZGK.

EDUCATION

Learning How to Prepare Computer Science High School Teachers

Mark Guzdial, *Georgia Institute of Technology*



In addition to more well-trained high school computer science teachers, we need more education research that is informed by understanding how CS is taught, what the current practices are, and what's important to keep as practices change.

Few US secondary school students have access to high-quality computer science education. There are only about 2,000 CS teachers at the Advanced Placement level for the more than 25,000 high schools in the US, and test attendance figures dramatically reflect this shortage. In 2010, 14,517 students took the AP Computer Science Level A exam, while 194,784 took the Calculus AB exam and 134,871 took the Biology exam (http://apreport.collegeboard.org/sites/default/files/downloads/pdfs/AP_RTN_2011.pdf).

According to a recent report by the Computer Science Teachers Association, less than 35 percent of CS learning objectives are part of the state curricula in high schools in the 50 US states (<http://csta.acm.org/runningonempty/roemap.html>). Some states do better than others, but in most cases, students are unlikely to find any CS curriculum in their high schools.

That's a problem for all of us. High school students who don't study CS hold negative stereotypes about the

field. Those stereotypes impact CS enrollment and hamper efforts to educate the public in understanding and valuing CS.

The US National Science Foundation (NSF) has set a "CS10K" goal of having 10,000 high school teachers ready to teach CS at the Advanced Placement level in 10,000 schools by 2015 (J. Cuny, "Transforming Computer Science Education in High Schools," *Computer*, June 2011, pp. 107-109). Where are we going to get all those teachers? How are we going to prepare them to teach CS?

My Georgia Tech colleagues and I work in computing education research, an interdisciplinary field that studies how we educate about CS and how we can improve that education. Our investigations have yielded some interesting insights that inform the CS10K goal.

OPERATION REBOOT

One solution to educating potential CS teachers is to start with people who already know about the discipline. One such program is Operation Reboot (<http://coweb.cc.gatech.edu/>

ice-gt/1077), which was established by Barbara Ericson, director of Computing Outreach at Georgia Tech, to help IT workers left unemployed by the recession to become high school CS teachers. Ericson is currently working with her third cohort of Operation Reboot teaching candidates.

In this program, the former IT workers start by working on professional development during the summer. They then go into the classroom in the fall, team-teaching with a business teacher. (In Georgia, as in most US states, CS is classified under career and technical education, and is often taught by business teachers.) An IT worker who needs to learn how to run a classroom partners with a business teacher who knows how to run a classroom and wants to learn more about CS. These partners learn from one another in a collaboration that results in two well-prepared high school CS teachers. A model high school CS teacher receives a stipend for mentoring the pair.

Operation Reboot has successfully produced new, well-prepared CS teachers. The mentoring model has

EDUCATION

been especially successful. Because the model teachers are rarely in the same district as the team-teaching pair, the cross-district mentoring serves to spread good ideas on how to teach CS.

Unfortunately, few Operation Reboot teachers are in the classroom today. The recession that displaced the IT workers has also impacted secondary school budgets, and CS is not a priority in high schools. Overall in Georgia, the number of positions available for high school CS teachers has decreased. When there is a CS opening, recently unemployed, experienced teachers receive preference in filling that position rather

(www.disciplinarycommons.org), a professional development activity. Developed by Josh Tenenberg and Sally Fincher, a Disciplinary Commons leads a group of teachers through an academic year in a process of self-reflection on what and how they teach.

Our Disciplinary Commons for Computing Education (DCCE; <http://home.cc.gatech.edu/dcce>) was led by two master teachers, Ria Galanos, a high school teacher, and Briana Morrison, a university faculty member. Ni interviewed and observed a group of high school teachers as they participated in the DCCE experience over the course of a year. Although they taught

that encouraged them to learn how to be better CS teachers.

TEACHING TEACHERS ONLINE

A possible source for finding potential CS teachers is by helping existing high school teachers retrain, much like the business teachers in Operation Reboot. How are we going to teach all those educators the CS they need for CS10K to succeed? One possible answer is with online courses.

Columbus State University in Columbus, Georgia, recently initiated an effort to educate high school teachers by having them take online courses originally developed for an MS in computer science program (<http://cs.columbusstate.edu/endorsement/index.php>). Klara Benda, another HCC PhD student, interviewed students currently taking these online courses and analyzed the results with my colleague, Amy Bruckman. These students were full-time professionals by day. She wanted to figure out what the issues were in learning CS in an online setting by students who are themselves full-time workers, before the existing, full-time high school teachers arrived.

Benda found that the online CS courses mirrored the pedagogy of face-to-face CS classes. The online classes offer lectures and book reading assignments, then expect students to work their way through novel programming challenges, interacting with some development environment. The focus on learning through practice in authentic settings that is characteristic of traditional CS pedagogy might make sense when the goal is to provide a kind of apprenticeship for students learning to become software developers. But is this a practical approach for full-time workers and part-time students who want to learn key CS concepts?

As one of Benda's interviewees said, "There were times that it would take me hours to find one comma out of place or find that one something

Computing education must change for different audiences, including future high school CS teachers.

than Operation Reboot trainees. Most candidate-teachers from Operation Reboot are actually finding employment in the IT industry, where there are now more jobs and the pay is better.

DEVELOPING CS TEACHER IDENTITY

Education researchers who study teachers know how important it is to develop a sense of identity in a specialty—for example, being a science teacher or a reading teacher. An educator who self-identifies as a "science teacher" is more likely to be retained, seek professional development opportunities, and join a community of fellow science teachers. Most teachers' sense of identity comes from their certification—for example, as a "mathematics teacher." Unfortunately, few US states offer CS certification.

To investigate how high school teachers come to identify themselves as "computer science teachers," Lijun Ni, a PhD student in Georgia Tech's Human-Centered Computing (HCC) program, has been tracking teachers engaged in Disciplinary Commons

CS, the participants didn't necessarily see themselves as CS teachers.

Educators aren't necessarily eager to label themselves as CS teachers, linking their performance evaluations to how well they do in CS and seeking professional development in the discipline. A calculus teacher pursues professional development to learn how to teach mathematics better, but the subject and the notation stay the same. In contrast, CS teachers must keep learning the subject because it changes. For example, they must learn a new language when the Advanced Placement exam replaces the required language, which it last did in 2004.

Ni found that DCCE did help the participants develop a sense of identity as CS teachers, even without certification. She found that participating in a community of teachers was critical. The DCCE participants discovered that others were teaching the same CS content and realized that they had teaching methods to contribute that others valued. What's more, the DCCE participants were inspired by the master teachers who led their group, benefitting from role models

that was wrong. I didn't mind sticking with it, but it got to the point where I just didn't get it." How many full-time professionals will be willing to waste hours trying to find a misplaced semicolon?

We need more high school CS teachers who are well-trained, valued, and part of a supportive community. NSF's CS10K goal is a tremendous challenge, and we frankly don't know how to meet that challenge.

Computing education must change for different audiences, including future high school CS teachers. How will it change? What should it look like?

We need more education research that is informed by understanding CS—how it's taught, what the current practices are, and what's important to keep as we change practice. We need more computing education researchers to help meet the workforce needs in our technology-based society. **■**

The work described here was supported by NSF Broadening Participation in Computing (BPC) grants CNS-0634629 and CNS-0940394, and CISE Pathways to Reinvigorate Undergraduate Education (CPATH) grant No. 0829601. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily

reflect the views of the National Science Foundation.

Mark Guzdial is a professor in the School of Interactive Computing at the Georgia Institute of Technology. Contact him at guzdial@cc.gatech.edu.

Editor: Ann E.K. Sobel, Department of Computer Science and Software Engineering, Miami University; sobelae@muohio.edu

cn Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.

Nokia Inc.

has the following open positions in

Burlington, MA

Senior Systems Engineer

Job ID# NOK-MA11J-SS

Provide support operations to include working with technologies & tools: RedLinux/CentaOS/Solaris system administration/ networking & Perl/Python/Bash scripting/PHP; & other duties/skills required.

Boston, MA

Senior Software Engineer

Job ID# NOK-MA11-BOS

Will work with Symbian, Windows mobile, Maemo, or J2ME platform development; C++ programming, client/server environments, & user exp. (user interface) implementation; relational database exp.; & other duties/skills required.

Mail resume to: Nokia Recruiter
3575 Lone Star Circle, Ste. 434
Ft. Worth, TX
76177 & note Job ID#.



is seeking a

Engineer, Staff- Electronic Design

San Jose, CA

Req. MS (or foreign equiv.) in Electrical Engg. Develop and improve new and existing circuit solutions, circuit simulation and circuit layout. May req. 1-2% domestic travel time. F/T. Must have unrestricted U.S. work authorization.

Mail resumes to:
HR Operations Coordinator
5300 California Ave.
Bldg. 2, #22108B
Irvine, CA 92617
Must reference
job code ENG7-SVCAHM.



i n v e n t

HP Enterprise Services, LLC is accepting resumes for the following positions:

Services Information Developer

Downers Grove, IL
Reference: ESDGPM11
and
Folsom, CA
Reference: ESFOLSID31

Conceptualize, design, develop, unit-test, configure, or implement portions of new or enhanced (upgrades or conversions) business and technical software solutions through application of appropriate standard software development life cycle methodologies and processes.

Mail resume to HP Enterprise Services, LLC, 5400 Legacy Drive, MS H1-6F-61, Plano, TX 75024. Resume must include Ref. #, full name, email address & mailing address. No phone calls please. Must be legally authorized to work in the U.S. without sponsorship. EOE.

THE PROFESSION

Continued from page 100

NEUROMANIA

Neuromania “rests on the belief that human consciousness is identical with activity in the brain.” The “fundamental problem” is that “there is nothing in [the brain’s] material properties to make it probable that it could on its own make other pieces of matter appear to it—let alone make them appear as the complex world in which we live.”

This seems to be equivalent to saying that there’s nothing in a computer’s material properties to make it probable that it could on its own make other computers appear to it. Of course, “on its own” is crucial here—no signal, no perception. Fur-

processor’s needs but physically independent of it.

Consciousness is also like the action of a central processor in respect to its peripheral devices. The CPU’s main storage holds data coming into the system for processing from peripheral devices and their controllers. It also stores processed data for use by peripheral devices and their controllers. The CPU is “unconscious” of what the peripheral devices are doing; it is only aware of the data coming from the devices.

Consciousness is like the action of user programs in respect to the operating system in the background. The operating system provides

host conscious or unconscious thought is dead. The brain is active from well before birth until death. What’s neuromaniacal about this?

DARWINITIS

Darwinitis “confuses the biological evolution of the species with the development of our culture.” To see how easily Darwinitis is contradicted, Tallis asks the reader to “look at the difference between an hour of animal life and an hour of human life.” He sees some resemblance between a cow called Daisy and himself, but there is, in his view, a profound difference. “Humans lead our lives, regulating them by shared and individual narratives, whereas beasts merely live them.”

One of the unfortunate concomitants of modern city life is that people rarely have anything to do with animals other than individual pets. I was fortunate to grow up on a dairy farm and know from personal experience that cows have personalities and experience social interaction. And from what I read, animals like whales and chimpanzees do much more than merely live their lives (tinyurl.com/MACHmpz).

Curiously, I think the cause of Darwinitis, if in fact it exists to any degree, is the failure to understand the significance of digital technology to the development of human culture. Digital technology is based on communicating facts or ideas by combining symbols chosen from a limited set. Many animals are known to signal others in various digital ways, as do bees (tinyurl.com/WkBCmnc). Birds signal by song, and it’s even possible that the spoken language of humans started out as song (The Profession, Sept. 2009).

Communicating by spoken language involves mental activity, as does learning it. Once humanoids evolved a vocal tract that could produce a variety of complex sounds, no further biological evolution was needed. Cultural evolution took over.

For the computing professional, there are several possible analogies to animal consciousness in the functioning of computers.

thermore, the computer’s material properties are not of themselves effective without geometric properties like componentry and circuitry.

The emphasis here is on consciousness, specifically human consciousness. One misgiving I have is the implication that consciousness is either unique to humans or at least distinct from that of other animals. Perhaps, since this implication is clearly false, it was not intended, but the discussion of Darwinitis in Tallis’s article makes me uneasy on this point.

For the computing professional, there are several possible analogies to animal consciousness in the functioning of computers.

Consciousness is like the action of a processor and its cache. The cache holds the data the processor is working on for the moment, and peripheral processes feed needed data from main storage to the cache and send result data back to main storage. The flow of data between cache and main storage is automatic or “unconscious,” influenced by the

automatic support for a user program in feeding to it the data it needs and in disposing of the data it computes.

There’s an important mismatch in these analogies between consciousness and computation and between unconsciousness and automatic support. Unlike on the computational side, there’s no clear distinction between consciousness and unconsciousness. For example, there’s a spectrum of awareness between being wide awake and being fast asleep.

But the idea of consciousness is only part of the story. The basis of Tallis’s condemnation of neuromania is what he sees as the impossibility that the brain “could on its own make other pieces of matter appear to it,” and he is quite right. However, the brain is only on its own in a material way, and that material is continually changing. The incoming perceptions that are the essence of appearance change the brain, which adapts to those perceptions at various time scales.

A computer that doesn’t host a process is dead. A brain that doesn’t

Spoken language developed, and different forms of written language were eventually adopted. Language was the basis for cultural development and for the accumulation of knowledge. Modern digital technology simply extends that.

In summary, Darwinitis seems to me to be a straw man.

BIOLOGISM

If we abandon neuromania and Darwinitis, what then is left of biologyism?

Tallis avers that “so long as we ignore the irreducibly relational aspect of human consciousness ... we shall get hung up on sterile and inappropriate questions as to where it is located, if not in the brain.”

Certainly, consciousness is relational, but this is significant because it properly stresses the social aspect of thought and behavior, not because of any neuroanatomical issues.

Again, there is an analogy with computing systems. Consider the difference between an isolated computer and the World Wide Web. An isolated computer can only do for its user whatever can be produced from data stored within it. In contrast, a computer connected to the Web via the Internet is quite something else again.

A child brought up in isolation has no way to learn social skills such as a language. A newborn infant has a relatively undeveloped brain (tinyurl.com/WkChld). How the child's brain develops, and how far, depends initially on interaction with its parents and siblings, and later with its teachers and schoolmates.

Of course, there are numerous other factors, and the use of modern digital technology can have a great effect on mental development (The Profession, Mar. 2008). Much has been written about the adverse social effects of digital technology, in particular, of social websites. Computing professionals should

be aware of both the positive and negative social effects of their work, and try to avoid the negative.

ROBOTICS

The obverse side of issues about the human brain and digital technology is the mental nature of robots, which are often made in the image of people and are expected to acquire human qualities. An article about robots in a popular magazine highlights the statement that they “are making forays into the treacherous terrain of human mental states and emotions” (tinyurl.com/NGCrRbt).

People who see robots in this light don't properly understand the nature of the human mind. Robots are a product of digital technology. Humans are not.

which, back in the days of typewriter terminals and before networking, could get people emotionally involved in very simple dialogues (tinyurl.com/Eliza).


As Noel Sharkey's essays describe (The Profession, Aug. 2009, pp. 116, 113-115; Aug. 2010, pp. 116, 114-115), many professional issues are involved in the use of robots, and computing professionals should be alert to both misunderstandings and misuse of robots in society at large.

The more pervasive digital technology becomes, the more important it is to give the public a reasonable account of how it works, how it might be used, and how its use might affect us and our society.

Computing professionals should be aware of both the positive and negative social effects of their work, and try to avoid the negative.

Although human society depends on digital technology, including language, the digitality is superposed on a fundamentally analog brain. The action potentials used to pass signals along axons from neuron to neuron might appear to be digital, but they most definitely aren't. The signal lies in the timing, both between action potentials in a particular axon and between action potentials from different axons arriving at the synapses of a particular neuron. Furthermore, there is much more signaling in the brain beyond the interneuronal. The brain is a bit like DNA—the more scientists study its operation, the more complexity they uncover.

We tend to see human qualities in anything that can interact successfully with us. This is understandable in interaction with animals, as anyone who has been to a sheepdog trial will recognize. More bizarre is early experience with a program called Eliza,

While the marketing of computers and software will unavoidably dominate in the public arena, the entry point for promoting a full understanding of digital technology must be schooling at various levels. At present, the focus is on encouraging routine use of computers, and little time is spent on getting students to think about side effects. 

Neville Holmes is an honorary research associate in the School of Computing and Information Systems at the University of Tasmania. Note that some aspects of the issues discussed here will be found in earlier essays published in this column, in particular, in May 2002 and July 2007. Contact Holmes at neville.holmes@utas.edu.au.

 Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.

THE PROFESSION

Digital Machinery and Analog Brains

Neville Holmes, *University of Tasmania*



The more pervasive digital technology becomes, the more important it is to give the public a reasonable account of how it works, how it might be used, and how its use might affect us and our society.

An article recently reprinted in a local newspaper under the title “Blow Your Mind” did indeed blow my mind. This article originally appeared in the respected British monthly magazine, *Prospect*, under the title “Neurotrash” (tinyurl.com/PrRTsNt).

The blowing of my mind caused a mixture of bewilderment and dislocation. I have studied cognitive science formally and taught it repeatedly (eprints.utas.edu.au/2391), yet I couldn’t understand much of the article and disagreed with quite a bit of what I could understand.

It was rather daunting to find that the author of the article, Raymond Tallis, is “a philosopher, poet, novelist, and cultural critic and was until recently a physician and clinical scientist.” In the *Economist’s Intelligent Life* magazine (Autumn 2009), he was listed as one of the top living polymaths in the world” (raymondtallis.com). Nonetheless, it isn’t professional to avoid a dispute

simply because of the opponent’s reputation, however grand. So here goes.

THE PROFESSION

The introductory subheading in “Neurotrash” gives the article’s gist: “It is a common and dangerous mistake to think that our minds are no more than electrical impulses in our brains.” This is somewhat like saying that it’s a mistake to think that our computations are no more than electrical impulses in our computers.


On this basis, computing professionals would certainly agree with the claim of mistakenness, but looking more closely at the accompanying argument, they would find much to disagree with in the details given. People with a background in digital technology, for example, would consider that computers and people have much in common in an illuminating way, which offers a strong basis for disagreeing with at least some of Tallis’s arguments.

Indeed, similar arguments are commonly presented in popular writings and discussions. The computing profession has a social responsibility to clearly understand the relationship between computers and people at various levels and to counter misunderstanding both within the profession and in the general public.

Tallis calls the alleged common and dangerous mistake *biologism*. In Wikipedia, biologism is equated with biological determinism in an item stating that “there is currently no support for strict biological determinism in the field of genetics or development” (tinyurl.com/WkBLDt). Determinism was a popular topic in philosophy over a century ago, and was even the topic of popular limericks (tinyurl.com/MLMHPrd).

However, Tallis sees biologism as having two strands: neuromania and Darwinitis.

Continued on page 98




Focus on Your Job Search

IEEE Computer Society Jobs helps you easily find a new job in IT, software development, computer engineering, research, programming, architecture, cloud computing, consulting, databases, and many other computer-related areas.

New feature: Find jobs recommending or requiring the IEEE CS CSDA or CSDP certifications!

Visit www.computer.org/jobs to search technical job openings, plus internships, from employers worldwide.

<http://www.computer.org/jobs>

IEEE  computer society | **JOBS**



The IEEE Computer Society is a partner in the AIP Career Network, a collection of online job sites for scientists, engineers, and computing professionals. Other partners include Physics Today, the American Association of Physicists in Medicine (AAPM), American Association of Physics Teachers (AAPT), American Physical Society (APS), AVS Science and Technology, and the Society of Physics Students (SPS) and Sigma Pi Sigma.

Discover the Latest Titles from Wiley and Wiley-IEEE Computer Society Press!

The Dark Side of Software Engineering: Evil on Computing Projects

Johann Rost, Robert L. Glass

9780470597170, Paper, 305pp, \$34.95, February 2011, Wiley-IEEE Computer Society Press

Industry experts Johann Rost and Robert L. Glass explore the seamy underbelly of software engineering in this timely report on and analysis of the prevalence of subversion, lying, hacking, and espionage on every level of software project management. Based on the authors' original research and augmented by frank discussion and insights from other well-respected figures, **The Dark Side of Software Engineering** goes where other management studies fear to tread—a corporate environment where schedules are fabricated, trust is betrayed, millions of dollars are lost, and there is a serious need for the kind of corrective action that this book ultimately proposes.



Systems and Software Engineering with Applications

Norman F. Schneidewind

9780738158525, Paper, 443pp, \$98.00, September 2009, Standards Information Network

By way of this book, Norman Schneidewind has officially bridged the gap between the two disparate fields. Filled with many real-world examples drawn from industry and government, **Systems and Software Engineering with Applications** provides a new perspective for systems and software engineers to consider when developing optimal solutions. This unique approach to looking at the big picture when addressing system and software reliability can benefit students, practitioners, and researchers. Excel spreadsheets noted in the book are available on CD-Rom for an interactive learning experience.



Design of Multithreaded Software: The Entity-Life Modeling Approach

Bo I. Sandén

9780470876596, Cloth, 320pp, \$89.95, March 2011, Wiley-IEEE Computer Society Press

This book assumes familiarity with threads (in a language such as Ada, C#, or Java) and introduces the entity-life modeling (ELM) design approach for certain kinds of multithreaded software. ELM focuses on “reactive systems,” which continuously interact with the problem environment. These “reactive systems” include embedded systems, as well as such interactive systems as cruise controllers and automated teller machines. Part I covers two fundamentals: program-language thread support and state diagramming. These are necessary for understanding ELM and are provided primarily for reference. Part II covers ELM from different angles. Part III positions ELM relative to other design approaches.

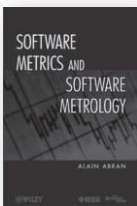


Software Metrics and Software Metrology

Alain Abran

9780470597200, Paper, 328pp, \$69.95, June 2010, Wiley-IEEE Computer Society Press

Software Metrics and Software Metrology looks at the fundamentals of the design of a measurement method, which forms the foundation of the measures available in the sciences and in engineering. Alain Abran provides a step-by-step approach to both analyzing the design of current software measures and designing new, robust software measures for a specific business or engineering need. He draws upon years of experience to ensure that software engineers and managers will apply the best practices in software measurement—and therefore be equipped to respond to the most demanding customers and feel supported by senior executives.



Distributed Database Management Systems: A Practical Approach

Saeed K. Rahimi, Frank S. Haug

9780470407455, Cloth, 728pp, \$135.00, August 2010, Wiley-IEEE Computer Society Press

Distributed Database Management Systems is divided into three units. The first provides a theoretical foundation for understanding the internal processing of the DD-BMS available to address these issues. The second unit presents the “state of the practice,” examining the architectural alternatives that practitioners will likely encounter in the real world and the exploring the general requirements for any platform capable of implementing a DD-BMS architectural alternative—including those yet to be invented. The final unit focuses on distributed database implementation, examining three platforms suitable for the development of a real DD-BMS system—the Java Message Service (JMS), the Java 2 Enterprise Edition (J2EE), and the Microsoft .NET Framework. For each, a “starter kit” is provided (containing a detailed overview and an extensible framework) and discussed in detail.



Tutorial on Hardware and Software Reliability, Maintainability and Availability

Norman F. Schneidewind

9780738156774, Paper, 31pp, \$98.00, October 2008, Standards Information Network

This tutorial serves as a companion document with the purpose of elaborating on key software reliability process practices in more detail than can be specified in the Recommended Practice. However, since other subjects like maintainability and availability are also covered, the tutorial can be used as a stand-alone document. While the focus of the Recommended Practice is software reliability, software and hardware do not operate in a vacuum. Therefore, both software and hardware are addressed in this tutorial in an integrated fashion. The narrative of the tutorial is augmented with illustrative solved problems. The recommended practice [IEEE P1633] is a composite of models and tools and describes the “what and how” of software reliability engineering. It is important for an organization to have a disciplined process if it is to produce high reliability software.

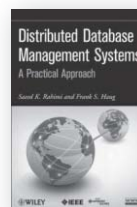


Fundamentals of Performance Evaluation of Computer and Telecommunications Systems

Mohammed S. Obaidat, Noureddine A. Boudriga

9780471269830, Cloth, 459pp, \$116.00, January 2010, Wiley

Fundamentals of Performance Evaluation of Computer and Telecommunication Systems uniquely presents all techniques of performance evaluation of computers systems, communication networks, and telecommunications in a balanced manner. Written by the renowned Professor Mohammad S. Obaidat and his coauthor Professor Noureddine Boudriga, it is also the only resource to treat computer and telecommunication systems as inseparable issues. The authors explain the basic concepts of performance evaluation, applications, performance evaluation metrics, workload types, benchmarking, and characterization of workload. This is followed by a review of the basics of probability theory, and then, the main techniques for performance evaluation—namely measurement, simulation, and analytic modeling—with case studies and examples.



Enter promotion code **SOF11** to receive **20% off** these featured titles at checkout.

To Order

1 (877) 762-2974 North America
+ 44 (0) 1243 843294 in Rest of World