
COMPUTING
edge
> Software Process versus Design Quality: Tug of War?
> Architecture Haiku
> Designing Resource-Aware Cloud Applications
AUGUST 2015
www.computer.org
IEEE
IEEE computer society

Program Committee Members

- Artur Andrzejak, Heidelberg University
- Danilo Ardagna, Politecnico di Milano
- Martin Arlitt, HP Labs
- Jean Bacon, University of Cambridge
- Novella Bartolini, Univ. of Rome "Sapienza"
- Sonia Ben Mokhtar, LIRIS-CNRS
- Sara Bouchenak, University of Grenoble
- Yerom-David Bromberg, Univ. of Bordeaux 1
- Roy Campbell, UIUC
- Giuliano Casale, Imperial College London
- Abhishek Chandra, University of Minnesota
- Haibo Chen, Shanghai Jiao Tong University
- Jinjun Chen, Univ. of Technology Sydney
- Gregory Chockler, Royal Holloway Univ. of London
- Paolo Costa, Microsoft Research Cambridge
- Dilma Da Silva, Texas A&M University
- Eliezer Dekel, IBM Research Haifa
- Erik Elmroth, Umea University
- Patrick Eugster, Purdue University
- David Eyers, University of Otago
- Christof Fetzer, TU Dresden
- Jose Fortes, University of Florida
- Ian Foster, University of Chicago
- Xiaohui Gu, North Carolina State University
- Indranil Gupta, UIUC
- Seif Haridi, KTH Royal Institute of Tech./SICS
- Bingsheng He, Nanyang Technological University
- Ching-Hsien Hsu, Chung Hua University
- Jinho Hwang, IBM Research
- Alex Iosup, Delft Univ. of Technology
- Arun Iyengar, IBM Research
- K. R. Jayaram, IBM Research
- Hai Jin, Huazhong Univ. of Sci. and Tech.
- Ruediger Kapitza, TU Braunschweig
- Dejan Kostic, Royal Institute of Technology
- Philippe Lalanda, Universite Joseph Fourier
- Wolfgang Lehner, TU Dresden
- Baochun Li, University of Toronto
- Satoshi Matsuoka, Tokyo Institute of Tech.
- Klara Nahrstedt, UIUC
- Beng Chin Ooi, National Univ. of Singapore
- Manish Parashar, Rutgers University
- Fernando Pedone, University of Lugano
- Guillaume Pierre, IRISA/Univ. of Rennes 1
- Thomas Sandholm, HP Labs
- Karsten Schwan, Georgia Tech
- Marco Serafini, QCRI
- Weisong Shi, Wayne State University
- Liuba Shrira, Brandeis University
- Evgenia Smirni, College of William and Mary
- Stefan Tai, TU Berlin
- Francois Taiani, Universite de Rennes 1
- Jon Weissman, University of Minnesota
- Dan Williams, IBM Research
- Joel Wolf, IBM Research
- Bernard Wong, University of Waterloo
- Timothy Wood, George Washington University
- Yongwei Wu, Tsinghua University
- Cheng-Zhong Xu, SIAT

Important Dates

- Abstract Due: August 29, 2015
- Full Paper Due: Sept. 5, 2015
- Acceptance Notification: Dec. 5, 2015

IEEE International Conference on Cloud Engineering (IC2E) 2016

Call For Papers

The IEEE International Conference on Cloud Engineering (IC2E) conference series seeks to provide a high-quality and comprehensive forum, where researchers and practitioners can exchange information on engineering principles, enabling technologies, and practical experiences as related to cloud computing. By bringing together experts that work on different levels of the cloud stack - systems, storage, networking, platforms, databases, and applications, IC2E will offer an end-to-end view on the challenges and technologies in cloud computing, foster research that addresses the interaction between different layers of the stack, and ultimately help shape the future of cloud-transformed business and society.

We invite submissions of high quality papers describing fully developed results or on-going foundational and applied work on the following topics:

- X as a Service, where X includes Backend, Business Process, Database, Information, Infrastructure, Network, Platform, Security, Software, and Storage
- Big data management and analytics
- Software engineering methodology and practices for cloud computing
- Mobile cloud computing
- Virtualization technology
- Performance, dependability, and service level agreements
- Cloud security, privacy, compliance, and trust
- Workload deployment and migration
- Energy management in cloud centers
- Cloud programming models and tools
- Hybrid cloud integration
- Service lifecycle management
- Service management automation
- Metering, pricing, and software licensing
- Cloud applications

This year, the conference will be held in Berlin, Germany.

TCBIS

STC CLOUD COMPUTING

IEEE computer society

IEEE CLOUD COMPUTING

IC2E 2016 Organizing Committee

General Chairs
Goeffrey Fox, Indiana University, USA
Stefan Tai, TU Berlin, Germany

Workshops Chairs
Dennis Gannon, Microsoft, USA

Panels Chair
Schahram Dustdar, TU Vienna, Austria

Doctoral Symposium Chair
Judy Qiu, Indiana University, USA

Publicity Chairs
Ali Kanso, Ericsson, Canada
Ningfang Mi, Northeastern University, USA

Proceedings Chair
Sonia Ben Mokhtar, LIRIS-CNRS, France

Steering Committee
Jean Bacon, University of Cambridge, UK
Roy Campbell, UIUC, USA
Dilma Da Silva, Texas A&M Univ., USA

Program Committee Chairs
Lucy Cherkasova, HP Labs, USA
Peter Pietzuch, Imperial College London, UK
Cho-li Wang, University of Hong Kong, China

Industry Track Chair
Shu Tao, IBM Watson Research, USA

Tutorials Chairs
David Eyers, University of Otago, New Zealand

Local Arrangements and Registration Chair
David Bermbach, TU Berlin, Germany

Finance Chairs
Dominik Ernst, TU Berlin, Germany

Web and Information Chair
Jacob Eberhardt, TU Berlin, Germany

Masaru Kitsuregawa, University of Tokyo, Japan
Christos Kozyrakis, Stanford University, USA
Hui Lei, IBM Research, USA
Dejan Milojicic, HP Labs, USA

COMPUTING edge

IEEE COMPUTER SOCIETY http://computer.org • +1 714 821 8380

Printed with inks containing soy and/or vegetable oils

IEEE computer society

STAFF

BPA WORLDWIDE

Circulation: ComputingEdge 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; voice +1 714 821 8380; fax +1 714 821 4010; IEEE Computer Society Headquarters, 2001 L Street NW, Suite 700, Washington, DC 20036.

Postmaster: Send undelivered copies and address changes to IEEE Membership Processing Dept., 445 Hoes Lane, Piscataway, NJ 08855. Application to Mail at Periodicals Postage Prices is pending 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.

IEEE Computer Society Magazine Editors in Chief

Computer
Sumi Helal, *University of Florida*

IEEE Software
Diomidis Spinellis, *Athens University of Economics and Business*

IEEE Internet Computing
M. Brian Blake, *University of Miami*

IT Professional
San Murugesan, *BRITE Professional Services*

IEEE Security & Privacy
Shari Lawrence Pfleeger, *Dartmouth College*

IEEE Micro
Lieven Eeckhout, *Ghent University*

IEEE Computer Graphics and Applications
L. Miguel Encarnação, *ACT, Inc.*

IEEE Pervasive Computing
Maria Ebling, *IBM T.J. Watson Research Center*

Computing in Science & Engineering
George K. Thiruvathukal, *Loyola University Chicago*

IEEE Intelligent Systems
Daniel Zeng, *University of Arizona*

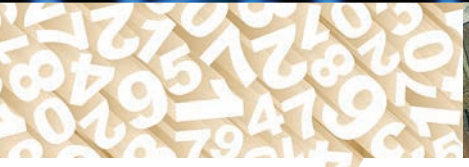
IEEE MultiMedia
Yong Rui, *Microsoft Research*

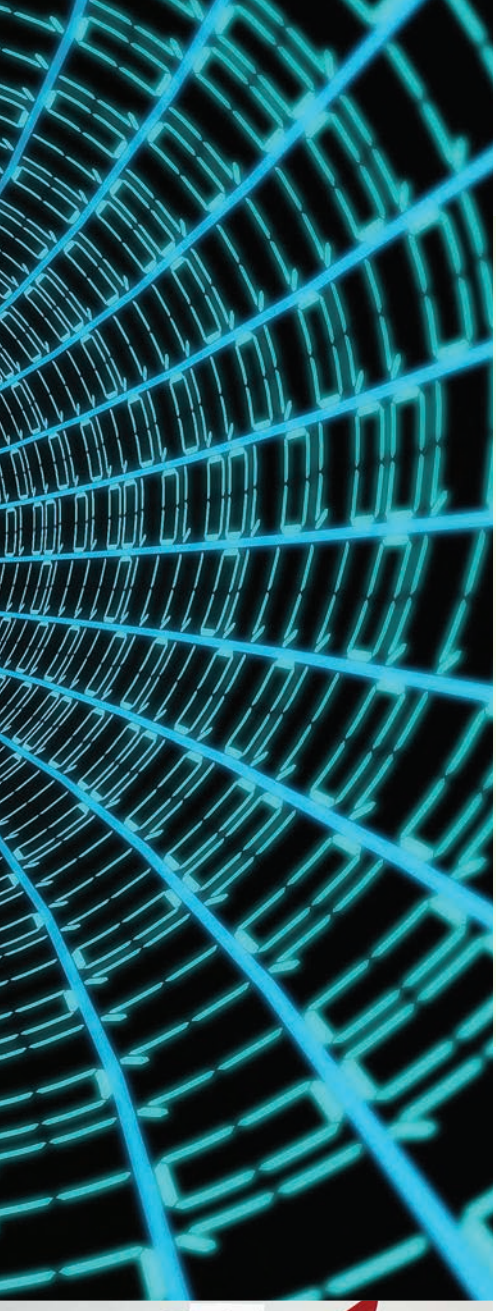
IEEE Annals of the History of Computing
Nathan Ensmenger, *Indiana University Bloomington*

IEEE Cloud Computing
Mazin Yousif, *T-Systems International*

AUGUST 2015 • VOLUME 1, NUMBER 8

COMPUTING edge





44

Connected Tools in Digital Design

Departments

Increasing Interconnection Network Throughput with Virtual Channels

Stephen W. Keckler, NVIDIA and University of Texas at Austin

This installment highlighting the work published in IEEE Computer Society journals comes from IEEE Transactions on Parallel and Distributed Systems.

Interconnection networks form the backbone of scalable computer systems, including high-performance computers and datacenters. High-performance networks must provide high throughput and bandwidth while delivering messages with minimal latency.

As processors that connect to the network inject more messages, the network load rises and can cause two potential deleterious conditions: network deadlock—where messages are blocked in the network and cannot progress—and an exponential increase in end-to-end latency, as messages must wait in the network for prior messages to be delivered. The network architect's job is to deliver high bandwidth and low latency for the largest network load possible, but with limited cost and power.

Before the 1990s, scalable computing systems were typically built with two separate networks to avoid deadlock. These networks were often assigned to "request" and "reply" traffic classes, and messages in separate classes did not interfere as they progressed through the network. The networks had their own individual wires to ensure that reply messages did not get stuck behind request messages. Unfortunately, this approach increases network cost by doubling the number of wires, and can leave links idle as congestion rises.

In "Virtual Channel Flow Control" (*IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 2, 1992, pp. 194–205), William J. Dally proposed an alternate approach to alleviate deadlock and provide better network performance. Virtual channels include two or more virtualized networks that share the physical wire links and routers, but have their own buffer storage for in-flight messages. In modern interconnection networks, messages are decomposed into flow-control digits (*flits*) that transit the network like train cars. When a message reaches a congested point, its flits reside in the virtual channel buffers, leaving the links free for messages belonging to other virtual channels. The decoupling of buffering from links in virtual channels enables deadlock-free routing algorithms and the passing of blocked messages in the network, using otherwise idle link resources. With virtual channels, a system doesn't incur the link cost of multiple networks and can more easily multiplex traffic from different flows over a single set of physical links.

Virtual channels have been extremely influential in the theory and implementation of interconnection networks. Almost every high-performance network designed since the publication of "Virtual Channel Flow Control" uses virtual channels. Examples include the Cray T3D, T3E, BlackWidow, and Aries networks, as well as the IBM BlueGene L, P, and Q networks. Virtual channels are also critical to preventing deadlock in adaptive routing algorithms that are prevalent today.

Dally's work constituted a fundamental advance in interconnection network architecture that provided better capability and performance while reducing both cost and complexity.

STEPHEN W. KECKLER is the senior director of architecture research at NVIDIA and an adjunct professor at the University of Texas at Austin. Contact him at skeckler@cs.utexas.edu.

 Published by the IEEE Computer Society

Magazine Roundup

The IEEE Computer Society's lineup of 13 peer-reviewed technical magazines covers cutting-edge topics ranging from software design and computer graphics to Internet computing and security, from scientific applications and machine intelligence to cloud migration and microchip manufacturing. Here are highlights from recent issues.

Computer

Self-aware and self-expressive systems support effective, autonomous, and self-adaptive behaviors, fostering advanced intelligent decision making in dynamic and uncertain environments. These systems are the topic of *Computer*'s July 2015 special issue.

IEEE Software

Over the years, Web software has become more widely used, including for mission-critical tasks. This makes developing secure software that preserves privacy particularly important. However, doing so is increasingly challenging because Web software is becoming more complex. This and related subjects are addressed in *IEEE Software*'s July/August 2015 special issue on **security and privacy on the Web**.

IEEE Internet Computing

Connectivity, interoperability, sensing, and instant feedback via smartphones provide new opportunities for gaining insights into our health-related behavior. Such insights improve our understanding of what motivates people to make healthy changes throughout their lives. *IEEE Internet Computing*'s July/August 2015 special issue on **continuous digital health** reviews advances in wireless, connected, and mobile health research.

Computing in Science & Engineering

Scientists and engineers constantly face new challenges, despite myriad advances in computing. New technology advances have empowered the general public to participate in the scientific process via individual and computational resources, which could help alleviate these challenges. This is called **citizen science** and is the subject of *CiSE*'s July/August 2015 special issue.

IEEE Security & Privacy

Biometrics are easier to utilize than long text passwords on mobile devices. However, the approach must be implemented

2376-113X/15/$31.00 © 2015 IEEE Published by the IEEE Computer Society

properly, and users might want to think twice until biometric authentication can be proven secure and reliable. These issues are addressed in the article **"Biometric Authentication on Mobile Devices"** from *IEEE S&P*'s May/June 2015 issue.

IEEE Cloud Computing

Security concerns related to managing data, applications, and interactions limit the rapid, large-scale deployment of cloud-based services. The article "Security and Privacy in Cloud Computing: Vision, Trends, and Challenges," from *IEEE Cloud Computing*'s March/April 2015 issue, explores the various challenges to, approaches for, and limitations of **cloud security and privacy**.

IEEE Computer Graphics and Applications

The IEEE VIS 2014 workshop program offered a unique peek at **emerging data-visualization research topics**. *IEEE CG&A*'s May/June 2015 special section—titled "Data-Driven Discoveries: Pushing Visualization Research Further"—showcases two research pieces from the workshop that push the state of the art to enable data-driven discoveries with visualization.

IEEE Intelligent Systems

Social media sites for medical patients have emerged as major platforms for discussion of treatments and drug side effects, making them a promising source of information for analysis. The authors of "Identifying Adverse Drug Events from Patient Social Media: A Case Study for Diabetes," from *IEEE Intelligent Systems'* May/June 2015 issue, developed a novel **information-extraction framework** that has significantly outperformed prior work in this area.

IEEE MultiMedia

Food is an emerging topic of interest for the multimedia community. Projects in this area have focused on tracking people's food intake, with the aim of improving health. The authors of "FoodLog: Multimedia Tool for Healthcare Applications," from *IEEE MultiMedia's* April–June 2015 issue, developed a **multimedia food-recording tool** that offers a novel method for recording daily intake. It could also be used in other healthcare-monitoring apps.

IEEE Annals of the History of Computing

"Notes on the Evolution of **Computer Security Policy in the US Government**, 1965–2003," in *IEEE Annals*' April–June 2015 issue, discusses White House, congressional, and Department of Defense cybersecurity policy documents. The documents suggest that early legislative approaches' focus on privacy and innovation might have complicated security improvements.

IEEE Pervasive Computing

Living laboratories are real-life settings with embedded heterogeneous technology in which subjects can conduct everyday activities while researchers measure and observe their interactions. In "Living Labs for Pervasive Healthcare Research," from *IEEE Pervasive Computing*'s April–June 2015 issue, the authors focus on several spaces instrumented to design and evaluate pervasive healthcare systems and applications.

IT Professional

"Internet of Things Perspectives," in *IT Pro*'s May/June 2015 issue, offers an expert opinion on two important **Internet of Things** (IoT) issues: an IoT research agenda for ensuring the development of a trusted, secure, reliable, and interoperable net-centric computing environment; and the IoT as a human agent, extension, and complement.

IEEE Micro

IEEE Micro's May/June 2015 special issue on the 2014 **top picks in computer architecture** presents 12 articles that describe novel, exciting research in diverse areas including reconfigurable logic, hardware acceleration, design for correctness and verification, nonvolatile memory, high-throughput computing, and programmability-enhancing frameworks.

Computing Now

The Computing Now website (http://computingnow.computer.org) features **up-to-the-minute computing news** and blogs, along with articles ranging from peer-reviewed research to opinion pieces by industry leaders.

Focus on Software

In this month's *ComputingEdge,* we look at some of software's many complex, important issues. For example, there have long been discussions about the effect that closely following software life-cycle processes can have on software quality. This is the focus of "Software Process versus Design Quality: Tug of War?," from *IEEE Software*.

For the general public, how software is made remains a mystery. In "Then a Miracle Occurs," *IEEE Software* columnist Grady Booch says developing software-intensive systems is like many other things but also like no other thing.

Software patents have been a controversial topic for years. *Computer*'s " 'Curiouser and Curiouser!'—The Fallout from Alice" looks at the ramifications of last year's US Supreme Court decision on the subject in Alice Corp. v. CLS Bank International.

There are people who say something must be done because efforts to produce open, flexible, and reusable software have failed. This is addressed in *Computer*'s "Do We Need a Software Czar?"

"The Approximation Tower in Computational Science: Why Testing Scientific Software Is Difficult," in *Computing in Science & Engineering*, uses a solar-system simulation to help explain approaches to solving the thorny problem of testing scientific software.

IEEE Cloud Computing's "Invention, Innovation, and New APIs" explores new API developments and trends that are emerging from both standards organizations and software companies.

An architecture haiku captures software system architecture's most important details on a single piece of paper, to help development teams focus on essential issues. *IEEE Software*'s "Architecture Haiku: A Case Study in Lean Documentation" examines this approach.

ComputingEdge articles on topics other than software include:

- *IEEE Internet Computing*'s "Lest We Forget," in which author Vinton Cerf discusses how digital information can be preserved for future access;
- *IEEE Pervasive Computing*'s "Crowdfunding for Ubicomp Products: Interviews with Amanda Williams and Khai Truong" explores what it takes to exploit crowdfunding to turn an idea into a product;
- *IEEE Security & Privacy*'s "Silver Bullet Talks with Brian Krebs" is an interview with a well-known information security reporter and blogger;
- *Computer*'s "Designing Resource-Aware Cloud Applications" examines how the cloud requires a rethinking of software development; and
- *IT Professional*'s "Internet of Cores" addresses a new approach for provisioning on-demand elastic computing and storage resources to people and things anywhere, anytime.

2376-113X/15/$31.00 © 2015 IEEE Published by the IEEE Computer Society August 2015

Invention, Innovation, and New APIs

ALAN SILL

Texas Tech University,
alan.sill@standards-now.org

COMPUTING STANDARDS MUST KEEP UP WITH THE PACE OF CHANGE IN SOFTWARE AND HARDWARE DEVELOPMENT TO REMAIN RELEVANT AND USEFUL. This requires paying attention to methods by which such innovations are created, adopted, updated, and socialized within their target user communities. Significant changes are already taking place in practice, led by work from individual contributors as well as formal and informal organizations.

In this issue, I cover some recent developments in application programmer interfaces (APIs) and factors that affect them, and explore new trends that are emerging from both standards organizations and software companies to keep their work up to date, relevant, and successful.

How Standards Influence the Pace of Change

Introducing a film in the American Experience series on the history of the telephone, series host David McCullough invited viewers to consider the astonishing period of technological innovation that took place in the late 19th century, leading to consequent changes throughout the globe:

> All at once, in a great inventive surge came the light bulb, the refrigerator, electric street cars, barbed wire, the typewriter, the elevator, skyscrapers, the horseless carriage—all in about thirty-five years, plus one other extremely important contribution that would change our whole way of talking to each other—the telephone.[1]

The success of each of these innovations required standards to be invented and adopted. Electricity required reproducible components that would fit together, and choices had to be made in terms of voltage, physical sockets, and connection interfaces. Units of usage in terms of billable measurements and metrics had to be standardized and adopted. Standardized streetcar designs and rail widths had to be adopted or migrated from similar technologies and had to be uniform at least within each city or region in which the lines ran.

Each such innovation created a corresponding need for development of standards in material composition, tooling, gauges, and associated measurements and metrics. Standardized machinery and component designs, material strengths, and other engineering details were used as the basis for growth in this astonishing period of invention and adoption. New industrial techniques came into use in a wide variety of arenas far beyond those in which they were initially created and applied, partly because industrialized and standardized components could be designed and mass-produced using methods that could easily be adapted for use in other areas.

This cross-fertilization of industrial engineering standards allowed invention in other fields to spin into being rapidly, with advances in one area feeding quickly into adoption and progress in others. As a result, audio recording, radio transmission, moving films, television, and, decades later, digital processing, computers, and the Internet came into existence, all the result of this incredible period of concentrated cross-domain multidisciplinary expan-

 Published by the IEEE Computer Society

sion and invention. Standards were at the core of this technological revolution.

Invention, Innovation, and Serendipity

Progress can take unanticipated turns. Rather than trying to invent the telephone, Bell was trying to develop his idea for a multiwavelength version of the simpler telegraph, with each coded signal carried as modulations on its own separate sound frequency.[1] If he had succeeded, this would have provided the logical precursor for modern multiwavelength communications, a technology that is used now in a different but substantially similar way on essentially all long-distance optical fiber connections.

Direct sound transmission proved more popular with the public and with investors than a multiplexed version of the telegraph. As a result, an entire new field of communication was invented in a period that could be characterized in modern terms as "disruptive." Telecommunications did not stand alone among the innovations of this period, but played a crucial role in allowing these developments to be useful to human interaction.

Future scholars might look back on the period of change that we're living through now, and seek key elements that were agents of change in the same way that electricity and the telephone fundamentally altered the nature of society in the 19th century. When they do so, the invention of the RESTful API could stand out as one of the most important elements of 21st century change in both software development and standards.

APIs for Automation and Machine-to-Machine Communication

The fundamental paradigms for APIs have been explored in this column and related articles in previous issues. Let's concentrate for a moment on the use of APIs for automated tasks and for machine-mediated communication.

Although machine-to-machine communication might seem to be the primary function for APIs in cloud and distributed computing, significant aspects of current designs are driven by the need for humans to understand the characteristics of the resulting communication patterns. The "P" in API is of course the programmer, and the design of APIs is largely influenced by the need for programmers to be able to manage their use, even in highly automated and complex settings.

The invention of the RESTful API could stand out as one of the most important elements of 21st century change in both software development and standards.

Representational state transfer (REST) became popular as a design pattern for APIs over previous models partly because it's easier to understand the individual and collective interactions between multiple automated systems than other formally defined systems. Although REST design patterns can obviously be implemented using any transport protocol, the use of hypertext and hypermedia written in human-readable formats, along with relatively readable data formats such as JavaScript Object Notation (JSON) and Extensible Markup Language (XML), allows relatively easy creation and debugging for interfaces that use these tools. Consequently, REST-over-HTTP using JSON and/or XML data formats is now the dominant design pattern for new APIs.

At the time that I'm writing this column, there are more than 13,000 APIs registered on a popular website that catalogs such things (http://programmableweb.com). Many of these offer little more in terms of design than access to the data available at a particular website through programmable methods. Others are deeply complicated interfaces to engines of state for complex cyberinfrastructures.

The picture I've drawn is good as a general overview. At a more detailed level, part of the reason that this pattern is successful is that it coexists with other ways of accomplishing things. A variety of data formats and transfer protocols can be used successfully to augment, supplement, and extend this design pattern or to completely replace it if necessary. Beyond JSON and XML, over twenty other data formats are accepted in the predefined list for APIs registered on programmableweb.com, and while REST-over-HTTP is presently popular, it coexists with many other older approaches that remain workable and usable. A successful paradigm will accommodate all of these methods.

Along the way, certain design patterns have emerged to rationalize the ability for programmers to provision and advertise features of APIs in automated ways. As I've discussed in previous issues, these include Swagger (http://swagger.io),

API Blueprint (http://apiblueprint.org), and the RESTful API Markup Language (RAML; http://raml.org). There are even products emerging that allow developers to generalize these API specification rule engines and treat them as plug-ins to software development processes.

It's therefore worth looking now at current trends in API design to see if human readability and RESTful features can be expected to continue to dominate API design in the future, or whether this may be a temporary state of affairs in the evolution of machine-to-machine communication and the design of automated systems.

API Usage and the Problem of State

REST came into being to allow communication of state changes, primarily for Web applications, through hypermedia. This model works well when the state being conveyed is in the form of presentation of new information or the expression of state transitions in information on either end of a communication pipeline that can be carried in a single communication. The basic rule in REST design is that changes in application state must be carried and expressed through hypermedia.

When used for machine control or manipulation of information that might be influenced in other ways by out-of-band factors, the model develops some weaknesses that are hard to correct or address in purely RESTful terms. These weaknesses are not limited to REST (they could be present using other communication models, such as SOAP), but they produce a variety of symptoms that lead to further discussion of the model. In addition, many API usage patterns are now called RESTful even if they're simply Web-based APIs that communicate over HTTP, even if they don't conform to Roy Fielding's original concepts.[2]

> The need to express state changes quickly and in a nonserial fashion is being built into future versions of the HTTP protocol.

The point to be made here is that the great multiplicity and ubiquity of use of APIs, and even of development patterns that allow automated production and discovery of API features, are driving software development in the direction of machine usability, rather than human readability, becoming the primary factor in the production and use of APIs. Although specifications have emerged, and standards in API patterns are important, this factor more than any other is driving the evolution in this area.

Further complications arise when we consider that a large and growing number of APIs aren't simply limited to addressing the problem of data exchange. As the Internet of Things grows around us, and as the use of automated and autonomous systems in cyberspace continues to grow in importance, information about transitions in state that take place out of the context of the API itself must necessarily also be conveyed.

Even this very brief overview gives enough information to conclude that APIs will have to become more complex, should probably be rewritten to optimize at least some portions of communication for machine readability even at the expense of human readability, and will have to carry more complex and richer information about asynchronous changes in state than were usually the case in simple Web-oriented HTTP communications.

Elements of Future API Design

It's hard to see how the present picture, which is currently so successful in distributed computing design, can last in the face of this onslaught of complexity. It's also difficult to see how things can change quickly from the situation I've described, given the enormous number of ways in which these design patterns have been successful.

One element that might contribute to future evolution in API design is already under way. The need to express state changes quickly and in a nonserial fashion is being built into future versions of the HTTP protocol. At the time of this writing, the HTTP/2 and HPACK final drafts are working their way through publication after receiving final approval as standards-track IETF documents.[3,4]

Changes in HTTP/2 over previous versions include binary instead of textual representations; fully multiplexed, instead of ordered and blocking communications that can therefore use as little as one connection for parallelism; the use of header compression to reduce overhead; and provisions that allow servers to push responses into client caches rather than wait for the client

to issue pull requests. These features are primarily aimed at allowing conventional Web communications to take place more quickly and replace previous efforts, such as SPDY, for this purpose, but it's easy to see the potential to apply them to API design.

In general, HTTP/2 will be backward compatible and can be used without impacting existing HTTP-based API models. Inspecting the list of features makes it clear, however, that they represent a step in the direction of more API- and control-friendly protocols for machine-to-machine communication. As an example, Google has already begun to build on HTTP/2 and to extend beyond it in its gRPC remote procedure call interface (www.grpc.io). This interface indicates the possibility of future trends in that it isn't primarily intended for use in browser settings. It includes high-level features that HTTP libraries typically don't provide, such as interaction with flow control at the application layer, cascading call cancellation, and provisions for load balancing and failover within the protocol itself. It diverges in other significant ways also from pure REST conventions (www.grpc.io/faq).

Proposed changes in network addressing are also being considered that have implications for content delivery and for the design of APIs. Examples of such proposals would include the US National Science Foundation's Named Data Networking project (NDN, http://named-data.net), which aims to provide a future Internet architecture that that would function as a distribution network, and in effect be a replacement for existing network functionalities at layer 3. Another example is the Content-Centric Networking approach (CCN, http://www.ccnx.org/), which is designed to deliver content objects directly by name-based URIs instead of by addressing network end-points, and to secure these objects independently as opposed to securing the connection used to retrieve them.

When combined with other related concepts, such as unikernel-based single-instance server instances for automation and control, existing approaches to problems of state transition, management, and security could radically be altered by such methods, especially in Internet-of-Things and autonomous computing settings.

The approaches I've described can thus lead to a reexamination of the basic design patterns of REST versus object orientation versus service orientation in machine-to-machine communication, and even of Internet addressing itself, but ultimately bring us back to consider topics that are more closely related to innovation, serendipity, and invention. After all, many years in the future, when we look back at this period, wouldn't it be nice if we could view it not only as a period of disruption, but also one of great understanding and forward progress?

WHAT DO YOU THINK THE DESIGN PATTERN FOR THE BUILDING BLOCKS OF FUTURE API CONTROL SHOULD BE? Are we on the right track, and if not, what do you think should change? Please respond with your opinions on this topic or on previous columns, and include any news you think the community should know about standards, compliance, or related topics. I can be reached for this purpose at alan.sill@standards-now.org.

References

1. "The Telephone," *American Experience*, PBS Video, WGBH Boston, 1997; www.pbs.org/wgbh/amex/telephone/index.html.
2. J. Louvel, "RESTistential Crisis over Hypermedia APIs," InfoQ, 31 Mar. 2014; www.infoq.com/news/2014/03/rest-at-odds-with-web-apis.
3. R. Peon and H. Ruellan, "HPACK: Header Compression for HTTP/2," IETF working draft, work in progress, Feb. 2015.
4. M. Belshe and R. Peon, "Hypertext Transfer Protocol version 2," IETF working draft, work in progress, Feb. 2015.

ALAN SILL *directs the US National Science Foundation Center for Cloud and Autonomic Computing site at Texas Tech University, where he is also a senior scientist at the High Performance Computing Center and adjunct professor of physics. Sill holds a PhD in particle physics from American University and is an active member of IEEE, the Distributed Management Task Force, TeleManagement Forum, and other cloud standards working groups, and serves either directly or as liaison for the Open Grid Forum on several national and international standards roadmap committees. He serves as vice president of standards for the Open Grid Forum and co-chairs the US National Institute of Standards and Technology's "Standards Acceleration to Jumpstart Adoption of Cloud Computing" working group, and is one of the organizers of the ongoing developer-oriented Cloud Interoperability Plugfest series of meetings. For further details, visit http://cac.ttu.edu or contact him at alan.sill@standards-now.org.*

This article originally appeared in IEEE Cloud Computing, *vol. 2, no. 2, 2015.*

"Curiouser and Curiouser!"—The Fallout from *Alice*

Brian M. Gaff, McDermott Will & Emery, LLP

The ramifications of last year's US Supreme Court decision in Alice Corp. v. CLS Bank International *concerning software patentability have become apparent as other courts grapple with and implement that decision.*

For an expanded discussion on this topic, listen to the podcast that accompanies this column at www.computer.org/computing-and-the-law.

On 19 June 2014, the US Supreme Court decided in *Alice Corp. v. CLS Bank International* that patents on computerized methods for financial trading were invalid. Many observers were hoping that the decision would provide much-needed clarity on whether software was patentable.

Generally speaking, the *Alice* decision didn't provide that clarity. However, lower courts—which are bound to follow that decision—have provided guidance on the application of *Alice* and its scope as they have ruled in other patent cases.

See **www.computer.org/computer-multimedia** for multimedia content related to this article.

ALICE—A QUICK REVIEW

Alice Corp., a financial services firm, accused CLS of infringing its computerized financial trading patents. CLS argued that the patents were invalid because their subject matter—computerized methods—wasn't patentable. The Supreme Court issued a unanimous decision to uphold two lower courts' rulings that the patents were invalid.

The Supreme Court looked to section 101 of US patent law, which says that "any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof" is eligible for a patent. However, for more than 150 years, the Court has said that laws of nature, physical phenomena, and abstract ideas aren't patentable. These exceptions have blocked patents on, for example, an algorithm for converting binary-coded

 Published by the IEEE Computer Society 2376-113X/15/$31.00 © 2015 IEEE

EDITOR **BRIAN M. GAFF**
McDermott Will & Emery, LLP; bgaff@mwe.com

decimal numbers into binary form and a mathematical formula for computing alarm limits for a catalytic conversion process, which the Court found to be abstract ideas. Similarly, in 2008's *Bilski v. Kappos*, the Court said that a method for buyers and sellers of energy-market commodities to protect themselves against the risk of price changes wasn't eligible to be patented because it was an abstract idea.

The Court arrived at its decision in *Alice* using a two-step test. First, it determined that the subject matter claimed in Alice Corp.'s patents was like that in *Bilski* and thus wasn't eligible for a patent. Second, the Court found that the claimed methods were implemented on a generic computer that used generic computer parts (for example, "data storage unit" and "communication controller"); these added "nothing of substance to the underlying abstract idea," effectively invalidating the patents at issue and ending Alice Corp.'s infringement suit.

The *Alice* decision didn't provide a clear and universal yes or no answer to the question of whether software is patentable. Instead, the decision showed how to apply a test to answer that question on a case-by-case basis.

THE DECISIONS

The US Court of Appeals for the Federal Circuit hears all appeals of rulings and verdicts in patent cases in the US. It is one step above federal district courts, where most patent cases are tried, and one step below the US Supreme Court. In the year since the Supreme Court ruling on *Alice*, the Federal Circuit issued a handful of decisions that applied *Alice* to different cases with different facts. Those decisions reveal *Alice*'s scope.

Among this first group of cases that the Federal Circuit considered since *Alice*, only once has the court upheld the validity of the patent at issue; in the other cases, it invalidated the patents. Though they're based on a small sample size, those statistics don't bode well for patents that cover nothing more than computerized versions of basic processes. That's consistent with the second step of the Supreme Court's test: if the patent claims don't add something significant to the process, then those claims are likely invalid.

Over the past year, some of the subject matter that the Federal Circuit said was unpatentable includes methods of reducing distortion in image processing. The court ruled that this amounted to nothing more than manipulating existing data using an algorithm. The court also invalidated a patent on a computerized bingo game on the grounds that it was only a series of mental steps implemented by a computer.

Many patents challenged under *Alice* relate to using the Internet in conjunction with well-recognized processes or methods. Not surprisingly, the Federal Circuit has been critical of those patents. For example, it ruled that a method of showing an advertisement before showing free content on the Internet wasn't eligible for a patent. Similarly, it ruled that providing a sales guarantee for an online transaction wasn't patentable.

These last examples show that the Federal Circuit has disfavored patents that, in effect, take a well-known or basic process and implement it on a generic computer using generic computer hardware and generic instructions. The court reiterated this by invalidating a patent that covered a scanning method used in an automatic teller machine to recognize amounts on checks being deposited. The court noted that data collection, recognition, and storage are well-known, and having a computer perform those steps didn't add anything significant to render the method eligible for a patent.

> Avoid writing patent claims that describe using a generic computer, generic hardware, and generic commands to implement common methods.

The one case in which a software patent's validity was upheld involved website design, specifically a method of generating a composite webpage. The court distinguished this patent from others by noting that it aimed to solve a specific computer-related problem, namely, "a way to automate the creation of a composite webpage."

Looking at how the Federal Circuit made these decisions, one takeaway is to avoid writing patent claims that describe using a generic computer, generic hardware, and generic commands to implement commonplace methods. Such claims are likely to be found ineligible. Instead, consider claims that describe modifying a computer's operation in response to certain conditions. Some district court decisions seem to endorse this approach, finding processes that reallocate computer memory or improve computer performance to be eligible for a patent.

THE DISMISSALS

Typically, a patent case gets to the Federal Circuit after many months, if not years, of litigation in district

court. Invoking *Alice* early in a patent infringement lawsuit might short-circuit some of this litigation and end it more quickly.

One way to resolve a lawsuit before going to trial is to file a motion for summary judgment. That motion asks a court to look at the facts as they currently stand and rule whether one or more issues in the case can be decided by the judge—not a jury—without further delay. Some summary judgment motions can resolve an entire case, while others might address only parts of the case. Summary judgment motions help limit the number of issues that need to go to trial, making it easier for the jury to decide the case and reducing litigation cost.

Some parties who have been sued for infringement of patents that cover computerized implementations of basic processes have relied on *Alice* to file summary judgment motions that the patents are invalid. These motions are typically filed early in the case before significant costs are incurred. District courts appear willing to consider and rule on these motions promptly, probably because it gives the courts the opportunity to quickly dispose of cases that are based on questionable patents.

A patent owner whose infringement case is ended in district court by a ruling in favor of the accused infringer's summary judgment motion can appeal that result to the Federal Circuit. But as described above, the statistics don't bode well for the patent owner. In addition, the Federal Circuit has generally approved of filing early summary judgment motions based on *Alice* by stating that it's unnecessary to wait to file until after the district court has construed the claims, which usually occurs much later in the lawsuit.

Early summary judgment motions invoking *Alice* are a potent weapon that an accused infringer might be able to use. When confronted with a patent infringement lawsuit, consulting with your attorney will help to determine whether this tactic is appropriate.

Alice is a recent decision, and many other cases are likely to follow in which the lower courts will interpret and apply that decision under myriad factual situations. We still don't have a clear answer on software patentability, but the Federal Circuit has given us solid guidance on how to adhere to the Supreme Court's requirements.

BRIAN M. GAFF is a senior member of IEEE and a partner at the McDermott Will & Emery, LLP law firm. Contact him at bgaff@mwe.com.

This article originally appeared in Computer, *vol. 48, no. 7, 2015.*

Editor: **Cesare Pautasso**
University of Lugano
c.pautasso@ieee.org

Editor: **Olaf Zimmerman**
University of Applied Sciences
of Eastern Switzerland, Rapperswil
ozimmerm@hsr.ch

Software Process versus Design Quality

Tug of War?

Girish Suryanarayana, Tushar Sharma, and Ganesh Samarthyam

"Something smells rotten in the state of our design." This realization might come despite all good intentions behind choosing and following the "right" process. Don't underestimate a process's inadvertent effects on the resulting software design's quality, as evidenced in two insightful stories from Girish Suryanarayana, Tushar Sharma, and Ganesh Samarthyam that are based on their recently published book on design smells. Enjoy! —*Cesare Pautasso and Olaf Zimmermann, department editors*

SOFTWARE LIFE-CYCLE PROCESSES provide a structured, disciplined way to guide the development of complex real-world software.[1,2] These processes can be primary (acquisition, supply, development, operation, and maintenance), supporting (for example, documentation, configuration management, quality assurance, reviews, audits, or problem resolution), or organizational (management, infrastructure, improvement, and training).[3] Our interests lie in software design, so we pose this question: Do software life-cycle processes (henceforth simply called software processes) benefit software design?

The answer is a clear yes! For instance, a software process that recommends periodic architecture and design reviews to ensure the design quality, and supports traceability between the requirements and design elements to ensure the design's completeness, helps ensure high-quality design.[4,5]

However, our experience with design smells in real-world projects and interviews with software engineers from various organizations[6] have revealed a paradox. Sometimes, a design exhibits smells because a software process (or combination of processes) has inadvertently become a significant hindrance to high design quality, thus negating the benefits the process was meant to deliver. In some cases, a process has actually undermined design quality. In these cases, an approach that aims to address the design smells and improve the design quality can't merely rely on tactical refactoring of the design artifacts. It also must refactor the process, remove it, or introduce another process.

2376-113X/15/$31.00 © 2015 IEEE Published by the IEEE Computer Society

THREE SOFTWARE DESIGN SMELLS

Here we look in more detail at the three design smells mentioned in the main article. For more on them and other design smells, see *Refactoring for Software Design Smells*.[1]

DUPLICATE ABSTRACTION

This design smell has two forms. *Identical name* is when two or more abstractions have identical names. *Identical implementation* is when two or more abstractions have semantically identical member definitions, but the design hasn't captured and used those implementations' common elements.

INSUFFICIENT MODULARIZATION

This smell arises when an abstraction hasn't been completely decomposed and a further decomposition could reduce its size, implementation complexity, or both. This smell has two forms. *Bloated interface* is when an abstraction has many members in its public interface. *Bloated implementation* is when an abstraction has many methods in its implementation or has one or more methods with excessive implementation complexity.

MULTIPATH HIERARCHY

This smell arises when a subtype inherits both directly and indirectly from a supertype, causing unnecessary inheritance paths in the hierarchy. This complicates the hierarchy and increases developers' cognitive load, thus reducing the hierarchy's understandability. Furthermore, developers might overlook existing implementations on the redundant paths and try to provide their own implementation for the realized interface. In this process, they could provide a considerably different implementation (or no implementation). Such mistakes can lead to run-time problems. So, this smell can impact reliability.

Reference

1. G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*, Morgan Kaufmann, 2014.

A key insight from these cases is that software processes and design quality are inextricably intertwined. To highlight this interplay, we examine the following two real-world cases.

A Suboptimal Change Approval Process

In a globally distributed software development project, a central team (Team A) owned the code base. Team A included domain experts who had originally designed the software. The development was offshored to another team (Team B).

When an external consultant analyzed the code base after the release of the software's first version, he found smells in the design and code. For instance, several classes suffered from the identical-implementation form of the Duplicate Abstraction smell (see the sidebar). Surprisingly, Team B was aware of the extensive code duplication but had made no effort to refactor the design.

The consultant discovered that the lack of refactoring was due to the process the project followed. To prevent unwarranted modifications that could negatively impact the product's functionality, the project relied on a stringent process to control source code changes. Team A had to review any code change made by Team B before the change could be approved. This review focused on functional correctness (from a domain perspective), not the code's structural quality. This change approval process was long and arduous and required multiple emails and telephone interactions between the teams.

To reduce the time to ratify changes that slowed the development rate, Team B wanted to avoid this process as much as possible. Because refactoring (including refactoring to eliminate code clones) would involve only structural changes to the code without impacting the functionality, it seemed logical to avoid refactoring to avoid the change approval process. So, Team B wasn't keen to refactor the source code. In this case, the environment viscosity[7] created by the process led to the software's poor design quality. The consultant shared this finding in his final report and suggested refactoring the change approval process.

The project management could have refactored the change approval process in several ways. For example, they could have incorporated review of code quality into it. Instead, they refactored it to have Team A ratify only new class additions and not every small change that Team B made. The project management be-

lieved this would eliminate the viscosity (by reducing the number of times the change approval process was initiated), which in turn would improve the design quality.

Upon the release of the software's second version, the consultant again analyzed the design quality. Surprisingly, he found many more instances of the Insufficient Modularization smell (see the sidebar), compared to the first release. For example, one class in the source code had approximately 40,000 LOC, and the weighted methods per class (the sum of the cyclomatic complexities[8] of a class's methods) exceeded 2,000.

The consultant found that part of the root cause of the many instances of the Insufficient Modularization smell was still the change approval process. Toward the software's second release, the project management at the off-shore location became concerned about the many bugs (in the order of hundreds) found during testing, which posed a risk for timely delivery. To ensure that the software was released on time, the project management assigned each developer in Team B a target of fixing four bugs every week.

Fearing that they would be considered underperforming if they couldn't fix four bugs per week, the members of Team B explored ways to avoid the change approval process for new classes that bug fixing might require. One easy way to avoid introducing classes was to insert new code in existing classes. Because of schedule pressures, many developers adopted this workaround, which resulted in many Insufficient Modularization instances.

Because this workaround was convenient and the large classes produced no immediate runtime effects, it evolved into a bad habit during development for the third release. It also explained why Team B didn't refactor these large classes immediately after the release. Refactoring them would have required introducing smaller classes, which would have required going through the change approval process.

The project management could have addressed this problem in various ways. For instance, it could have refactored existing processes (for example, refactor the change approval process to remove bottlenecks and improve the turnaround time for change review). Or, it could have removed them (for example, remove the bug-target-setting process so that developers don't bypass change approval). Alternatively, or in combination with the ways we just mentioned, the project management could have introduced a process (for example, introduce a local change approval process to speed up approval).

In this case, for the product's third release, the project management removed the bug targets. They also adopted a *design quality gate* process that required each developer to run a set of design analyzers on the portion of code he or she had modified, before checking-in the code. This helped address the problem significantly, and the number of smells drastically decreased during the third release.

An Ineffective Design Communication Process

Industrial software systems often create complex domain objects to fulfill complicated business requirements. The initialization of such objects typically involves a sequence of steps, including preinitialization and postinitialization. These two steps are crucial, and software developers must remember to write an implementation for them. To ensure this, a common practice is to create an interface that encapsulates them and require developers to realize this interface in the classes corresponding to the domain objects.

> Software design quality is a function of the effectiveness of the followed process in a given context.

In this context, we share an anecdote in which one of us helped develop an application for creating visually attractive user interfaces, using the concept of gadgets. Figure 1a shows a fragment of the application design wherein a `TextGadget` class extends its parent class `GadgetBase` and realizes an `ISupportInitialize` interface. This interface contains two methods, `preInitialize()` and `postInitialize()`, that must be defined by `TextGadget`.

Over time, the need arose to support multiple gadgets such as `GraphicGadget` and `NumericGadget`. The team realized that the implementation for `preInitialize()` and `postInitialize()` remained similar across gadgets. So, the team decided that instead of each gadget separately realizing `ISupportInitialize`, `GadgetBase` could itself

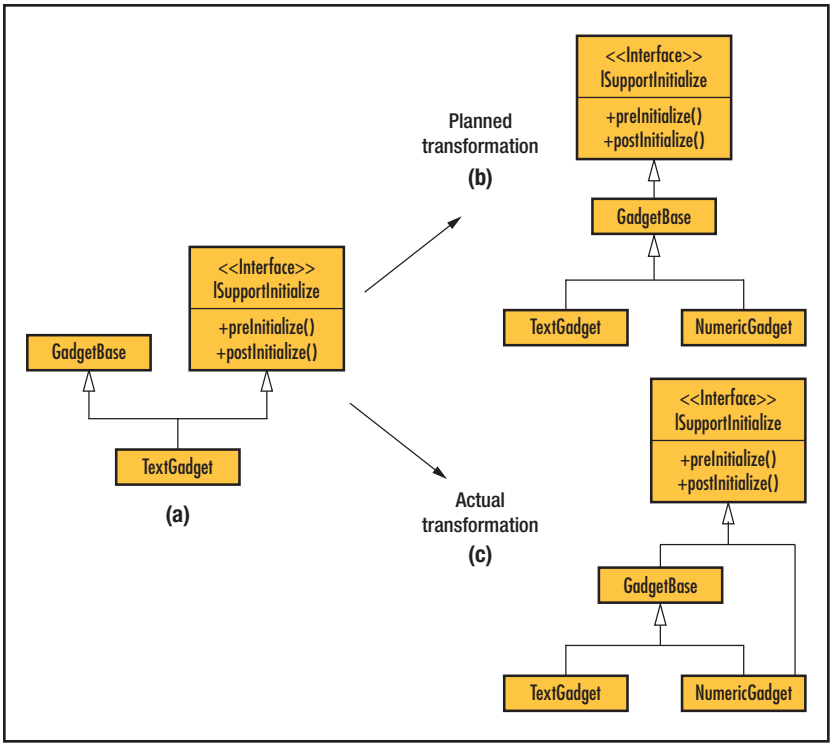


FIGURE 1. Planned versus actual transformation in the **TextGadget** class hierarchy. (a) A fragment of the original application design wherein **TextGadget** extends its parent class **GadgetBase** and realizes an **ISupportInitialize** interface. (b) Instead of each gadget separately deriving from **ISupportInitialize**, **GadgetBase** realizes **ISupportInitialize** and provides the default implementation in the planned refactoring. (c) However, in the realized design, **NumericGadget** extends **GadgetBase** and unnecessarily realizes **ISupportInitialize**. This design fragment suffers from the Multipath Hierarchy design smell.

realize **ISupportInitialize** and provide the default implementation of **preInitialize()** and **postInitialize()** (see Figure 1b).

However, the developer entrusted with implementing **NumericGadget** was on leave when the rest of the team discussed this new design. By the time he returned, the old design had been refactored and the new design was in place. Unfortunately, no one told him about this design decision.

This situation occurred because the project followed an agile methodology and subscribed to the principle that individuals and interactions are more important than detailed documentation.[9] Specifically, the team discussed and communicated design decisions during stand-up meetings. It explicitly documented only the architectural design decisions[10] (such as introducing a new layer or changing the middleware being used) in the architecture specification. In this case, the team considered that having **GadgetBase** directly implement **ISupportInitialize** wasn't an architectural decision, so the team didn't explicitly document this design change.

Because the developer was unaware of the noncritical design decisions, he implemented **NumericGadget** using the old paradigm. That is, **NumericGadget** extended **GadgetBase** and realized **ISupportInitialize** (see Figure 1c). The resulting design thus suffered from the Multipath Hierarchy design smell (see the sidebar).

In short, this design smell arose because the process used to communicate design decisions and changes to all the team members wasn't effective. A more potent process aligned with the agile methodology would have employed multiple communication modes to convey design decisions. For example, it would have additionally used emails or lightweight knowledge management systems such as wikis to document all design decisions so that they were always available to the entire team.[9]

These two cases highlight the process–quality paradox: software processes are designed to bring discipline to software development and intend to help achieve and maintain high-quality software design. However, some software processes (because of how they're implemented or the project conditions) turn out to be cumbersome or porous, leading to situations that can decrease design quality. Software design quality is a function of the effectiveness of the followed process in a given context. So, such situations require us to introduce, tune, or refactor existing processes to achieve and maintain high design quality. In conclusion, these cases lead to the following insights.

First, all the relevant stakeholders need to recognize the interplay between software processes and design quality.

Second, development teams must periodically evaluate design quality (for instance, by looking for design smells). If the quality is poor, teams must determine whether any software process is the cause.

Finally, if the cause of poor design quality is process-related, teams can address it by refactoring or removing an existing process or introducing a new one, as we mentioned before. Refactoring a process might include identifying and removing the obstacles that directly or indirectly hamper good quality.

References

1. N.S. Potter and M.E. Sakry, *Making Process Improvement Work: A Concise Action Guide for Software Managers and Practitioners*, Addison-Wesley, 2002.
2. D. Damian et al., "An Industrial Case Study of Immediate Benefits of Requirements Engineering Process Improvement at the Australian Center for Unisys Software," *Empirical Software Eng.*, vol. 9, nos. 1–2, 2004, pp. 45–75.
3. *IEEE/EIA Std. 12207-2008—ISO/IEC/IEEE Standard for Systems and Software Engineering—Software Life Cycle Processes*, IEEE, 2008.
4. M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.*, vol. 38, nos. 2–3, 1999, pp. 258–287.
5. G. Samarthyam et al., "MIDAS: A Design Quality Assessment Method for Industrial Software," *Proc. 2013 Int'l Conf. Software Eng.* (ICSE 13), 2013, pp. 911–920.
6. G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*, Morgan Kaufmann, 2014.
7. R.C. Martin, *Agile Software Development: Principles, Patterns, and Practices*, Addison-Wesley, 2003.
8. T.J. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.*, vol. 2, no. 4, 1976, pp. 308–320.
9. M. Paasivaara, S. Durasiewicz, and C. Lassenius, "Using Scrum in Distributed Agile Development: A Multiple Case Study," *Proc. 4th IEEE Int'l Conf. Global Software Eng.* (ICGSE 09), 2009, pp. 195–204.
10. O. Zimmermann, "Architectural Decisions as Reusable Design Assets," *IEEE Software*, vol. 28, no. 1, 2011, pp. 64–69.

GIRISH SURYANARAYANA is a senior research scientist at the Corporate Research and Technologies Center, Siemens Technology and Services Private Ltd., India. He's a member of the *IEEE Software* advisory board. Contact him at girish.suryanarayana@siemens.com.

TUSHAR SHARMA is a technical expert at the Corporate Research and Technologies Center, Siemens Technology and Services Private Ltd., India. Contact him at tusharsharma@ieee.org.

GANESH SAMARTHYAM is an independent consultant and a corporate trainer. Contact him at ganesh.samarthyam@gmail.com.

This article originally appeared in IEEE Software, *vol. 32, no. 4, 2015.*

Editors: Konrad Hinsen, konrad.hinsen@cnrs-orleans.fr | Matthew Turk, matthewturk@gmail.com

The Approximation Tower in Computational Science: Why Testing Scientific Software Is Difficult

Konrad Hinsen | Centre de Biophysique Moléculaire in Orléans

Science is all about simplification. We can't describe the whole universe down to the level of elementary particles, and even if we could, this would probably not help us much in understanding the world around us. For every phenomenon that we wish to understand, finding the right level of description is crucial. When studying the dynamics of the solar system, we could describe the Earth as a point mass. But to understand the climate, we need a much more detailed description that takes into account the roles of land masses, oceans, and various layers of the atmosphere.

For every simplification we make, we should try to find a justification, in terms of physical or mathematical arguments, and ideally a way to validate our assumptions quantitatively. Methodologically, it's useful to distinguish between two kinds of simplification: *idealizations*, which are assumptions that can't be justified in any other way than by the ultimate success or utility of a scientific model, and *approximations*, which involve the deliberate choice of a description that isn't the most accurate one available. Describing an electron as a point mass is an idealization: there's no justification for using a more complicated description, given our current state of knowledge about electrons, although we might suspect that there is more to an electron's "true nature" than the point mass description suggests. On the other hand, describing the Earth as a point mass that moves around the sun is an approximation: we

 Published by the IEEE Computer Society

know enough about the Earth's size, shape, and mechanical properties to make a more accurate description.

Computational science has its own specific approximations as well as techniques for justifying and validating them. Like the physical and mathematical approximations that scientists have used for centuries, computational approximations have the goal of converting an intractable problem into a tractable one. The gain in tractability covers both the practical aspect of being able to do a computation in a reasonable amount of time and within the limits of available computing resources and the epistemological aspect of using computational methods that we can understand and reason about, although those two goals are often in conflict. In the practice of scientific programming, approximations are closely related to the subject of testing, which is frequently discussed in this department,[1] and to the issue of performance, which is an important motivation for making approximations and a criterion for evaluating them.

In the following, I use as an example a numerical simulation of the solar system (see Figure 1). The level of description chosen here includes the sun and the planets, but ignores smaller bodies such as moons, asteroids, or comets. This is representative of the very first approximation made in any scientific model: define a system of interest and pretend that the rest of the universe doesn't exist. In addition to ignoring small bodies, we'll ignore the rest of our galaxy and all the other galaxies, which are very far away. The physical justification for both of our simplifying assumptions is the fact that gravitational forces are small for bodies that have a small mass or that are very far away. We could verify the impact of neglecting small bodies by running a larger simulation that includes some of them, say, the moons. The impact of neglecting the rest of the universe is much more difficult to verify explicitly because we don't have enough observational data to construct a good model, but we could still add stars at hypothetical positions to get at least an idea of the order of magnitude of the changes.

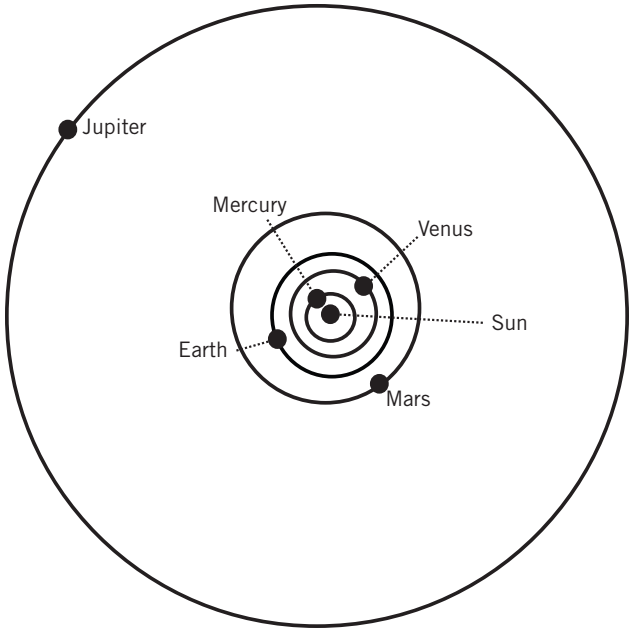


Figure 1. The sun and the five inner planets of the solar system, represented as point masses. This drawing describes the level of detail chosen in the simulations discussed here. It ignores small bodies (moons, asteroids, comets) as well as the whole universe outside of the solar system.

Tower of Power

Before discussing how and why approximations matter for writing and testing scientific software, we need to look one by one at the approximations required to obtain a working numerical simulation of the solar system. Figure 2 shows the complete "tower of approximations," with physical reality as the ground truth at the bottom and numerical results on top.

After defining the system of interest, the next approximation we make is the reduction of each celestial body to a point mass. This step has both a physical and a mathematical justification. Physically, each celestial body has a diameter that's much smaller than its distance from any other body, so shifting all the mass of the body to its center shouldn't make much of a difference. Mathematically, we can show that any spherically symmetric body behaves externally like a point mass. The point mass approximation is thus no more than an assumption of spherical symmetry. Real planets don't have spherical symmetry, so we should think about quantifying the impact of this approximation. In principle, this can be done by using a more detailed description for each celestial body. Finally, our choice of a point mass model means that we can't study each body's rotation about its axis, which limits the range of applications for our model.

As a side note, Figure 1 makes some approximations specific to the task of visualization: it's a 2D projection, out of technical necessity; it omits the outer planets, whose inclusion would require a very big drawing; and it approximates our model's point masses as circles of finite size. The circles all have the same size, so they can't be considered a representation of each body's physical extent. The finite size of the circles is here an approximation to the ideal zero-area point of geometry, made to render the point visible.

At this stage, we can write down a complete mathematical model for the solar system based on the assumption (idealization) that each point mass i moves on a continuous trajectory in space that can be described by a vector-valued function of time $\mathbf{r}_i(t)$. These N functions for N point masses are determined by Newton's laws of motion:

$$\frac{d}{dt}\mathbf{r}_i(t) = \mathbf{v}_i(t) \qquad \mathbf{v}\text{: velocity}$$

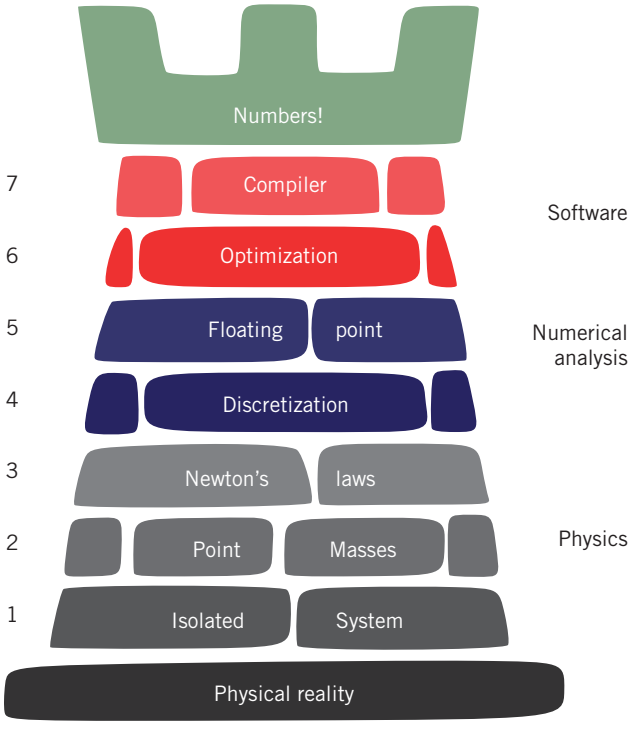


Figure 2. The tower of approximations that must be applied to obtain numbers in a simulation of the solar system. Physical reality is the ground truth at the bottom and numerical results are on top.

$$\frac{d}{dt}\mathbf{v}_i(t) = \mathbf{a}_i(t) \qquad \mathbf{a}\text{: acceleration}$$

$$\mathbf{F}_i(t) = m_i\mathbf{a}_i(t) \qquad \mathbf{F}\text{: force, } m\text{: mass.}$$

The forces $\mathbf{F}_i(t)$ are given by Newton's law of gravitation:

$$\mathbf{F}_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} \mathbf{F}_{ij}$$

$$\mathbf{F}_{ij} = -G\frac{m_i m_j}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^2}\frac{\mathbf{r}_i - \mathbf{r}_j}{\left|\mathbf{r}_i - \mathbf{r}_j\right|},$$

where G is a universal constant. When Newton wrote down this model for the solar system, it wasn't an approximation but an idealization. From Einstein's work on the special and general theory of relativity and the subsequent verification of these theories, we know that Newton's theory isn't the last word, and that we can consider it an approximation of Einstein's theory for the case of slowly moving bodies having small masses.

To apply Newton's model to the solar system, we need to determine values for several parameters: the universal constant G, the masses m_i, and a set of orbital parameters, for example the positions and velocities of each body at some time t_0. Obtaining values for these parameters from available astronomical observations is an inference problem that I won't discuss here—let's just assume we have them. Mathematical analysis shows that these parameters define the trajectories $\mathbf{r}_i(t)$ uniquely for all times t, both before and after t_0. However, while the methods of calculus yield a proof of existence and uniqueness for the trajectories, they don't provide a recipe for actually obtaining numerical values.

The fundamental reason for the impossibility of getting numbers out of differential equations such as Newton's laws is the impossibility of computing with real numbers. A real number contains an infinite amount of information that can't be represented by a finite number of bits. Therefore, the best we can ever hope to obtain is some approximation with finite precision. This isn't a problem by itself: astronomical observations have finite precision as well. However, the trade-off between the precision and accuracy of the results and the computational resources required to obtain them is a complicated subject involving numerical analysis and the theories of computability and computational complexity.

A quick reminder of the definitions: precision refers to the number of digits of a result that are computed, whereas accuracy measures how close a computational result is to the true (usually unknown) value of the same quantity. High precision is a necessary but insufficient condition for high accuracy: it's possible and even common to have a result that's very precise but not at all accurate. The difficult issue in numerical work is accuracy—precision is merely a technical detail.

Simulating the Solar System

Ideally, our numerical simulation of the solar system would allow us to request the positions of all planets at time t and with accuracy e, say at most 1 km from the true positions. It's a little-known fact that this is actually possible, though not practically feasible for our problem, because of the enormous computational resources that would be required. A branch of mathematics called *computable analysis*[2] asserts that under certain not very restrictive conditions, solutions to initial-value problems are computable to arbitrary accuracy. Such results are obtained by reformulating analysis in terms of computable real numbers instead of the full set of real numbers. The computable reals are a subset of the real numbers that includes the rational numbers, the algebraic numbers, and many other numbers relevant to geometry and physics. By definition, a real number is computable if there's an algorithm that returns a rational approximation that differs from the true value by no more than a specified limit.

The representation of a computable real in the computer is the algorithm itself.[3] For example, the representation of π is a procedure $\pi(n)$ that takes an

In theory, program transformations shouldn't change a program's result in any way. However, both compilers and human software developers think they have the right to re-arrange floating-point operations, even if this changes the program's output.

integer argument n and returns an integer p such that $|\pi - 2^{-n}p| < 2^{-n}$, that is, $2^{-n}p$ is an approximation of π with a precision of n binary digits and an accuracy given by the value of the last binary digit (I use base 2 here for simplicity; other choices are possible). It's possible to perform arithmetic and other computations on such numbers. For example, given two procedures $a(n)$ and $b(n)$ representing the numbers a and b, their sum $c = a + b$ is represented by $c(n) = 1/4[a(n + 2) + b(n + 2)]$. Note that a and b must be computed to higher precision to guarantee the error bound on the sum. Similar precautions must be taken for all computations with computable reals. Consequently, computing the result of a complex operation to five digits can require hundreds of digits in the inputs. For this reason, computable real numbers can in practice only be used for simple operations. An online calculator using computable reals (www.hboehm.info/new_crcalc/CRCalc.html) is a good way to explore their possibilities.

With the next approximation step, we enter the domain of numerical analysis (level 4 of the tower in Figure 2). The differential equations are discretized into finite-difference equations of some kind. The continuous time t is replaced by a series of discrete values t_n, and time derivatives are replaced by the differences of the respective quantities at neighboring points in time. This approximation removes the limit-to-zero operations that are implicit in the use of derivatives. The former differential equations become algebraic equations that can be rearranged to express $\mathbf{r}_i(t_n)$ in terms of quantities for $t_k < t_n$, permitting an iterative computation along the time axis. A large number of discretization schemes differ essentially in the choice of the discrete time values and in the approximation of derivatives by finite differences.

Discretization leads to algebraic equations, which are still relations between real numbers. Numerical analysts can make general statements about the stability and convergence of such equations, but it's still impossible, for problems of realistic size, to compute results or judge their accuracy. Practically useful numerical algorithms are obtained only by introducing yet another approximation: rounding. This consists of reducing the precision of intermediate and final results to some constant value independent of t. The most common approach is the use of floating-point arithmetic,[4] in particular, as defined by the IEEE standard 754, whose single- and double-precision binary formats are directly supported by many of today's processors. The impact of the use of floating-point arithmetic is often difficult to estimate in practice.

Numerical analysis can shed some light on the effect of small mistakes, such as those introduced by rounding, but only locally—that is, for a single integration step. The same is true in general for the error introduced by discretization. However, the errors due to the lower levels of the approximations tower aren't fundamentally different. We can estimate by how much the forces on each planet are changed by ignoring the moons in our simulation, but this doesn't tell us by how much the planetary orbits over a year are modified by this approximation. In fact, the long-time impact of any small change, be it in the forces or in the computed positions, depends not only on the approximations we make, or even on just the equations, but on the parameters on the specific system we want to study. Newton's equations have both stable solutions, such as planetary orbits, and chaotic solutions for systems with colliding or nearly colliding bodies. The latter are unpredictable by computation beyond very short time spans. The very distinct behavior of these two kinds of physical systems is explained by the Kolmogorov–Arnold–Moser theorem, which operates at level 3 of our tower, meaning that it isn't related to computational issues at all.

Our recourse to numerical analysis (levels 4 and 5) has forced us to abandon our original specification for the solar system simulation. We can no longer write a program that computes positions at a given time to a given accuracy. We have to select the accuracy-related parameters (discretization time step, floating-point precision) upfront and try to evaluate their impact on the planetary orbits from the numerical calculations. This is typically done by comparing the orbits obtained for different parameter values and aiming for convergence within the desired accuracy.[5]

Testing

At this point, you're entitled to believe that we've arrived at the top of the approximation tower, but there are still two more floors to go: program transformations. The first level contains transformations performed manually by a software developer, usually for performance. Parallelization falls into

this category. The second level is about the transformations automatically performed by compilers in the process of optimization. This level of approximation is special in that it's the only one in the whole tower that isn't under a programmer's full control.

In theory, program transformations shouldn't change a program's result in any way. However, both compilers and human software developers think they have the right to re-arrange floating-point operations, even if this changes the program's output. An annoying consequence is that the result of a numerical computation can differ between two computers or even between two runs on the same (parallel) computer if a different number of processors is used,[6] in spite of 100 percent identical source code. In fact, most popular programming languages don't clearly specify what the result of floating-point expressions should be, giving software developers no effective way to implement a numerical algorithm exactly.

This is where the approximation tower becomes relevant for software testing. It's often said that scientific computing is a particularly difficult application domain when it comes to testing. The specificity comes from the heavy use of floating-point arithmetic, and its main cause is the top floor of our approximation tower. The theory of software engineering says that tests verify that a program computes what its specification says it should compute,[7] which assumes that there exists a specification, and that it corresponds to the last approximation step. But when floating-point operations enter the game, the last step is out of our control and we can't write a specification for it.

Scientific software developers do write tests, however, so what do they do? First, most scientific software contains significant parts that don't rely on floating-point operations and that aren't affected by the approximation tower. For the numerical computations, one popular approach is downright cheating. You write your program, you check it using informal means (until the results "look right"), and then you write some unit tests that check for the results you got on your machine within some arbitrary error margin. But this doesn't test your program at all. All you verify is that the parts outside of your control—the compiler and pre-installed dependencies—are similar enough to those on your development machine. You also have some chance that a bug introduced during a future modification will be caught if it changes the results beyond the error margin in your tests. Therefore, this fake test approach isn't entirely useless, but it shouldn't be called testing.

Another popular approach is to compute a result for which an exact reference value is known from analytical calculation. Instead of simulating the whole solar system, you run a simulation on a two-body subsystem (say, sun and Earth), for which the exact orbit is known to be an ellipse. You compute the parameters of the ellipse from your input data, and then you compute a few points on this ellipse and compare with your simulation. This is clearly a much better approach than fake tests because you compare positions obtained using very different methods and thus different software. Of course, the so-called exact result isn't really exact. It shares the first three levels of the approximation tower and is also subject to levels 5 and 7, assuming that you avoid level 6 by not optimizing anything in the test code. So what you're testing with such a comparison is a combination of your simulation code, the discretization of Newton's equations, and the test code. You can't really expect to get identical results down to the last bit, so you have to introduce some tolerance into the tests. Estimating a reasonable tolerance is very difficult, so most developers don't try and make up some value that makes the tests pass on a few machines.

Could we do better? I believe that we can, but much work remains to be done. A big part of the problem is the program transformations introduced by compilers because they're out of our control. This level of the tower is also unique in not being technically justified. A reasonable programming language should allow a software developer to specify the exact order of floating-point operations, and a compiler shouldn't modify it in any way, at least for the most basic optimization level. If today's languages and compilers try to outsmart the programmers, this is only because the latter accept the situation. And this, in turn, is mainly due to the aura of mystery and general fuzziness surrounding floating-point arithmetic.

Getting rid of the last level of the approximation tower wouldn't magically solve all problems, of course. We would gain the possibility of writing specifications at level 5 and test our programs against them. We could also evaluate the impact of explicit optimizations (that is, level 6) by comparing to the specification. However, moving from level 4 to level 5—making the step from discrete equations for real numbers to an algorithm for floating-point numbers—would still be a difficult task. Programmers would have to make explicit choices for the order of all floating-point operations, which they aren't prepared to make because today's tools don't let them. Moreover, such choices should have a rational basis. But we can't say which floating-point algorithm yields the best approximation to the solution of our discretized equations if we can't compute the latter. This is where computable reals could find their role in computational science. Even if we can't perform a long-time simulation of the solar system using computable reals, we can do a few steps of integration that serve as a reference for fine-tuning a floating-point algorithm.

The analysis I've described here is specific to a simulation of the solar system, but most aspects remain valid for other tasks in scientific computing. Levels 2 and 3 of the tower are the mathematical model for the problem being studied. This model is usually taken as given in computational science, although it also happens that the initial mathematical model is approximated to facilitate computation. Level 4 is the step that replaces continuous functions by a discrete and finite set of values. The two main approaches are discretization and expansion of the solutions in terms of a truncated basis set. The latter category includes Fourier and wavelet transforms, normal modes, and so on. The truncation offset is then the accuracy-related parameter that replaces the discretization time step in the above example. Levels 5 to 7 of the tower, and thus the problems they create, remain the same in all numerical methods. ■

References

1. P.F. Dubois, "Testing Scientific Programs," *Computing in Science & Eng.*, vol. 14, no. 4, 2012, pp. 69–73.
2. M.B. Pour-El and J.I. Richards, *Computability in Analysis and Physics*, Springer, 1989.
3. V. Ménissier-Morain, "Arbitrary Precision Real Arithmetic: Design and Algorithms," *J. Logic and Algebraic Programming*, vol. 64, no. 1, 2005, pp. 13–39.
4. N. Toronto and J. McCarthy, "Practically Accurate Floating-Point Math," *Computing in Science & Eng.*, vol. 16, no. 4, 2014, pp. 80–95.
5. T. Boekholt and S. Portegies Zwart, "On the Reliability of N-Body Simulations," *Computational Astrophysics and Cosmology*, vol. 2, 2015, p. 2.
6. K. Diethelm, "The Limits of Reproducibility in Numerical Simulation," *Computing in Science & Eng.*, vol. 14, no. 1, 2012, pp. 64–72.
7. K. Hinsen, "Writing Software Specifications," *Computing in Science & Eng.*, vol. 17, no. 3, 2015, pp. 54–61.

Konrad Hinsen is a researcher at the Centre de Biophysique Moléculaire in Orléans (France) and at the Synchrotron Soleil in Saint Aubin (France). His research interests include protein structure and dynamics and scientific computing. Hinsen has a PhD in theoretical physics from RWTH Aachen University (Germany). Contact him at konrad.hinsen@cnrs-orleans.fr.

This article originally appeared in
Computing in Science & Engineering, *vol. 17, no. 4, 2015.*

EDITOR **DAVID ALAN GRIER**
George Washington University; grier@gwu.edu

This article originally appeared in Computer, *vol. 48, no. 7, 2015.*

Do We Need a Software Czar?

David Alan Grier, George Washington University

We need to temper authoritarian leadership with the insights of individual programmers.

In my travels through the world of technology, I've met many people who believe that software has gone astray and that we desperately need to take action to put things right. My colleague Curtiss is one such person. He believes that our efforts to produce open, flexible, and reusable software have utterly failed. Instead of developing a body of software that's getting better and better, he says we've been creating programs that are brittle, incomprehensible, and nearly impossible to change. Our efforts to solve this problem with open standards and the free flow of information have been misguided. According to him, we need to appoint a software czar, someone vested with absolute authority to bring some order to the field.

I suspect that Curtiss would like to be that software czar. I also suspect that he'd like me to help him get this position. I acknowledge some of the flaws he sees in software, but I'd argue that we need to be more subtle in our leadership of software projects. We need to temper authoritarian leadership with the insights of individual programmers.

We start designing software by drawing boxes on a whiteboard that represent different system modules, and then we divide our programming staff into groups that correspond to those boxes. Each of these groups is responsible for their own code. They can't concentrate on their assignment if they have to know every detail about every other module. Yet, the quality of the system will be determined by how well those groups and the modules they create work together.

> The connections between software modules are more than technical specifications.

Pioneering computer scientist David Parnas famously observed that the connections between software modules are more than technical specifications. These connections are assumptions that the modules make about each other—or, more accurately—assumptions that programming groups make about the work that other programming groups are doing.

Although the free exchange of information often helps clarify assumptions, it can also have side effects. It can allow individual programmers to impose their ideas upon others. Programmers can find unintended uses for libraries, exploit coincidences in structures, and conclude that data follows a pattern where none exists. Through any of these actions, programmers can introduce an unanticipated assumption into the system.

Over the past few decades, we've increasingly used open, market-based techniques to manage software projects. However, markets don't anticipate change well, and open information can hide important facts. Software development projects can't be successful without some form of leadership, which helps a programming team decide which ideas will produce a strong, flexible system and which won't. Many developers find it difficult to provide that judgment, as it's easy to let a project drift from one extreme to another by either dictating every detail of the system or by accepting every contribution from every programmer.

One successful CTO claims that his software teams produce the best work when his leadership style is closer to the role of a judge. "If you want to make any change to the plan," he told his team, "you have to bring it to me and convince me that it's a good change. If you believe the idea isn't worth bringing to me, then it isn't worth putting in the system." However, this role isn't as easy or as passive as he makes it sound. He gets his team together to make sure that information about the design has been thoroughly reviewed and is understood by all. He also reviews the code to make sure the developers are following his direction. He's a strong leader but not the type who would dictate the future of software—he's not a software czar.

See **www.computer.org/computer-multimedia** for multimedia content related to this article.

DAVID ALAN GRIER is the author of *When Computers Were Human* (Princeton Univ. Press, 2007) and the producer of the new podcast series "How We Manage Stuff" (http://itunes.managestuff.net). Contact him at grier@computer.org.

 Published by the IEEE Computer Society

Editor: **Eoin Woods**
Endava
eoin.woods@artechra.com

Architecture Haiku
A Case Study in Lean Documentation

Michael Keeling

SOFTWARE ARCHITECTS ARE taught to create comprehensive, complete documentation. We're taught to stamp out all assumptions and explicitly articulate our designs. Without restraint, however, architecture descriptions can become long-winded treatises on a system's design. In the course of being thorough, we often inadvertently obfuscate the architectural vision in documents that consequently aren't read.

An architecture haiku is a "quick-to-build, uber-terse design description" that lets you distill a software-intensive system's architecture to a single piece of paper.[1] Creating an architecture haiku is more than just an exercise in terse documentation. To write an effective haiku, you must focus on only essential design decisions and rationale. Much like its poetic cousin, an architecture haiku follows strict conventions and can express a mountain of information with a tiny footprint.

What Is an Architecture Haiku?

An architecture haiku aims to capture the architecture's most important details on a single piece of paper. Placing extreme constraints on the architecture description forces architects to focus on the design's most important aspects. With only one page to work with, there simply isn't space to waste on extraneous details.

For an architecture haiku to be comprehensive, Fairbanks recommended that it include[1,2]

- a brief summary of the overall solution,
- a list of important technical constraints,
- a high-level summary of key functional requirements,
- a prioritized list of quality attributes,
- a brief explanation of design decisions, including a rationale and tradeoffs,
- a list of architectural styles and patterns used, and
- only those diagrams that add meaning beyond the information already on the page.

An important key to success is to focus on the most essential ideas for each item on this list. If something isn't important, don't include it. If an idea requires more detail, don't skimp—but don't overdo it. The point isn't to omit or reduce critical information but to highlight it. Removing noise that might otherwise drown out the most important ideas helps shine a spotlight on them.

Architecture haiku creates a *cognitive trigger* that facilitates the recall of essential contextual information about a design decision. Rather than fully describing a decision using comprehensive prose and diagrams, assume the reader already has much of the required knowledge, and focus on the critical details.[2]

Two of the most common examples of a cognitive trigger are architectural styles and patterns. With a single word

2376-113X/15/$31.00 © 2015 IEEE Published by the IEEE Computer Society

or phrase—for example, "layer" or "pipe and filter"—you can reference a wealth of knowledge that doesn't need to be repeated in your documentation. Less formally, architects often use homegrown metaphors[3] or even refer to systems the team has previously built, to the same effect. Any common reference point—such as a whiteboard discussion, a framework, an argument, or even other published material—can be a cognitive trigger that connects an essential decision to the context and background supporting it.[2]

For example, an architecture haiku might provide a "take-away lesson" from a collaborative whiteboard discussion, the details of which might be stored as raw notes and whiteboard photos. In many cases, a couple of sentences or a few bullet points will get the point across. From a team perspective, an architecture haiku should comprehensively describe the whole system for small or routine designs. For large or extremely complex systems, an architecture haiku might describe only a subcomponent for which a team is responsible. Or, it might provide an overview at the highest level to help create a common vision among distributed teams. You might even try creating a haiku of haikus, opting to use single-page descriptions for each viewpoint that comes together to create a comprehensive description. In any case, the objective is the same: record sufficient information so that team members can recall essential design decisions based on previous collaborations or on knowledge from other sources.

Effectively Using Architecture Haiku

While using architecture haiku, my project teams at IBM discovered how to consistently create useful one-page architecture descriptions. My teams were experienced but small—typically, three to five engineers with an average of at least five years' software industry experience. The teams always worked directly with customers, so the customer's architectural experience and documentation requirements often influenced how a team approached design and documentation.

> An architecture haiku aims to capture the architecture's most important details on a single piece of paper.

Although architecture haiku was extremely effective, it wasn't the right tool for every project or situation. Using it effectively requires some thought and practice. Here, I describe our best advice for creating effective architecture haikus.

Start with a Shared Understanding

When we first learned about architecture haiku, we thought it was a great idea—but there were some problems. Although we were well versed in enterprise search concepts, not everyone spoke the same software architecture dialect, and some engineers weren't familiar with certain concepts. For example, some engineers were familiar with the 4+1 view model, whereas others preferred Software Engineering Institute viewpoints, and still others used IEEE terminology. Some engineers talked about nonfunctional requirements, and others discussed quality attributes. We quickly resolved these differences, but not before creating some confusion that was occasionally amplified by the sparse language of a haiku.

Architecture haiku relies heavily on domain-specific vocabulary and cultural norms within a team to build cognitive triggers. So, readers must be educated in basic software architecture concepts to understand implied assumptions in the haiku. This includes a basic understanding of architectural drivers; quality attribute scenarios; the use of structures, architectural styles, and patterns; and how to identify and evaluate design tradeoffs. The team must also agree on a vocabulary.

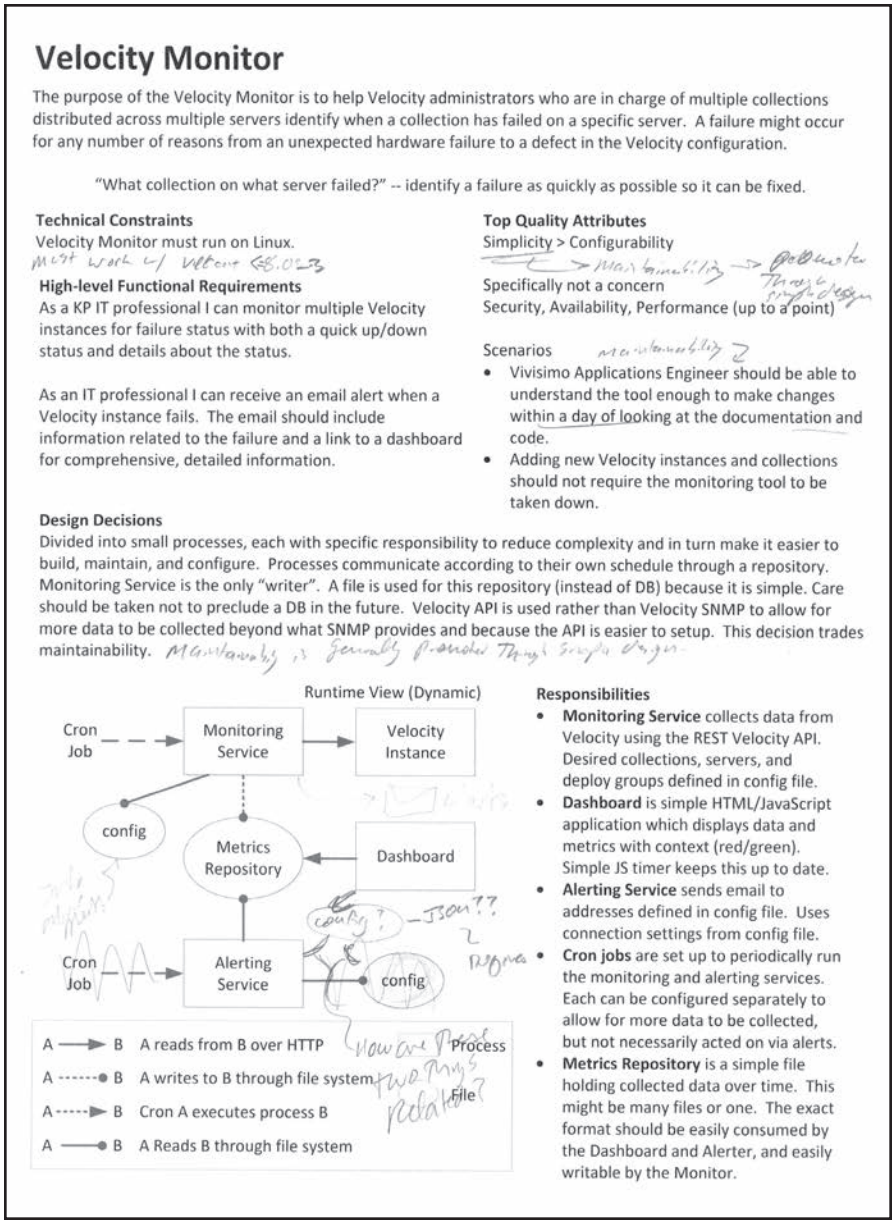
Explore First, Then Record Decisions

Creating an architecture haiku doesn't replace design exploration, nor does it magically transfer knowledge from one teammate to another. A good haiku will act as a medium for recalling essential information about design decisions such as previous discussions and collaborative whiteboard sessions. Without this context, an architecture haiku is just a terse document.

You can create context by collaboratively exploring the design space with stakeholders. Collaborative exploration lets multiple stakeholders walk through the design process and arrive at a shared understanding of the design. Then, the haiku be-

comes a lightweight monument to those creative insights, a reminder of what you decided to build and why.

In our experience, an architecture haiku is excellent for communication and recall but terrible for exploration. The biggest problem we experienced was deadlock. Artificially constraining design activities to a single page too early constricted creativity and didn't leave breathing room to explore alternatives. We found that the best strategy is to explore first—collaboratively when possible—then record decisions in the haiku, updating it as you learn more about the architecture.



Velocity Monitor

The purpose of the Velocity Monitor is to help Velocity administrators who are in charge of multiple collections distributed across multiple servers identify when a collection has failed on a specific server. A failure might occur for any number of reasons from an unexpected hardware failure to a defect in the Velocity configuration.

"What collection on what server failed?" -- identify a failure as quickly as possible so it can be fixed.

Technical Constraints
Velocity Monitor must run on Linux.

High-level Functional Requirements
As a KP IT professional I can monitor multiple Velocity instances for failure status with both a quick up/down status and details about the status.

As an IT professional I can receive an email alert when a Velocity instance fails. The email should include information related to the failure and a link to a dashboard for comprehensive, detailed information.

Top Quality Attributes
Simplicity > Configurability
Specifically not a concern
Security, Availability, Performance (up to a point)

Scenarios
- Vivisimo Applications Engineer should be able to understand the tool enough to make changes within a day of looking at the documentation and code.
- Adding new Velocity instances and collections should not require the monitoring tool to be taken down.

Design Decisions
Divided into small processes, each with specific responsibility to reduce complexity and in turn make it easier to build, maintain, and configure. Processes communicate according to their own schedule through a repository. Monitoring Service is the only "writer". A file is used for this repository (instead of DB) because it is simple. Care should be taken not to preclude a DB in the future. Velocity API is used rather than Velocity SNMP to allow for more data to be collected beyond what SNMP provides and because the API is easier to setup. This decision trades maintainability.

Runtime View (Dynamic)

Responsibilities
- **Monitoring Service** collects data from Velocity using the REST Velocity API. Desired collections, servers, and deploy groups defined in config file.
- **Dashboard** is simple HTML/JavaScript application which displays data and metrics with context (red/green). Simple JS timer keeps this up to date.
- **Alerting Service** sends email to addresses defined in config file. Uses connection settings from config file.
- **Cron jobs** are set up to periodically run the monitoring and alerting services. Each can be configured separately to allow for more data to be collected, but not necessarily acted on via alerts.
- **Metrics Repository** is a simple file holding collected data over time. This might be many files or one. The exact format should be easily consumed by the Dashboard and Alerter, and easily writable by the Monitor.

FIGURE 1. An architecture haiku that we printed out and then annotated as we learned more about the system and refined the architecture.[2]

Treat the Haiku as a Living Document

One benefit of constraining an architecture haiku to a single page is that it can provide guidance in a way that can easily evolve as the team learns more about the system being built. Figure 1 shows the haiku we used to describe the Velocity Monitor, a software tool that monitors the crawling and indexing status of search collections. This tool's scope was relatively small but included several interesting integration points with other systems.

We recorded design decisions in this haiku as brief prose, including rejected choices. It included a diagram of the most informative view of the architecture because the design didn't exploit any architectural patterns.

Note the annotations on the page. As construction began, we learned that some decisions could be improved, so we altered the design accordingly. Updating the haiku was simply a matter of scribbling notes and scratching out elements on the page. Such changes are expected and should be embraced.

Use the Haiku as an Outline for Future Documentation

Using an architecture haiku as a recall mechanism for established rationale, context, and decisions requires that the readers remember information. Although this is usually easy

for project teams, there are risks to consider, such as long-term recall of information, turnover, and project transition.

Refreshing teams owing to turnover or project transition is fairly common. In the former case, engineers might simply no longer be available to consult on a project because they've moved on to new roles or organizations. In the latter case, a project might be transitioned to a new team for maintenance or future work. Over months or years, memories decay, people forget specifics, and the effectiveness of the cognitive triggers captured in an architecture haiku decreases.

The simplest way to mitigate these risks is to write a more traditional architecture description before the team disbands. This documentation aims to capture ideas referenced in the haiku that might exist only as an oral history within the team.

Once you've created an architecture haiku, writing a tight, well-organized architecture description becomes much easier. By creating the haiku first, the team has already identified the essential information, much like an outline. Furthermore, the haiku can serve as an executive summary or be included in the appendix of a longer document. Using the haiku in this way can help achieve balance among agility, delivery speed, and the need for documentation.

Create a Template

This tip might seem obvious, but it's important. To promote communication within the team, our haikus follow the same basic template. This template promotes basic graphic design practices, including the use of white space and alignment. We also don't allow fonts smaller than 10 points. Small fonts are too difficult to read and defeat the purpose by allowing for too much text.

Following a template makes it easier to quickly discern the documentation's salient points and helps provide guidance to the haiku writers. We've distributed our template as two PowerPoint slides: one with the MegaVista example (see the sidebar "Architecture Haiku for the MegaVista Project") and the other with the tips shared in this article. You can access the template at www.neverletdown.net/2015/03/architecture-haiku.html.

It's easy to shrink the font and cram words onto a page. Producing a highly valuable, information-dense resource that people actually want to read is more difficult and much more rewarding.

When I was in high school, my calculus teacher let us bring a single page of notes to the final exam—an officially sanctioned "cheat sheet." I spent weeks preparing it, carefully selecting concepts I thought would be on the exam, including complex equations, proofs, and examples. On exam day, much to my surprise (but not to my teacher's), I hardly used my cheat sheet at all. Creating it was the best study guide I could have used. In many ways, architecture haiku is a cheat sheet for your architecture.

Philippe Kruchten said, "If, instead of a fully robust process, I were permitted to develop only one document, model, or other artifact in support of a software project, a short, well-crafted Vision document would be my choice."[4] The intent of architecture haiku is very much in this spirit.

Architecture haiku helps your team focus on the most essential information relevant to the architecture, provides clear guidance for construction, and encourages collaboration. Combining architecture haiku with traditional documentation is a clear winner. Although architecture haiku in its most literal form is a useful tool, it also offers a chance for general reflection on architecture documentation practices. The number of pages used is less important than designing documentation that's right for your team and situation. Whether you use architecture haiku as the only documentation or simply as a litmus test for how well the architecture is understood, the goal is always to create high-value documentation that maximizes awareness and understanding across the team.

In conclusion,

Design for your team
Less is more when describing
Your architecture.

References

1. G. Fairbanks, "Architecture Haiku," blog, 2011; http://georgefairbanks.com/architecture-haiku.
2. M. Keeling, "Creating an Architecture Oral History: Minimalist Techniques for Describing Systems," presentation at the 8th Software Eng. Inst. Architecture Technology User Network Conf. (SATURN 12), 2012; http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=20330.
3. M. Keeling and M. Velichansky, "Making Metaphors That Matter," *Proc. 2011 Agile Conf.* (AGILE 11), 2011, pp. 256–262.
4. D. Leffingwell and D. Widrig, *Managing Software Requirements: A Unified Approach,* Addison-Wesley Professional, 1999, p. 169.

MICHAEL KEELING is a software engineer at IBM. Contact him at mkeeling@neverletdown.net.

This article originally appeared in IEEE Software, *vol. 32, no. 3, 2015.*

ARCHITECTURE HAIKU FOR THE MEGAVISTA PROJECT

Clearly listing the tradeoffs in the design helped us with decision making during implementation. Because the team was experienced in building search applications, it was sufficient to list the key architectural patterns used in the design. Diagramming was essential when we collaborated around whiteboards before creating the haiku. However, it added little value to the haiku itself, so we didn't include it.[1]

MEGAVISTA GUIDES SEARCH

A publicly consumed search solution based on the Velocity Search platform that will help users find companies of interest.

Business Drivers

- Better search encourages listed companies to purchase advertising, metadata supplements, and paid listings.
- Faster results for users and advanced search options (such as refinement and spelling suggest) bring more users to the site.

Key Tradeoffs

- Crawlability over Maintainability—Need to split the data source across multiple search collections, distributed across several servers. Incidentally promotes Scalability.
- Crawlability and Queryability over Configurability—Highly configurable system reduces speed of crawl and query.
- Flexibility/Development Speed over Cost—Stakeholders are not 100 percent on all features and have a strong desire to go live as soon as possible.
- Modifiability over Maintainability—IT will maintain the system and know databases, not Engine, so when they make a change it should be detected and reflected.

Architecture Styles and Patterns Used

3-tier (data, crawl, query)
Source-Selector—promotes reliability
Query Redundancy—promotes availability
Virtual Documents—all data crawled
Document Enqueue—for website data
Collection Sharding—promotes crawlability
Crawler Clone—promotes availability
Geolocation Lookup—promotes maintainability/modifiability

Top Quality Attributes

Crawlability > Queryability > Scalability

- Crawlability—A batch of up to 900 metadata updates are published to the database and reflected in search within 3 minutes.
- Queryability—An average-sized result set can be calculated within 2 seconds of Engine receiving the query.
- Scalability—The size of the data source increases beyond current capacity, and the system can be easily expanded to deal with this.

Design Decisions with Rationale

- Database must be crawled as multiple views (vice-JOINed) to avoid stressing it too much. Currently 1.6 million companies, each with dozens of metadata fields. Metadata joined via Virtual Document (company ID = vse-key).
- Company website must be crawled and website content made searchable under the company (i.e. return a single results with all content searchable as the same document). Website data joined with metadata via Virtual Document (company ID = vse-key).
- Full recrawl / week, DB refresh / 30 sec. if no crawl running. Full recrawl will pick up changes made to company websites (external data). 30-sec. refresh enables catchup for metadata crawls on DB updated every 60 seconds at most.
- Collections divided by company ID. Dividing allows maximum crawl throughput, using company ID (e.g. collection A has IDs 1–10) allows us to control partitioning.
- Paid listings and keyword weighting factors used for relevancy calculation will come from the database (not Engine configuration). This will allow the business unit to make changes quickly using their existing tools.

Reference

1. M. Keeling, "Creating an Architecture Oral History: Minimalist Techniques for Describing Systems," presentation at the 8th Software Eng. Inst. Architecture Technology User Network Conf. (SATURN 12), 2012; http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=20330.

Editor: **Grady Booch**
IBM, grady@
computingthehumanexperience.com

Then a Miracle Occurs

Grady Booch

DEVELOPING a software-intensive system is like raising a child.

As Will Pleasant-Ryan observed, both babies and code have smells, you get used to changing requirements, you have to take the long view, you can't do it alone, and eventually you have to let go.[1] I would add to Will's most excellent simile that, although you might think you know everything up front, you certainly don't, and therefore you end up discovering a lot of things along the way. Furthermore, in this age of cognitive systems, you also have to both teach them and give them the skills to learn on their own. Finally—as they all probably should anyway—they sometimes turn out very different from what you first expected.

Developing a software-intensive system is like producing a movie.

Tony DeRose of Pixar had much to say about this simile,[2] as did the good folks in the early agile community.[3] Both software development and filmmaking require the skills of a diverse set of stakeholders who must work together intensely for short bursts of time. Talent matters, and as such you must often deal with strong personalities. Although the bones of the delivered artifacts might be clear, opportunities exist for creatively pivoting along the way to tell the right story at the right time in the right way. If you're really successful, you'll be asked to make a sequel, and you might even turn a small indie project into a large money-making franchise.

Developing a software-intensive system is like releasing a new drug.

Pedram Alaedini and his colleagues observed that software development and drug development are "highly complex, rely on multidisciplinary teams, and usually come in late and over budget. In addition, they are often judged based on the same quality factors: reliability, safety, and efficiency."[4]

I would add that in both cases, the stakes are high: your efforts might improve the lives of millions, but you might also cause some deaths along the way.

Developing a software-intensive system is like writing a novel.

J.J. Merelo gave us a riff on this comparison.[5] Both tasks are creative endeavors. Both require collaboration. Both must "find their own voice, but at the same time rely on the tropes of the genre and on plot lines that have been there for millennia" (in the world of software, we call these *patterns*). Both must appeal to their particular audience. Both are full of surprises and contain many little languages. And while they must adapt themselves to the medium, their very presence changes the medium itself.

Developing a software-intensive system is like dating.

The delightful book *Dating Design Patterns*[6] explored this rich

 Published by the IEEE Computer Society

simile. Surprise Statefulness, Interested Listener, Decorated Visitor, and Fifth Position Break are all patterns found in dating and well-structured software.

Developing software is like making a work of art.

Jonathan Wallace provided some interesting insight on the matter.[7] Art springs from the mind of the creator, as does software. Both might be rendered on a literal blank page or might have a patron who dictates certain characteristics. Most art (and software) is bound by the materials at hand, but great art (and great software) creates and defines its own medium. Art builds on the fundamental physics of light, the foundations of color theory, the chemistry of materials, and the cognitive and emotional nature of the human mind. Software builds on logic, mathematical theory, the physics of hardware, and human needs. Both art and software might themselves be works of beauty.

Developing a software-intensive system is like the travels of Lewis and Clark.

If you stay close to home, you might be quite content walking the same path every day. This is the bread and butter of most life and most software. In well-understood domains, the journey is predictable, the risks are known, and the outcomes are generally satisfying.

However, if some hint of exotic lands captures you, you must find a new way. If the rewards are high but the path unmapped and particularly dangerous, rarely will you go on your own. Others must accompany you, especially those with the same spirit of adventure. Although you can equip yourself with the best knowledge and the best equipment, you know that along the way you'll have to rely on your own instincts. Sometimes—quite often, actually—you'll try one direction but have to backtrack and try another. You might lose some people along the way. You might even fail and never make your destination. But if you do succeed, you'll reap the rewards of being the first, and others will soon follow you. If you fail but the journey is still interesting, then you'll have succeeded in showing others part of the way.

If some hint of exotic lands captures you, you must find a new way.

Developing software is like teaching cats to walk in a line.

It's not easy: cats and developers are often quite independent creatures that want to do things their own way, on their own time, with minimal ceremony. But if you offer the right incentives (a nice piece of fish for the cats, the vision of making something Really Cool for the developers), then you'll find them collaborating beyond reason.

Developing a software-intensive system is like building with Legos.

I loved building with Legos as a kid. Back then, you'd get just a box of random blocks, and you'd use your imagination to build whatever came to mind. These days, most Legos come in a kit, often with special parts but always with detailed instructions. On one hand, having standard parts is nice: you don't have to reinvent everything, and you can always fall back on common ways to build certain things. On the other hand, sometimes you have to wedge existing components in unnatural ways to get them to do what you want. The results might be clumsy, but mostly they'll be good enough.

Developing a software-intensive system is like building with Legos and having your own 3D printer to make custom blocks.

The same rules I just mentioned apply, but this time you can craft your own parts. Most will be one-offs. But, if they prove particularly useful, you might find yourself replicating the custom component over and over, to the point where others want to use it.

Developing a software-intensive system is like building a dog house.

Here, the Nike development process is sufficient: Just do it. If the results are good enough, you'll end up with a dry, happy dog. If the results are a failure, you can just tear down the old doghouse and start from scratch or break the old one apart and put it back together in a new way. Failing that, you can always get a new, more accepting dog.

Developing a software-intensive system is like a city growing old.

Your city probably started generations ago as a modest outpost along some interesting terrain. It grew because it was a useful place—economically viable, rich with

This article originally appeared in IEEE Software, *vol. 32, no. 4, 2015.*

natural or human resources—and a place that attracted others to live. Time passes, as it tends to do, and suddenly you find yourself living in a 100-mile city badly in need of new things to keep it viable: a new transit system, an upgraded sewer system, and special services for a changing population. Problem is, no matter what you do you can't just start over (too much legacy), and you can't just apply the Nike development process (too many stakeholders with concerns far more diverse than you could ever imagine). Change takes time, money, and patience. However, if you fail to change, your city will die. Guaranteed.

Developing quality software-intensive systems that matter and systems that endure and make a difference is all of these things.

It's also none of them.

As I reflect back on the history of software engineering, some clear patterns emerge. In the earliest days—the dawn of digital computing in the '30s and '40s—we'd see small teams of systems people (back then, software was so very mixed up with hardware). Personal processes tended to dominate. From the '50s through the '70s, the programming priesthood arose, giving us the dogma of structured methods and waterfall processes. Don't get me wrong: I'm not using these terms in disparaging, emotional ways. They were quite reasonable in their time, given the nature of the software-intensive systems we built. In the '80s, with the rise of personal computing, we saw a split: personal processes were reinvented for these personal things, and larger teams grew (with outsourcing as a new feature) at the other end. By the '90s, the presence of the Internet and the rise of object-oriented languages changed everything, yielding incremental and iterative methods—the precursor to today's agile methods.

From the outside, from the public's viewpoint, developing software-intensive systems looks either very simple or incredibly mysterious. Given the recent emphasis on teaching programming to every K–12 student, you might be left with the impression that everything is just a Simple Matter of Programming. Of course, computing insiders know that this simply isn't true. Similarly—especially to the public fed on a diet of television and movies with incendiary topics, dubious science, and questionable timelines—it would seem that all software happens when you put bright people in a room, give them a lot of venture capital, and then wait until a miracle occurs.

Developing a software-intensive system isn't like that at all, and we, as insiders, have a responsibility to tell the stories of computing in an engaging, truthful way.

One parting thought. The field of software engineering might seem to be largely settled: we think we know the essentials, and everything that might follow is simply a variation on some agility-at-scale best practices.

If you believe that, you are so very wrong.

The future will bring us systems of breathtaking complexity and importance. The advent of machine learning changes how we grow systems because we must not only build the software but also train the system. The evolution of quantum computing and neuromorphic computing challenges the very way we program our devices.

Indeed, we've just begun. Software is the invisible writing that whispers the stories of possibility to our hardware, yielding computing systems of exquisite complexity and promise.

And we, as insiders to computing, have the privilege of making the possible manifest.

References

1. W. Pleasant-Ryan, "Six Ways Developing Software Is Like Being a Parent," blog, 6 Sept. 2014; http://spin.atomicobject.com/2014/09/06/development-and-parenting.
2. T. DeRose, "The Connection between Movie Development and Software Development," *Proc. 2013 Int'l Conf. Software Eng.* (ICSE 13), 2013; http://dl.acm.org/citation.cfm?id=2486901.
3. "SoftwareAsFilmMaking," 2012; http://c2.com/cgi/wiki?SoftwareAsFilmMaking.
4. P. Alaedini, B. Ozbas, and F. Akdemir, "Agile Drug Development: Lessons from the Software Industry," *Contract Pharma*, 14 Oct. 2014; www.contractpharma.com/issues/2014-10-01/view_features/agile-drug-development-lessons-from-the-software-industry.
5. J.J. Merelo, "Five Reasons Why Writing Is So Much Like Software Develoment," blog, 30 May 2013; https://medium.com/i-m-h-o/five-reasons-why-writing-is-so-much-like-software-development-6d154a43719c.
6. E. Gordon et al., *Dating Design Patterns: Elements of Reusable Object Oriented Paired Programming*, Solveig-Haugland, 2003.
7. J. Wallace, "Is Software Art or Engineering?," *The Ethical Spectacle*, Nov. 1999; www.spectacle.org/1199/software.html.

GRADY BOOCH is an IBM Fellow and one of UML's original authors. He's currently developing *Computing: The Human Experience*, a major transmedia project for public broadcast. Contact him at grady@computingthehumanexperience.com.

See www.computer.org/software-multimedia for multimedia content related to this article.

EDITOR: Irena Bojanova, University of Maryland University College, irena.bojanova@umuc.edu

Internet of Cores

Saeid Abolfazli and Zohreh Sanaei, *YTL Communications and Xchanging Malaysia*
Irena Bojanova, *University of Maryland University College*

Leveraging global heterogeneous cloud-based resources,[1] mobile devices, and cyber-physical systems (CPSs) for smart and socially networked systems[2] is among several efforts to alleviate resource deficiencies in the global Internet of Things (IoT), or Internet of Anything. Currently, giant datacenters are provisioned to furnish computing and storage resources, while mobile devices are incorporated as sensing, interacting, and service delivery points. In this setting, the computing and storage capabilities of a few billion mobile devices, including smartphones, tablets, and car-mounted computers, are overlooked because such devices are mostly viewed as sensing and interacting end points or mobility enablers. Increasingly proliferating cloud and social networking service providers, such as Facebook and Google, will thus have to endlessly expand their infrastructures by purchasing acres of land, racks of servers, and megawatts of electricity to operate datacenters, which obviously isn't environmentally friendly.[3]

Furthermore, giant cloud datacenters feature coarse location granularity,[4] meaning that they're located in few geographical places far from the majority of users, whose service utilization originates long wireless network latency. Also, accessing such resources necessitates Internet connectivity, which can be a hurdle in disaster and emergency situations, such as the 2011 tsunami in Japan.

The current computing problem is due not only to insufficient computing and infrastructure resources, but also to imbalanced distributed resources. For a better understanding of the problem, we correlate the relationship between data or processes and resources to the water (resource) and Earth (demand) on a global scale. Although water covers more than 70 percent of the Earth, climate change and human impact have imbalanced its distribution. Under such an imbalanced water distribution, frequent flooding and drought are visible in different parts of the world. If all the giant, small, and tiny water sources worldwide were connected, excess water from one place due to massive rain could be pushed to another dry area to avoid drought. Similarly, if all computing and storage resources worldwide could be connected and accessed on demand, no resource would be over or underutilized, and people could do infinite processing using scattered but connected resources as they prefer.

The Internet protocol stack and massive infrastructure can technologically enable global connectivity of people and things, but resource discovery and addressing is a problem with current connectivity. Internet source and destination addresses are needed to communicate and to consume resources. This limitation removes autonomy from things and brings the technology to humans' attention (as opposed to Mark Weiser's vision of invisible computing that will be hidden in the background of humans' lives[5]).

Internet of Cores

Our envisioned Internet of Cores (IoC) will mitigate these problems. This "glocalized" (globally scaled, locally customized) elastic cloud-based infrastructure platform consists of a hybrid set of heterogeneous granular computing and storage resources accessible via the Internet and Ethernet through diverse wired or wireless communication and networking technologies. Glocalization allows globally accessible resources to be utilized

Figure 1. The Internet of Cores. In everywhere computing at a global scale, heterogeneous cores from on-premise and off-premise clouds, cloudlets, and mobile and vehicular computing resources would build a highly elastic and available infrastructure platform that's globally integrated with the Internet of Things.

in local settings with lower network latency. Hence, in the Internet's absence, the resources in a vicinity are accessible via varied low-latency Ethernet means. Ethernet connectivity can be any Internet-less link between consumers and providers—for instance, a private link between mobile users and mobile service providers to access either other mobile devices or providers' resources. Integration of such heterogeneous granular resources lays the base for a globally scaled IoT.

Furthermore, in the IoC, both resource consumers and providers are highly distributed, whereas with solutions such as grid computing[6] or SETI@Home,[7] the resource consumer applications are centrally built in the server and are sent to distributed resource providers for execution. Service consumption requests in the IoC can be sent from any consumer worldwide without being centrally stored in a node. Figure 1 illustrates the connected-everywhere computing environment at the global scale the IoC envisions.

A successfully adopted IoC platform would integrate every computing and storage device regardless of type, capability and capacity, location, ownership, accessibility, or features as long as there is a desire to participate in resource sharing. The IoC's ultimate goal is everywhere computing at a global scale (in line with that of the IoT and CPS), which can be further extended by focusing on computing and storage resource provisioning in emergency and low-access situations.

The IoC is a complement to IoT architectures and particularly to the US National Institute of Standards and Technology (NIST)'s vision of *cyber-physical cloud computing* (CPCC).[2] Glocalized computing and storage resources enable real-time and low-latency operations of the IoT and CPS. The IoC inherits elasticity, virtualization, and pay-as-you-use principles from cloud and sensing, and connectivity from the IoT.

IoC-enabled systems can operate in various architectural modes, including ad hoc, peer-to-peer (using P2P, Ethernet, or telecommunications networks), centralized, and hybrid.[8] Depending on the environment, communication facilities, and connectivity levels, each core can change roles from service consumer to service provider or service arbitrator (who manages the whole system). Wireless communications would take place using Bluetooth, wireless LAN, and cellular technologies, whereas wired connectivity would enable thing-to-thing communications.

Benefits of IoC

Successful integration of the IoC with current architectures, particularly NIST's CPCC, would yield several significant benefits, some of which we examine next.

Efficient Resource Allocation

Heterogeneous pools of computing and storage resources provided by the global IoC would enable efficient resource allocation to the given tasks based on inhomogeneous resource requirements. Scattered (global, continental, national, regional, and local) resources would be accessible and consumable by centralized, hierarchical, and P2P communication models. In the absence of IoC resources, assigning low-intensity tasks to rich computing resources leads to resource underutilization.

Improved Content Security and Privacy

Integrating a couple of billion computing devices worldwide into the IoT and CPS would create an opportunity for storing fine-grained, replicated data chunks globally instead of all in a single warehouse—that is, if we can maintain content vulnerability, availability, and security. Such highly distributed storage might benefit parties such as small and medium-sized enterprises not only in storing their data in cloud-based resources, but also in decreasing the chance of data exposure to competitors, given that small, meaningless data chunks would be spread around the globe.

Low Latency

The IoC would enable service consumers to utilize nearby computing resources instead of distant ones, whose utilization generates latency. Accessing distant cloud resources is associated with long latency, particularly for wireless clients, due to manifold intermediate hops between service consumers and providers. Such communication latency increases data access delay or remote computation time, leading the client devices to consume more energy.

Tourist Family in a Typhoon

Imagine Alex and his family spending a holiday at the seashore in Japan when a typhoon hits the beach, separating family members and pushing them to different remote islands. While Alex tries to locate them, each stressed-out, injured family member requires on-demand computational resources to translate signs and speech for communication with local rescuers. In the absence of a cellular network, accessing Google speech and text translation is impossible. In this scenario, Alex's separated family members can build an ad hoc sensing, communication, computing, and storage cloud with local residents to perform required tasks.[9]

High-Performance Scientific Computing

Research and publication in science and engineering are a rapidly emerging trend in which research universities and institutes invest considerable money in equipping their high-performance scientific computing platforms. Despite these investments, researchers are usually queued for resources due to the imbalance between requests and the supply of such resources. Incorporating hundreds to thousands of highly heterogeneous computing devices owned by students and staff into the high-performance computing platforms in universities could facilitate and expedite R&D efforts.

The premises of university campuses and institutes are usually Wi-Fi enabled and provide free electricity to residents for their computers and mobile devices.

In the absence of IoC resources, assigning low-intensity tasks to rich computing resources leads to resource underutilization.

Green Computing

Computing devices are rapidly outdated by technology advancements, leaving behind a huge number of antiquated devices with a few cores, gigabytes of RAM and storage, and a few hundred milliwatts of battery storage. Moreover, underutilized computers are barely running at full capacity because they're used mostly for Web browsing, accounting, or playing music, or are waiting idle to serve customers. The IoC would bring such resources back into the computing cycle toward global green computing.

Use Cases

Integrating cores in the IoC can be beneficial in a wide range of domains.

Therefore, it's highly feasible and practical to utilize such resources as high-performance computing platforms, not only to expedite high-performance scientific computing, but also to increase resource utilization of residents' devices. Unlike previous works, including SETI@Home[7] and grid computing,[6] the IoC isn't only focusing on high-performance computing at a global scale, but also on a local scale. The IoC strives to provision on-demand, unrestricted, and unconditioned resources via any possible communication technology under both centralized and decentralized architectures, something that isn't currently available. Current systems are considering desktops and laptops, whereas the IoC considers every single device capable of performing computation, including car-mounted computers, smart watches, and smart coffee mugs.

Mobile Analytics

Highly profitable mobile telecommunications providers perpetually struggle to gain insight both into their networks and about subscribers to successfully adapt their services to better serve and ultimately retain subscribers. Users are constantly performing varied voice and data transactions via their mobile communication networks in a mostly pay-as-you-use fashion. Such a business model necessitates that vendors keep track of entire user and network activities to respond to complaints and inquiries. However, storing and computing such big data puts vendors under serious constraints due to the capital and operational costs of computing. The IoC can alleviate this problem by providing on-demand resources from administrators, employees, and even users to store and process big data in real time, a feature that is unavailable in cloud and grid computing models.

Challenges in Realizing the IoC Vision

Several social and technical implications, particularly those we next describe, need to be addressed before the IoC can be successfully implemented in practice.

Incentivizing resource sharing by individual owners is necessary because the IoC is based on the pay-as-you-use principle.

Global Hybrid Resource Scheduling

Global hybrid resource scheduling (GHRS) is essential in scheduling indexed and discovered resources on a global scale. The IoC requires hybrid scheduling, including both centralized and decentralized resource scheduling, to fulfill users' computing and storage requirements anywhere, anytime. If connected to the Internet, resource consumers could choose a centralized scheduler, whereas consumers can perform P2P discovery if centralized scheduling is either impossible or costly.

Interoperability

Interoperability among highly heterogeneous and diverse computing nodes in the IoC is one of the most significant technological and standards challenges to realizing the IoC vision. Existing hardware, platform, operating system, API, and feature heterogeneities among the multitude of cores in the IoC necessitate standardization.

Big Data Processing

Although the IoC will provide tremendous storage and computing resources to store and process large volumes of data, software solutions and algorithms that can perform high-performance distributed big data computing on this scale are a nontrivial challenge. The IoC demands highly distributed big data processing techniques with a low communication and computing footprint.

Persistent Elastic Communications

Persistent communications in the IoC refers to the ability to communicate with a desired node using any possible communication technology from a pool of heterogeneous communication technologies and mediums with minimum user interference. The term "elastic" is inspired by cloud computing's resource elasticity. So, in elastic communication, network capacity is expected to expand as traffic grows in an on-demand fashion. Realizing persistent elastic communications requires an integrated middleware and also traffic balancing solutions similar to big data stream mobile computing[10] on a global scale to automatically select the most appropriate networking technology that provides communication among nodes despite existing heterogeneities and inconsistencies.

Global Core Naming

In the future, each core will require a unique, public, global static identification number (GSID) that's accessible independent from any specific communication technology. Although landline numbers, IPv6 numbers, and international mobile station equipment identity

(IMEI) numbers are globally unique (except possibly fake IMEIs), their uniqueness is limited to certain technologies or communication networks.

Social Trust and Incentive

The IoC will comprise a huge number of individuals who own typically more than one core. Without trust among members of this society, this concept can't become reality due to the high impact of resource sharing among individuals.[11] Moreover, incentivizing resource sharing by individual owners is necessary because the IoC is based on the pay-as-you-use principle of cloud computing. Besides the financial cost of electricity to perform computing and storage, owners risk sharing their resources with strangers. Thus, it's impractical to expect people to share their resources and accept this risk without any incentive. Successful IoC deployment necessitates extrapolating incentive mechanisms in e-learning[12] to the crowd and personal computing areas.

International Governance and Regulation

The IoC requires setting international rules and regulations, particularly data sovereignty. In the global IoC, user data and some sensitive data might be stored in data warehouses outside countries in which it is banned in the current setting. Some countries don't allow certain data to move beyond their boundaries, which is known as the data sovereignty issue.

Things are increasingly connecting to the Internet for enhanced quality of life. However, successful deployment of IoT systems globally demands highly scalable, on-demand computing resources distributed across the globe with the least latency, which is nontrivial to realize. The envisioned IoC promises to cope with the scale and fulfill increasing computing demands. To realize this vision, we must answer questions such as how do we keep track of billions of resources, how do we efficiently allocate the most appropriate local resource to the given task, what are the incentives for resource sharing, and how do we secure such a system and establish trust among users? We are eager to see the IoC successfully implemented in the near future. IT

References

1. S. Abolfazli et al., "Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges," *IEEE Comm. Surveys and Tutorials*, vol. 16, no. 1, 2014, pp. 337–368.
2. E.D. Simmon et al., *A Vision of Cyber-Physical Cloud Computing for Smart Networked Systems*, US Nat'l Inst. of Standards and Technology interagency/internal report 7951, Aug. 2013.
3. J. Letzing, "Facebook Data Center Is Boon for Oregon Town," *Wall Street J.*, 21 Jan. 2011.
4. Z. Sanaei et al., "HMCC: A Hybrid Mobile Cloud Computing Framework Exploiting Heterogeneous Resources," *Proc. IEEE Mobile Cloud Conf.*, to appear, 2015.
5. M. Weiser, "The Computer for the 21st Century," *IEEE Pervasive Computing*, vol. 1, no. 1, 2002, pp. 19–25.
6. F. Berman, G. Fox, and A. Hey, *Grid Computing: Making the Global Infrastructure a Reality*, Wiley, 2003.
7. D.P. Anderson et al., "SETI@home: An Experiment in Public-Resource Computing," *Comm. ACM*, vol. 45, no. 11, 2002, pp. 56–61.
8. Z. Sanaei et al., "Hybrid Pervasive Mobile Cloud Computing: Toward Enhancing Invisibility," *Information—An Int'l Interdisciplinary J.*, vol. 16, no. 11, 2013.
9. S. Abolfazli et al., "RMCC : A RESTful Mobile Cloud Computing Framework for Exploiting Adjacent Service-Based Mobile Cloudlets," *Proc. IEEE Int'l Conf. Cloud Computing Technology and Science* (CloudCom), 2014, pp. 793–798.
10. J.S. Enzo Baccarelli et al., "Energy-Efficient Dynamic Traffic Offloading and Reconfiguration of Networked Datacenters for Big Data Stream Mobile Computing: Review, Challenges, and a Case Study," *IEEE Network*, to appear, 2015.
11. J. Seigneur, "Social Trust of Virtual Identities," *Computing with Social Trust*, Springer, 2009, pp. 73–118.
12. H.G.K. Hummel et al., "Encouraging Contributions in Learning Networks Using Incentive Mechanisms," *J. Computer Assisted Learning*, vol. 21, 2005, pp. 355–365.

Saeid Abolfazli *is a research lead and data scientist at YTL Communications and Xchanging Malaysia, where he leads various scientific and analytic R&D activities for WiMAX and LTE networks. His main research interests are mobile cloud computing, cloud-based mobile augmentation, and big data analytics for mobile users. Contact him at abolfazli@ieee.org.*

Zohreh Sanaei *is a research lead and data scientist at YTL Communications and Xchanging Malaysia. Her main research interests are mobile cloud computing, distributed computing, and big data analytics for mobile users. Contact her at sanaei@ieee.org.*

Irena Bojanova *is a professor and program director of information and technology systems at University of Maryland University College. You can read her Internet of Things blog and her cloud computing blog at www.computer.org. Contact her at irena.bojanova@computer.org.*

This article originally appeared in IT Professional, *vol. 17, no. 3, 2015.*

Designing Resource-Aware Cloud Applications

Reiner Hähnle, Technical University of Darmstadt

Einar Broch Johnsen, University of Oslo

Realizing the full potential of virtualized computation—the cloud—requires rethinking software development. Deployment decisions, and their validation, can and should be moved up the development chain into the design phase.

As data storage and processing move to the cloud, our interactions with computers are undergoing dramatic transformation. The cloud, as a network of virtual machines, has no fixed location and is only accessed remotely. Although many continue to use a desktop PC to access cloud-based data and applications, people increasingly rely on their smartphones, tablets, and other mobile devices.

Despite serious data-privacy concerns, several qualities make cloud computing compelling from a business standpoint.[1] One such quality is elasticity: businesses can pay for computing resources only when they're needed, avoiding major upfront investments for resource provisioning; the client application can add processing power, memory, and additional virtual machines on the fly as needed. The cloud also offers a scalable virtualized platform for data processing that can be shared among multiple devices—if a service uses cloud-based processing, its capacity can be adjusted automatically as new users arrive. Thus, another key cloud feature is agility: new services can be quickly and flexibly brought to market at minimal cost, without initial investments in hardware.

With these key advantages, cloud computing is rapidly gaining popularity throughout the world. Currently, there are more than 3.9 million cloud computing–related jobs in the US and more than 18 million worldwide.[2] In the EU, cloud computing is projected to create 2.5 million jobs and grow the economy annually by €160 billion by 2020.[3] However, inconsistent reliability and insufficient control

 Published by the IEEE Computer Society 2376-113X/15/$31.00 © 2015 IEEE

EDITOR **MIKE HINCHEY**
Lero—the Irish Software Research Centre;
mike.hinchey@lero.ie

of resources present significant barriers to the industrial adoption of cloud computing. To overcome these barriers and gain control of the cloud's virtualized resources, client services must become resource aware.

RETHINKING SOFTWARE DEVELOPMENT

The virtualization of computation has transformed how we interact with, share, and manage data, but taking full advantage of cloud computing requires us to rethink how we design and develop software.

The elasticity of software executed in the cloud gives designers control over the execution environment's resource parameters, such as the number and kind of processors, memory, storage capacity, and bandwidth. Parameters can even be changed dynamically at runtime. Thus, cloud-service clients can not only deploy and run software, but they can also fully control the tradeoffs between incurred costs to run the software and the quality of service (QoS) delivered.

Traditionally, software design is based on specific assumptions about deployment, including the size of data structures and the amount of RAM and number of processors required to execute it. However, because of the cloud's transformative effect on computation, software now must be designed for scalability from the beginning or else it will later require extensive design changes.

Later changes to software can be fatal because of the relative increase in cost to fix defects during successive development phases.[4] As Figure 1 shows, the IBM Systems Sciences Institute estimated that, compared to fixing a defect during design, it costs 6.5 times more to fix during implementation, 15 times more during testing, and 100 times more during maintenance. These costs don't take into account additional economic losses associated with delayed time to market, lost customers, and bad public relations.

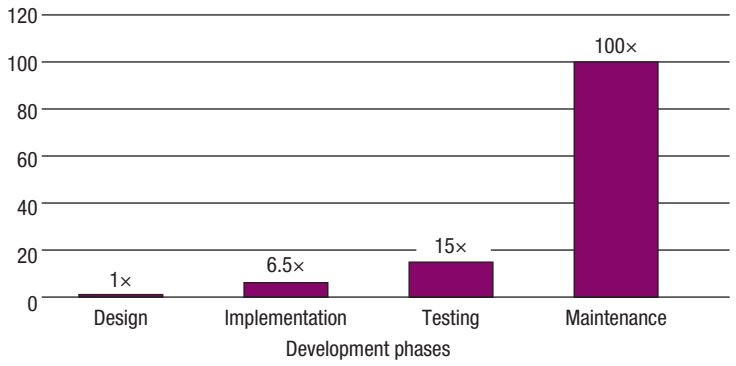


Figure 1. Relative costs to fix software defects at each phase of the development process in a static infrastructure. (Source: IBM Systems Sciences Institute)

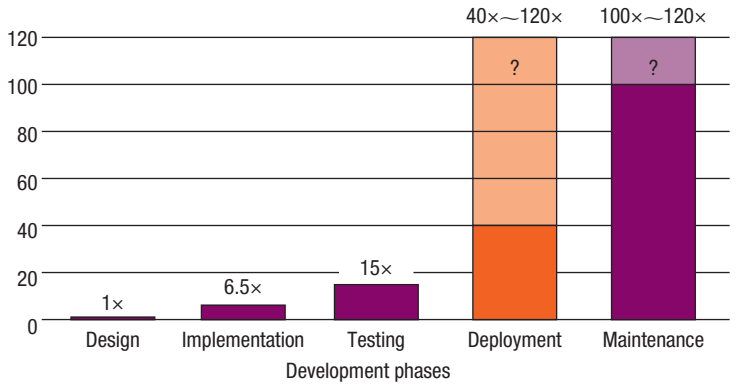


Figure 2. Relative costs to fix software defects at each phase of the development process in an elastic virtualized infrastructure. (Source: E. Albert et al., "Engineering Virtualized Services," *Proc. 2nd Nordic Symp. Cloud Computing Internet Technologies* [NordiCloud 13], 2013, pp. 59–63)

The ratios in Figure 1 are for static deployment, but it's reasonable to expect even more significant costs when fixing defects in elastic virtualized environments due to the additional complexity of resource management. Figure 2 presents a conservative estimate of the ratios in an elastic environment: it can cost between 40 and 100 times more to fix defects during deployment than during design. It's thus essential to detect and fix deployment errors—such as failure to meet service-level agreements (SLAs)—as early in the development process as possible, preferably during the design phase.

To maximize cloud computing's potential, software development demands a new methodology that models deployment during the design phase and facilitates detection of deployment errors early and efficiently through the use of software tools such as simulators, test generators, and static analyzers.

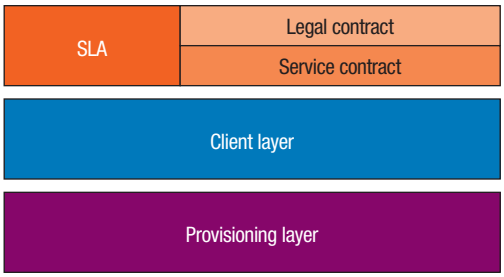


Figure 3. Conceptual layers of a deployed cloud service. Functionality is represented in the client layer. The provisioning layer makes resources available to the client layer and determines available memory, processing power, and bandwidth. The service level agreement (SLA) specifies what resources the provisioning layer should make available to the client service and includes both a legal and a service contract.

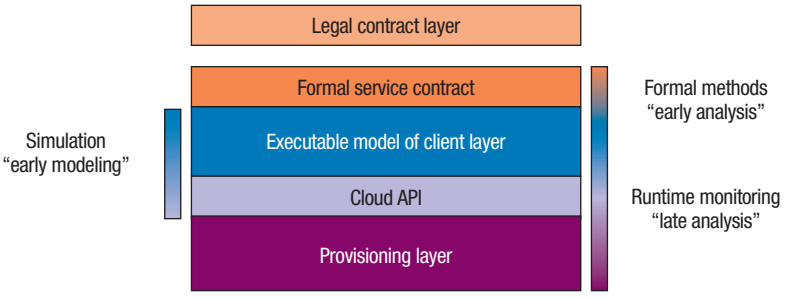


Figure 4. Conceptual layers of a resource-aware deployed cloud service. The service contract is formalized and statically checked against an executable client model at design time. The client model can access an abstract interface to the provisioning layer in the form of a cloud API, thus enabling simulation. The cloud API also permits instrumentation of deployed applications with monitors that ensure compliance with the service contract at runtime.

CONTROLLING DEPLOYMENT IN THE DESIGN PHASE

Our analysis presents a software engineering challenge: how to introduce deployment decision validation into the modeling phase of the software development chain without convoluting the design with deployment details.

With a cloud-based service, developers typically design its functionality first, then determine which resources are needed and how they'll ultimately be provisioned through an SLA, as Figure 3 shows. Functionality is represented in the client layer. The provisioning layer makes resources available to the client layer and determines available memory, processing power, and bandwidth. The SLA specifies what resources the provisioning layer should make available to the client service. A typical SLA has two components: a legal contract stating the mutual obligations of the client service and software developer, and consequences in the event of a breach by either party; and a service contract that details the technical parameters and cost figures for the offered services.

We propose connecting these currently distinct conceptual layers to better control deployment in the design phase.

First, the client should have access to the provisioning layer through a special cloud API to observe and modify resource parameters, as Figure 4 shows. The cloud API isn't identical to the APIs commonly offered by cloud providers, but is instead an abstract interface to the provisioning layer. Using an executable language, such as Abstract Behavioral Specification (ABS),[5] it enables design-phase modeling of client behavior to simulate different client-side provisioning schemes and observe their impact on cost and performance. Creating such an executable model is feasible during the design phase.

Second, to connect an SLA to the client layer, a formal semantics is given to service contract aspects of the SLA. This would allow, at design time, analysis of client behavior such as resource consumption and worst-case performance vis-à-vis the SLA; test-case generation; and functional verification.[6] This process is highly automated, for example, in ABS.[7] Design-time analysis makes the assumptions about the cloud API explicit. The cloud API also permits instrumentation of deployed applications with monitors that ensure compliance with the service contract at runtime.

BENEFITS OF RESOURCE AWARENESS

Making deployment decisions during the design phase can help shift control from the provisioning layer to the client layer and enables the client service to become resource aware. This provides numerous attractive opportunities.

Fine-grained provisioning

As in other metered-industry sectors such as telephony and electricity, business models for cloud-based resource provisioning are increasingly fine grained. Selecting the best software model is highly complex, which thus far has benefited resource providers. Design-time analysis and comparison of deployment decisions make it possible to optimize provisioning schemes for end users, such as spot pricing.

Tighter provisioning

Better profiles of the client layer's resource needs will help cloud providers avoid over-provisioning to comply with their SLAs. Better usage of resources means that more clients can be served with the same amount of hardware in the datacenter, without violating SLAs and incurring penalties.

Application-specific resource control

Through design-time analysis of scalability, the client layer can better leverage the cloud's elasticity. Knowing beforehand the load thresholds for scaling up deployment, cloud-services providers can avoid breaking SLAs and disappointing end users' expectations.

Application-controlled elasticity

Autonomous, resource-aware services that run their own resource-management strategy are possible. Such services could monitor loads on virtual machines as well as end-user traffic, and thus could make decisions about the tradeoffs between the QoS delivered and the incurred cost. The service could interact with the provisioning layer through an API to dynamically scale up or down, or it could even request or bid for virtual machine instances with given profiles in a future virtual resource marketplace.

The efficiency and performance of cloud-based services could be improved by moving deployment decisions up the development chain. In addition, resource-aware services could give the client better control of resource usage and thereby comply with SLAs more inexpensively.

Formal methods for early analysis, executable models of client behavior for early monitoring, and runtime monitors for late analysis are key elements to realizing this vision. Concrete examples of these elements are being implemented as part of the EU FP7 project "Envisage: Engineering Virtualized Services" and are described, along with other details about the project, at www.envisage-project.eu.

ACKNOWLEDGMENT

This work was supported in part by EU project FP7-610582 Envisage: Engineering Virtualized Services.

REFERENCES

1. R. Buyya et al., "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, 2009, pp. 599–616.
2. L. Columbus, "Where the Cloud Computing Jobs Will Be in 2015," *Forbes*, 12 Dec. 2014; www.forbes.com/sites/louiscolumbus/2014/12/12/where-cloud-computing-jobs-will-be-in-2015.
3. European Commission, "European Cloud Computing Strategy," Digital Agenda for Europe: A Europe 2020 Initiative, updated 27 Feb. 2015; http://ec.europa.eu/digital-agenda/en/european-cloud-computing-strategy.
4. B.W. Boehm and P.N. Papaccio, "Understanding and Controlling Software Costs," *IEEE Trans. Software Eng.*, vol. 14, no. 10, 1988, pp. 1462–1477.
5. E.B. Johnsen et al., "ABS: A Core Language for Abstract Behavioral Specification," *Proc. 9th Int'l Symp. Formal Methods for Components and Objects*, LNCS 6957, Springer, 2011, pp. 142–164.
6. E. Albert et al., "Engineering Virtualized Services," *Proc. 2nd Nordic Symp. Cloud Computing Internet Technologies* (NordiCloud 13), 2013, pp. 59–63.
7. E. Albert et al., "Formal Modeling of Resource Management for Cloud Architectures: An Industrial Case Study Using Real-Time ABS," *J. Service-Oriented Computing and Applications*, vol. 8, no. 4, 2014, pp. 323–339.

This article originally appeared in Computer, *vol. 48, no. 6, 2015.*

REINER HÄHNLE is a professor and dean of the Computer Science Department at the Technical University of Darmstadt. Contact him at haehnle@cs.tu-darmstadt.de.

EINAR BROCH JOHNSEN is a professor in the Department of Informatics at the University of Oslo. Contact him at einarj@ifi.uio.no.

Innovations in Ubicomp Products

Editor: Albrecht Schmidt ■ University of Stuttgart ■ albrecht@computer.org

Connected Tools in Digital Design

Christian Weichel, Jason Alexander, Abhijit Karnik, and Hans Gellersen, *Lancaster University*

EDITOR'S INTRO

The Internet of Things has become a mainstream topic. Many ideas and concepts discussed in pervasive and ubiquitous computing over the last two decades are now commonly accepted. Here, the authors give us a glimpse of the potential when we make tools part of the Internet of Things. Their prototypes aren't yet products. However, in looking at the potential for reducing errors and saving time, I expect many connected tools to hit the market in the next few years—not only in engineering but also in other domains such as medicine, cooking, and agriculture. The examples provided show impressively how tools, once connected, offer new opportunities.

The idea of bridging the digital and physical is intriguing, and many technologies get us closer to this vision. Some of the ideas, although they've been around for a while, are still inspiring. It might be a good time to read (or re-read) the book *Mirror Worlds: Or the Day Software Puts the Universe in a Shoebox—How It Will Happen and What It Will Mean*, by David Gelernter (Oxford, 1993). Also see this issue's related Spotlight department, "The Mirror World: Preparing for Mixed-Reality Living."

— Albrecht Schmidt

As digital fabrication and digital design become more and more pervasive, the physical tools we use in conjunction will have to catch up. With the Internet of Things, cyberphysical systems, and Industry 4.0 in our midst, connecting and integrating measurement tools into design processes is a logical step. Here, we describe the first steps in that direction, coming from a variety of communities: academics, makers, and industry alike. In particular, we present our spatio-tangible (SPATA) tools for fabrication-aware design.

DIGITAL DESIGN AND FABRICATION

Digital fabrication, such as 3D printing and laser-cutting, lets users quickly create physical artifacts from digital files. The interfaces and environments used for designing these artifacts are typically tied to a computer screen and are thus removed from the physical world. This separates the physical nature of the fabricated artifacts from the virtual environments in which they were designed. However, during the design of a fabricable artifact, physical features (such as size and angle) play an important role, because the artifact will be subject to that physicality once fabricated. Furthermore, fabricated artifacts often interact with previously existing objects—for example, holding, encasing or decorating them.

In digital fabrication, as well as in design and engineering, physical features play an important role. Many tools, such as calipers, rulers, and protractors, have been developed to measure those features so that they can inform a particular design. These tools, until now, have been analog tools, because users must manually read a value from them (for example, from a veneer scale or a display) to incorporate the value into a design. In particular, when using CAD software, or designing in any other virtual environment, users often employ these tools to get a sense of the size of what they're designing. How big is the box shown on the screen in the real world?

BRIDGING THE INPUT GAP

With digital design, these analog, disconnected tools create an inconvenient design experience. Every interaction with physical features requires us to shift our attention from the design environment to the physical world and back. The measured value must be manually transported from the measurement tool or vice versa. These context switches are time consuming and disruptive to the design workflow.

This disconnection has been recognized by measurement tool vendors and makers. Both communities are exploring and producing computer-connected tools—for example, calipers that can unidirectionally transfer measured value to a computer. An example is Mitutoyo; they sell data cables for their measurement tools that act as virtual keyboards. Measurements are directly entered into the computer, and these cables are advertised as a means

 Published by the IEEE Computer Society

to make repetitive data-entry tasks more efficient.[1] Because the input is the same as from a keyboard, it works with any software. The maker community uses an Arduino to connect to a cheap traditional caliper, read its value, and integrate that into a 3D design environment.[2] This is just a starting point for connected tools and, in our research, we explore how to extend this concept.

SPATIAL-TANGIBLE TOOLS

SPATA is a system that introduces tangible tools for fabrication-aware design (SPATA tools),[3] a digital adaptation of two commonly used measurement tools: calipers for measuring length, and bevel protractors for measuring angle. The SPATA tools can measure their respective value (length or angle), but are also actuated so that they can actively present the value in the physical world: the calipers have a self-actuated lower jaw that can physically represent length; the protractor can move its blade to output an angle (see the red parts shown in Figure 1).

Both tools can bidirectionally transfer—input and output—their value between the physical and virtual world. Users can measure a physical object, and the measured value is automatically transferred to the design environment. Conversely, length, distance, and angle measured in the virtual environment are automatically transferred to the physical world and presented by the SPATA tools. For example, to get an impression of the size of an object a user is designing, you can measure the object in the design environment and have the SPATA calipers tangibly output the size in physical space.

SPATA integrates closely into virtual environments used to design fabricable artifacts—namely, mechanical computer aided design (mCAD), mesh-based modeling, and 2D design. When designing new objects in these environments, users create shapes (such as primitives like boxes, cylinders, or rectangles), manipulate them, and combine them into new forms. Those tasks, in a fabrication-aware context, often require physical measurements taken from existing objects.

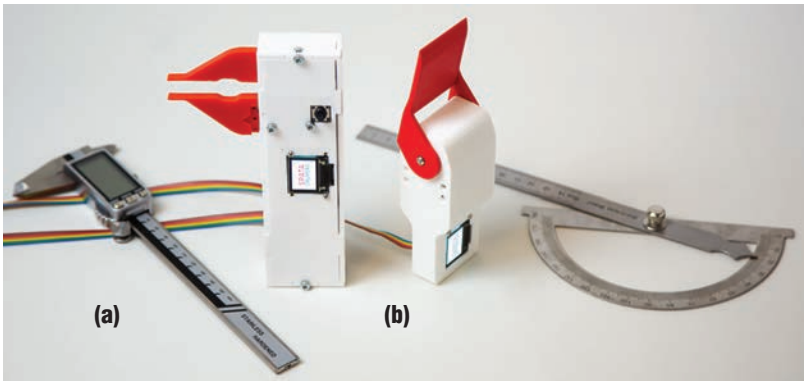


Figure 1. The two spatial-tangible (SPATA) tools next to their original counterparts. (a) calipers for size in-/output and (b) the protractor for angle in-/output.

To reduce the need to switch between the virtual and physical world, the integrated tools support those tasks in the respective design environments. By partially offloading control to the measurement tools, task execution becomes more fluid and convenient. For example, to model a box-shaped object (such as an enclosure) using SPATA, users can measure all three dimensions (width, height, and depth) in sequence without having to put down the SPATA calipers or manually type in measurements.

To further support design tasks, SPATA tools can sense their orientation, display additional information (such as estimated fabrication time or the amount of required material), and have a button built in. We combine these capabilities to provide a more integrated and convenient design experience when designing for the physical world.

INTEGRATION WITH DESIGN ENVIRONMENTS

Modern mCAD systems, such as Autodesk Inventor or SolidWorks, are based on 2D sketches that are extruded or revolved into solid 3D objects. When drawing sketches or creating these objects, users need to constrain different dimensions, often using physical values, such as length and angle.

SPATA supports the creation of boxes from a prescribed series of real-world measurements (width, height, depth, and so on). The measurements can be performed in rapid succession using the button on the SPATA tool, so users can create a new cube with no context switch (see Figure 2). A similar sequence exists for cylinders: first measuring the diameter, then height. After a primitive has been created, SPATA stays in this mode, enabling a series of primitives to be built on top of each other. Elsewhere, we describe a use case in which this workflow integration reduced the number of context switches by 74 percent.[3]

Another common design environment type is mesh-based 3D modeling, which is a general-purpose modeling paradigm. It's often used to create organic and artistic models in tools, such as Autodesk Mudbox, Blender, or ZBrush. The smallest unit of manipulation is a vertex or an edge of the 3D model. Among other things, SPATA can be used to scale models in these environments, bringing, for example, the model to a certain size based on a single dimension. The application scenario in the related sidebar demonstrates more of the integration available.

GENERALIZING BEYOND FABRICATION

The SPATA concept—automated measurement transfer and integration into

INNOVATIONS IN UBICOMP PRODUCTS

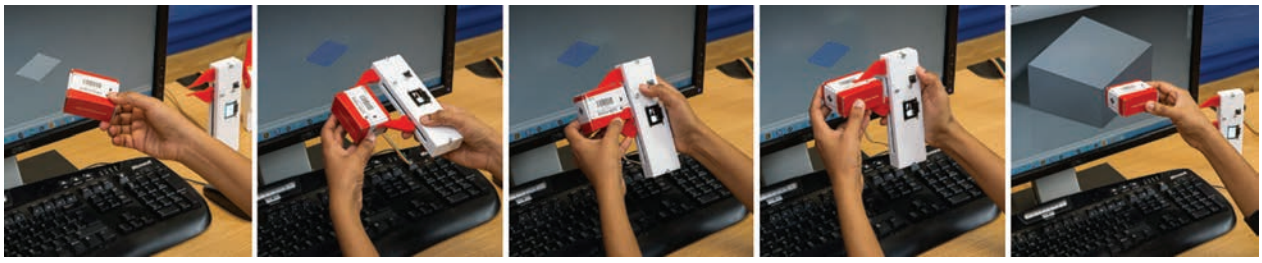


Figure 2. Creating a box (from left to right): select the ground plane, measure width, height, and depth.

APPLICATION SCENARIO

In this scenario, we want to create a flower vase that will be 3D printed. To model the vase, we use a mesh-based design environment that supports vertex-based modeling, sculpting, and constructive solid geometry. For artistic modeling, open input is often used instead of a mouse; we follow this practice.

We start the design process by creating a new cylinder. The vase needs to be correctly sized so that flowers fit in it and it can be placed on a desk. Using the SPATA calipers and their ability to globally scale, we scale the cylinder until it is 8 centimeters high. Using local scaling, we scale the diameter of the vase to 4 centimeters (see Figure A1).

Next, we add the decorative features by drawing on the cylinder using the pen. We use the SPATA calipers, which we now hold in our nondominant hand, to rotate the model so that we can draw on all sides (see Figure A2). This way, we don't have to change the mode from drawing to rotating; use the pen to draw and the SPATA tool to rotate.

Sculpting the shape has changed its size as well. Using Blender's built-in measurement tool, we measure the vase model. This causes the SPATA calipers to output that size in the physical world (see Figure A3). This way we can compare the size against the flower, or get a feel for the dimensions of the vase we're creating.

To make the vase more interesting, we want it to stand slightly angled. To explore different angles, we use the SPATA protractor. During this exploration, our focus is on the SPATA tool, which gives us additional, fabrication-specific feedback. When we use a too steep angle, we'll be warned when the current angle will make the fabrication take longer and be more expensive (see Figure A4).

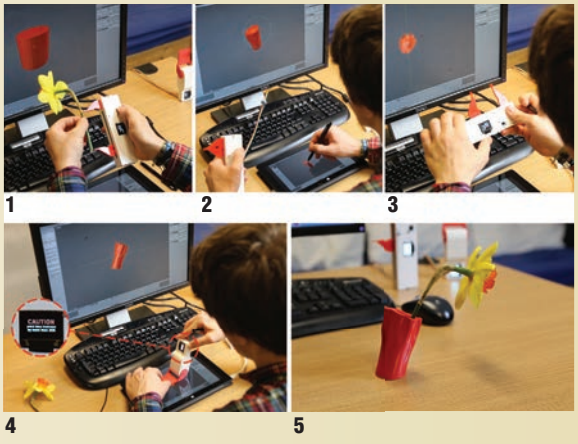


Figure A. Creating a vase. (1) We start with a cylinder, scaled to 8 centimeters using the SPATA calipers. (2) We sculpt decorative features, using the SPATA tools for orientation. (3) We check the size of the model and (4) explore different flower-hole angles.

Lastly, we cut off the bottom to create a flat surface for the vase to stand on and add the flower hole. We then send it to a 3D printer. The resulting vase fits the flower as designed and doesn't need support structures to print (see Figure A5).

design environments—generalizes to tools other than calipers and protractors. On different scales, different tools are used. HandSCAPE,[4] the digital measurement tape, for example, could also output its value using an additional motor. Alternatively, a folding ruler could be augmented to support input and output of not only length but also the angle along its joints.

Physical features besides length and angle could also be considered. For example, an integrated measurement tool for material stiffness could be used to design multimaterial 3D printed objects. Adobe sells tangible, integrated tools (a pen and a ruler) for Desktop Publishing and sketching (www.adobe.com/uk/products/ink-and-slide.html). The pen can pick up colors from the environment and display the current ink color on its back.

The need for integrating spatial features extends beyond design for fabrication. In computer supported collaborative work, or whenever there is a spatial/temporal division between users, SPATA could be used to transfer spatial features. For example, two spatially disconnected users could exchange the screen-size of the new

tablet they've bought. In a temporally disconnected scenario, users could get an impression of the size of an object offered in an online store, or measure parts of their body to order a custom-made product.

Connected and integrated design tools have the potential to make digital design not only more convenient but also easier to use. Through close integration into design environments, we can lower the barriers for new users. Future work will need to explore this potential and go beyond traditional tools. Connecting new tangible interaction technologies (such as shape-changing displays) with digital design environments will likely have a profound impact on how we design things.

REFERENCES

1. Mitutoyo America Corp., "New Mitutoyo USB Input Tool," 3 Aug. 2011; www.mitutoyo.com/press_releases/new-mitutoyo-usb-input-tool.
2. M. Thalheimer, 2013, "Arduino Reads Digital Caliper," blog, 2013; https://sites.google.com/site/marthalprojects/home/arduino/arduino-reads-digital-caliper.
3. C. Weichel et al., "SPATA: Spatio-Tangible Tools for Fabrication-Aware Design," *Proc. Ninth Int'l Conf. Tangible, Embedded, and Embodied Interaction* (TEI), 2015, pp. 189–196; http://doi.acm.org/10.1145/2677199.2680576.
4. J. Lee et al., "HandSCAPE: A Vectorizing Tape Measure for On-Site Measuring Applications," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI), 2000, pp. 137–144; http://doi.acm.org/10.1145/332040.332417.

Christian Weichel is a PhD student in the School of Computing and Communications at Lancaster University. Contact him at c.weichel@lancaster.ac.uk.

Jason Alexander is a lecturer in Human-Computer Interaction at Lancaster University, UK. He has a particular interest in designing systems that bridge the virtual-physical divide. Contact him at j.alexander@lancaster.ac.uk.

Abhijit Karnik is a lecturer of human-computer interaction at the Lancaster University, UK. Contact him at a.karnik@lancs.ac.uk.

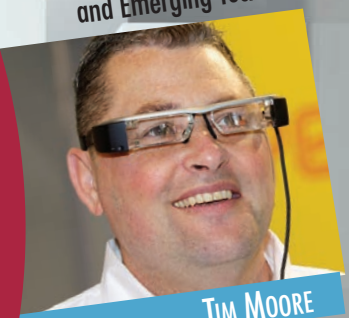
Hans Gellersen is a professor of interactive systems at Lancaster University's School of Computing and Communications. Contact him at h.gellersen@lancaster.ac.uk.

This article originally appeared in IEEE Pervasive Computing, *vol. 14, no. 2, 2015.*

IEEE computer society

ROCK STARS OF WEARABLES

Tom Emrich
Consultant
Mobile, Wearables, and Emerging Tech



Tim Moore
Director of Wearable Technologies, Rochester Optical

Pankaj Kedia
Senior Director & Business Lead, Smart Wearables
Qualcomm

What Technologies Are Needed for Wearables? What Markets are Emerging? Stake Your Claim in the Fast-Growing Future of Wearables.

At this must-attend event of 2015, top industry experts give you actionable solutions for wearable success.

You're invited.

23 September 2015

Brazos Hall
Austin, TX

REGISTER NOW

Early Discount Pricing Now Available!

computer.org/wearables

IEEE

Editor: Gary McGraw, gem@cigital.com

Silver Bullet Talks with Brian Krebs

Gary McGraw | Cigital

Hear the full podcast at www.computer.org/silverbullet. Show links, notes, and an online discussion can be found at www.cigital.com/silverbullet.

Brian Krebs is a reporter and blogger who runs the world-famous website KrebsOnSecurity.com. Krebs was a reporter for *The Washington Post* before forming Krebs on Security; he has written thousands of pieces for the Security Fix blog and many front-page stories. His book *Spam Nation* was released in November 2014.

You were an Apple II guy way back in the early '80s, and so was I. In fact, my machine led indirectly to my work in software security. Do you think that growing up with a machine in the house impacted your career?

As a kid, I spent a ridiculous amount of time tying up our phone lines at home working on an Apple II and then an Apple II Plus. I was on the bulletin boards, these text-based communities, downloading all kinds of stuff.

I think the idea of growing up around a machine and doing a little bit of coding and really understanding how something like that works helps to ground any sort of reporting in technology. A lot of people don't have that, so maybe that's an advantage for you.

People think of what they see in the movies about reporters running around, chasing down politicians. For the most part, reporting is boring as all get out, and it's data-intensive work. Half the time you put all your blood, sweat, and tears into something, and it turns out to be a pile of nothing. You've got nothing to show for it, and it's frustrating. The same thing is probably true with trying to produce a program or something that you really want to behave a certain way or look a certain way. Maybe it works out, maybe it doesn't.

Reporting does have this thing—well, should have this thing—called fact-checking, which it used to be better at. I know you're extremely careful with that.

Fact-checking is increasingly a luxury in journalism: having somebody looking over your shoulder, somebody to question your assumptions about things. As a journalist, you get yourself into trouble really quickly if you're not taking your assumptions at the door as often as possible.

Systems often break in surprising ways. So if you can get to the assumptions that the architect, the designer, or the engineer was making and make them disappear, the systems just crumble. That's a very high-level technique but a very useful one.

Who knew that there were such parallels?

I've been working with Richard Danzig, who used to be the Secretary of the Navy, on getting Washington, DC, to understand cybersecurity. He has a deeply disturbing view of MUD [mutually unassured destruction] versus MAD [mutually assured destruction]. The notion is that if we have cyberkinetic capability and can make a second strike possibility go away, in the nuclear realm it's much more likely that people will do a first strike, which is really, truly, deeply disturbing. I had never really thought of it until he expressed it

2376-113X/15/$31.00 © 2015 IEEE Published by the IEEE Computer Society August 2015

About Brian Krebs

Brian Krebs was a reporter for *The Washington Post* until 2009, when he started his blog, KrebsOnSecurity.com, which focuses on cybercriminals, mostly centered in Eastern Europe, who profit off malware, spam, frauds, and heists. Krebs became interested in computer security after his home network was overrun by a Chinese hacking group in 2001. He began learning all he could about writing code and hacking, and even taught himself Russian. Krebs was the first to report on the Stuxnet virus and the massive Target data breach, and he broke the news of hacks and fraud at Adobe Systems, Experian, and Neiman Marcus. His book *Spam Nation: The Inside Story of Organized Cybercrime—from Global Epidemic to Your Front Door* was released in November 2014 and is now a *New York Times* best seller.

clearly in a paper that he wrote for CNET. What do you think of the risks and the hype surrounding cyberwar versus the reality of the cybercrime that you're steeped in every day?

I think a lot of what gives Washington pause about taking the whole cyber issue seriously is that everybody's been holding their breath for a long time for some big attack that's going to rock the various services.

The cyber Pearl Harbor.

Right. We've been in a cyberwar with our adversaries for many years now, and it's only starting to dawn on a small cadre of folks in DC about how badly it's gone for us over the last few years. We are the world's most technology-dependent nation, and our vulnerabilities reflect that. So just because we haven't seen that cyber Armageddon attack, it doesn't mean that we're not already moving.

I want to compare it to cybercrime. If you look at the root cause of all this stuff, it's very similar. So if we build smarter, it will help us with cyberwar defense, or making sure that cyberwar doesn't happen, and help us address the cybercrime problem.

You're right in that both of the problems stem from an underappreciation of how much we rely on these technologies. Last year was a busy year for me. I had more than 25 speaking engagements and a book tour. Many of the companies I've spoken at have been Fortune 500 companies. It's been really interesting to hear the questions they're asking and hear the answers they're giving me in response to my questions, which are essentially trying to get a sense of how clued in the leadership within that organization is about their reliance on technology, computer systems, and information security. The point I try to get across to these organizations with varying degrees of success is that if you look at any modern business, it runs 100 percent on information technology, whether the people who run that company want to acknowledge that. That's the truth these days. If the technology breaks down, everything stops. But you wouldn't often notice from looking at what companies invest in—IT security—relative to their overall dependence on it. And I'm not just talking about money. It's not a spending issue. It's a priority issue.

The data you get back from that sort of thing differs by business vertical pretty vastly. The people in financial services were really woken up by Sarbanes-Oxley, which made them wonder where those accounting numbers came from. "Holy cow, there's a huge pile of software under there that we were assuming was always going to work properly!" I think that got those guys on the stick. The rest of the world and many other disciplines, retail for example, haven't quite gotten the message, although they've been getting huge wake-up calls lately, I suppose.

I still don't think they have gotten the message. I don't care what business you're in, if you're making widgets or selling retail goods or shipping cruise missiles. I think if the people who run the organization haven't undergone a critical shift in thinking about how they protect their systems, they're probably way more compromised than they know. You can usually tell which companies have a clue about this, because the board or the people in charge of operations are asking, above all, one very critical question, which is how much of our scarce IT security resources are we spending trying to keep the bad guys out versus figuring out when they get in as quickly as possible to stop the bleeding.

The issue in computer security is that measuring security is a difficult thing to do. It's something I've been working on for about 20 years, and I think that we're making some progress, but there's no magic x-ray machine where you could x-ray your IT infrastructure and say, "All right, we're good to go," and that makes it a challenge to talk to boards about.

It comes back to something we talked about earlier, which is assumptions. Having that kind of vigilance really does mean having some smart people who are empowered within your organization, who understand the technology and understand the dependence on it, and are empowered to ask questions that make people uncomfortable.

Let's shift gears a little. Malware plays a central role in the things you cover, but most malware is

parasitic—the result of bad software. Why do you think the popular press tends to ignore security engineering, software security, and any solutions to our never-ending parade of hacks, and gleefully talks about the next hack all the time?
I'm probably not the best person to answer this question because I'm always pointing out the problems. It's a lot more fun to point out the problems than it is to try to suggest solutions with a straight face.

I think the world thinks about computer security with a little too much Hollywood thrown in and not quite enough, "We need some of those people from Volvo that are thinking about safety to work on this," as opposed to the NASCAR watchers that are covering spectacular crashes. Do you have any good ways of getting the general public interested in solutions?
It would be nice if we had something in place that would make companies care a little bit more about shipping code that wasn't Swiss cheese. Any sort of liability for that isn't going to happen. I've said this before; I still believe it. Not much changes in that regard until we start to be able to measure the fact that people are dying because of poor software security.

We talked about that in Washington. People at the FDA, at the advisory board that I was talking to, were talking about medical records and patient records and data. I asked, "What if a medical device was used to kill someone?" And that surprised them.
The scary thing about the medical and computer security intersect is that, like most things in information technology, it's not enough just to protect these things; you have to keep them working. You have to keep them available. You have to keep them up. In the context of healthcare, it's way more important. If critical systems stop working, you can't get access to critical information, and people die. It's a real concern.

I want to push on the reporting angle one more time. Why can't old media support in-depth security reporting of the sort that you do? Do you think *The Washington Post* will recover from cutting most of its technology ranks, including yourself, but a lot of other people, too? Or are we going have a jillion "so-and-sos on such-and-such" that are super in-depth but not part of a big juggernaut?
The traditional media, which I like to call the "dead tree edition," does a pretty good job of understanding who their target market is. In the case of my former employer, they correctly identified that their target market cares more about national security than they do about technology or cybercrime. I think the *Post* has really tried to make its mission Washington focused; they want to put a policy slant on stuff, and they want to tell stories that help explain those issues through the lens of Washington, DC. It's not that the other things aren't important; it's just that they may not be priorities for them, which is fine. That was part of the reason we had a fundamental disagreement over this issue.

I just think the technology desk is way thinner than it used to be. I remember when I was doing the original Java security cracking back in the mid '90s, and there were some extremely good reporters who knew their stuff deeply. But when you look around at the tech press today, you don't see that same deep bench.
There are a couple things I want to touch on there. One is that a lot of major publications, a few years back, made a hard-target effort to figure out what they were good at, and then outsource the rest of it to somebody else. So tech coverage tends to be a casualty. On top of that, in the tech publications online that cover this stuff, there's a lot more competition for the same eyeballs, so they're pushing their reporters to produce three, four, five, and sometimes six stories a day.

That's where the fact-checking goes away.
The fact-checking starts to break down. Not only that, but you can forget about enterprise reporting. It doesn't happen in that environment. For me it was an existential thing. I was freaking out when they eliminated my job at the *Post*. I didn't really know what else to do except continue what I was doing. But then I thought, "Well, you know, as long as I'm good at this and people want to read what I have to say, I'll figure it out at some point." And I think I have, to some degree. I guess my focus initially was how to differentiate myself. How do I not get caught up in the story of the day? And then I said, "There's the answer. Just don't shape the story of the day. Just go write stuff that you can't get anywhere else."

Let's talk about the retail vertical. I think it's pretty clear to everybody that retail's kind of hosed. Now that you've given them some religion, do you think they'll go about their recovery correctly? What's your view of how they've done since the big breaks?
It's kind of a mixed review. Some of them have gotten the wrong message. They've expedited plans to move to chip readers and things like that. But that doesn't really address the underlying problem of unencrypted card data, and it's just a giant target on your back. It'll be interesting to see these companies pay a lot of lip service to things like end-to-end encryption that can actually help shift the threat to somebody else. But when it comes down to implementing these things, they always screw it up, and then they figure out ways to cut

corners or save money. At the end of the day, that kind of approach is actually pretty good at shifting the risk to somebody else, but it's also really expensive. Retailers will do anything to keep from spending money on security. I still think most retailers look at a cash register and think of it as physical security. They really don't consider these things that traditionally have been the bailiwick of physical security to be at all related to cybersecurity. And, of course, there is no difference anymore.

So these point-of-sale systems have important design and engineering and implementation aspects that have to be done right.

Right, and they set these things up and say, "Who cares? I mean, they can't connect to the Internet." Well, that's not really a speed bump.

Why do you think the US is rushing to adopt chip and signature (versus chip and PIN) when we all know it's very silly? Even if we did chip and PIN, nobody's listening to Ross Anderson about randomness. What can we do to get these people to do this right going forward?

This is not a new technology by any stretch. We're massively playing catch-up in this country, which should tell you something about our willingness and preparedness to go the extra mile.

Do you think they even understand the difference between chip and PIN and chip and signature?

I think they probably and rightly assume that most consumers don't care. So why should they? I asked Visa this very same question and they wouldn't give me an answer on the record, but they basically said, "We don't want to do anything to impede the acceptance of the payment." So if they have any reticence about requiring customers to use a PIN, well, we don't want to stop them from accepting that less-secure solution.

There's a reason for that, too, which is that Visa has been managing risk for years to nine basis points of fraud (or less), and that's why they can eat all the fraud, or they can make the retailers eat their chunk. So fraud is costing the economy but it's not costing Visa directly, as long as they're managing their fraud exposure properly.

I'm sure they would disagree, but I long ago came to the conclusion that the credit card industry is a bit like Vegas. They're running the house. It's their rules, their casino, and you can blame the guy sitting next to you for playing the cards that he did or not, but everybody does that. They just point the fingers at somebody else and say, "It's this guy's problem." And the card associations are laughing all the way to the bank.

The government likes to over-classify everything. They like to pretend they're the vanguards and that they alone see the true nature of the threat out there.

Let's talk about something even worse. In my view, when it comes to building security in, the government is way behind, maybe five years or more—especially the contractors. In terms of the kind of security that you do—intel and operational security in the cybercrime world—do you see the same lag, or is the government in fact up with everybody else, leading the charge?

That's a really hard question to answer because the people who'd be able to answer it generally don't talk to people like me. Most of these guys that actually know how much of a clue the government has on these topics have security clearance.

I think people hide behind that and it's a bunch of hooey.

The government likes to over-classify everything. They like to pretend they're the vanguards and that they alone see the true nature of the threat out there. I think that has probably, more than anything, stood in the way of most of the rest of the world having a better realization of just how screwed they are.

So you're not seeing any sort of way out that's any better than what I'm seeing. I'm pretty discouraged by the government's approach to security engineering. I'm not kidding when I say they're five years behind. Five years is an eternity when it comes to tech. Imagine what your phone looked like five years ago.

We went through this period where the government stopped investing in innovation and building a better mousetrap when it comes to programming and advancing the state of the art. I hope that's starting to change in the other direction, but I think the government long ago ceded that to the private sector. Not to say that the private sector is a whole lot more advanced, but at least there's a market out there for helping the infrastructure providers and important companies protect themselves a little better.

The last question's an easy one. Writing books is a labor of love. Tell me about writing *Spam Nation*. Did you have a good time?

Parts of it, yeah. Right when I left the *Post*, all this information on spammers and malware writers fell into my lap. I'm like, "Okay, well, this will be a book at some point." I just couldn't see clearly through to the other side, because it was this

Mount Everest of information. I'm not one of those reporters to whom the writing aspect of things comes naturally. When I first started writing for the *Post* and first started getting taken seriously and getting good treatment in the paper, I had one of those editors who was an old-fashioned newsman, kind of a grouchy guy, but a fantastic individual. I'll never forget, he called me into his office and he says, "Brian, you're a pretty good reporter, but you're a god-awful writer. But don't take that the wrong way. Most reporters aren't both of these things at once; they have to work at one or the other." And he was right.

The writing part of things is sometimes more challenging for me than I would like, but all in all it was a lot of fun. This is a good example: I work in an area where my training at the *Post* actually has been invaluable. Writing about technology for a general audience is hard. Writing about security and not totally screwing it up and getting it to be understandable by mere mortals is darn near impossible.

You're doing a great job at it, so thank you for doing what you're doing. We need more people that can do that sort of thing. Otherwise, those of us who are trying to get this accomplished won't get the resources and authority we need to get stuff done. People have to understand the problem.

The Silver Bullet Podcast with Gary McGraw is cosponsored by Cigital and this magazine and is syndicated by SearchSecurity. ■

Gary McGraw is Cigital's chief technology officer. He's the author of *Software Security: Building Security In* (Addison-Wesley 2006) and eight other books. McGraw received a BA in philosophy from the University of Virginia and a dual PhD in computer science and cognitive science from Indiana University. Contact him at gem@cigital.com.

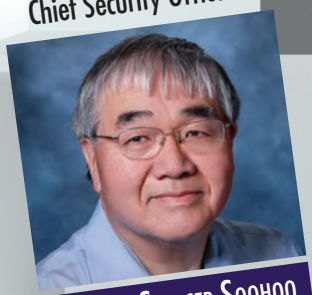
This article originally appeared in IEEE Security & Privacy, *vol. 13, no. 1, 2015.*

IEEE computer society

ROCK STARS OF CYBER SECURITY

Chris Calvert
Global Director, HP
Enterprise Solutions Products

Marcus H. Sachs
Senior Vice President,
Chief Security Officer (NERC)



Dr. Spencer Soohoo
CSO/Director, Scientific Computing
Cedars-Sinai Medical Center

Win the New Cybersecurity War with the New Rock Stars of Cybersecurity

Cybercrime is no longer a matter of credit card breaches. Cybercriminals are now trying to take down countries as well as top companies. Keep your organization safe. Come to the premier, one-day, high-level event designed to give real, actionable solutions to these cybersecurity threats.

Learn from and collaborate with the experts—

27 October 2015
The Fourth Street Summit Center
San Jose, CA

REGISTER NOW

Early Discount Pricing Now Available!

computer.org/cyber2015

IEEE

Innovations in Ubicomp Products

Editor: Albrecht Schmidt ■ University of Stuttgart ■ albrecht@computer.org

Crowdfunding for Ubicomp Products:

Interviews with Amanda Williams and Khai Truong

Albrecht Schmidt, *University of Stuttgart*

In the pervasive computing research community, many of us build prototypes—systems and prototypes can be a central part of research papers. Demo sessions at conferences showcase new ideas for devices and systems using often crudely built prototypes, but these help us study the user experience. Such prototypes generate excitement among fellow researchers, who might then state that "this should really be a product" or "people would certainly want to buy this." Nevertheless, few concepts shown as prototypes at conferences directly transition into products available in the market, usually because the process of turning something into a product isn't easily achieved by individuals.

Recently, however, crowdfunding has started opening doors, enabling more people to create products for a potentially global audience (see the "Crowdsourcing Platforms" sidebar). It gives developers and researchers a platform to interact with potential customers directly and to see whether people want to buy the product. Furthermore, it makes it possible to produce niche products for specific user groups or bootstrap a company with an initial product. Over the last few years, many interesting products have become available that would have been unlikely to hit the market without this instrument.

Yet crowdfunding isn't a silver bullet. If you look closely through the listings at Kickstarter.com and Indiegogo.com, you'll find many ideas that weren't successful. Central to understanding whether a concept could be turned into a viable business is determining how much a customer would be willing to pay for the product or service and whether that amount covers the cost of building the product. Additionally, achieving successful funding is only the start. In particular, for those ideas that involve physical goods, the inventors then must source the components, manufacture at scale of 100 to 1,000, and deliver the product to various countries. Often, their initial costing was flawed; they simply didn't factor in all costs. Even for buyers, although it's great to get a unique piece of technology that only a few hundred people around the world have, in many cases, it requires patience (because of timelines that aren't met) and hidden costs (such as import taxes).

To learn more, I asked Amanda Williams and Khai Truong to share their experience with crowdfunding (see the related sidebar). Both are active researchers in our community who have created successful crowdfunding projects. The full interviews are available in my blog.[1]

WILLIAMS: BUILDING CLYDE

Williams and her colleagues ran a kickstarter campaign for a smart, programmable lamp named Clyde (see Figure 1).

AMANDA WILLIAMS AND KHAI TRUONG

Amanda Williams is cofounder and CEO of Fabule Fabrications. She's in charge of creating beautiful interactions, beautiful hardware, and customer development. She has worked at Xerox PARC, Adobe, Intel Research, and Microsoft Research. Williams has a BS in symbolic systems from Stanford University and a PhD in information and computer sciences from UC Irvine. Indecisively, she loves both qualitative user research and hardware design. Contact her at amanda@fabule.com.

Khai Truong is an associate professor at the University of Toronto. His research lies at the intersection of HCI and ubiquitous computing—specifically, examining the mutual impact of usability and technical constraints on the design of applications and interaction techniques for novel, off-the-desktop computing systems that might be commonplace in 5 to 10 years. Truong received his PhD in computer science from the Georgia Institute of Technology. Contact him at khai@cs.toronto.edu.

2376-113X/15/$31.00 © 2015 IEEE Published by the IEEE Computer Society

INNOVATIONS IN UBICOMP PRODUCTS

CROWDFUNDING PLATFORMS

There are many crowdfunding platforms, and technology products are only a small portion of the items presented. Examples of platforms that support campaigns and products are www.kickstarter.com and www.indiegogo.com. There are also platforms that target (micro)-investors, such as www.crowdcube.com. For a detailed overview, see "Democratizing Entrepreneurship: An Overview of the Past, Present, and Future of Crowdfunding."[1]

There is also a wide variety of crowdfunding platforms. Some target a certain country or region, while others are for specific products (such as games) or campaigns (such as art or music). Yet others have specific aims, such as boosting business in local communities or in developing regions of the world. Choosing the right platform is important, and depending on what should be funded, the biggest platform might not always be the best one.

Funding goals for products are typically set at a point where the inventors assume they have a break-even point. Here, too, there are differences between the platforms and the strategies people use. Some platforms only provide funding if the goal is met. Other platforms leave it to the person running the campaign as to whether or not to try to do it with less money. Furthermore, because the listings in some crowdfunding platforms use the percentage of funding received as a measure for popularly, some ask for a very low number in order to appear massively overfunded.

REFERENCE

1. Z.D. Kaufman, T.W. Kassinger, and H.L. Traeger, "Democratizing Entrepreneurship: An Overview of the Past, Present, and Future of Crowdfunding," *Bloomberg BNA Securities Regulation & Law Report*, vol. 45, no, 5, 2013, pp. 208–217; http://ssrn.com/abstract=2211698.

Figure 1. Clyde, a programmable lamp.

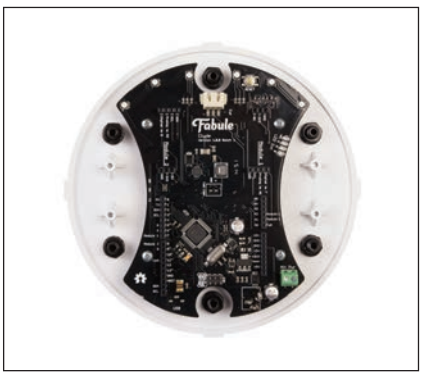


Figure 2. The bottom view of Clyde, showing the Arduino compatible circuit board.

According to Williams, "Clyde is a unique, jellyfish-like desk lamp that does both bright task lighting and colored ambient lighting." She further explains that "he comes equipped with environmental sensors that automatically detect changes, and he reacts to the changes, so whether you want him to be 'touchy feely' or 'afraid of the dark,' you can change his personality without programming." In other words, you can change the personality by changing the sensors attached. Clyde is Arduino compatible and open source, so it can also become a starting point for learning to program (see Figure 2).

Why Crowdfunding?

When I asked Williams why she used crowdfunding, her answer was multifaceted. First, she explained that most start-up funding focuses on software companies, and designing software is very different from making a product that includes both hardware and software. Software costs mainly comprise the developer's time, so US$25,000 to $50,000 in seed funding can get you started. She goes on to say that "this is virtually impossible for a hardware start-up, no matter how disciplined the founders are, because they will have to deal with the costs of materials, manufacturing, and shipping physical products."

Another issue is certifications, which "are a significant cost and a real hurdle for people who don't have prior manufacturing experience." She said that the "costs vary depending on the product, but I think it's pretty rare to be able to do this well for less than $100,000, and even that is a stretch for all but the easiest hardware projects."

This is why it is hard to find investors and why crowdfunding offers a good alternative. "It can provide the money you need to cover non-recoverable expenses, and if it goes well, it shows potential investors that you have a market for your product."

Yet her experience shows that crowdfunding is not without risk. "It's also extremely common for first-time project creators to underestimate their expenses." She said that while people worry that their campaign will fail, that's not the biggest problem. "The worst outcome is if it succeeds [and] you take on the obligation to deliver what you promised [but] then run out of money without delivering."

Testing Your Idea

Williams and her team tested the water for their ideas at Maker Fairs, and the feedback reshaped the product they created significantly. They moved from a LED solder kit to a lamp that also appealed to a less technical audience. The month prior to their Kickstarter campaign, they were accepted into a hardware accelerator program (HAXLR8R) and spent four months

in Shenzhen to develop their final product.

In describing the experience, Williams said "we had access to fabrication techniques that we couldn't have afforded back home. We could cheaply CNC [computer controlled milling] or vacuum cast really professional-looking prototypes, the kind of thing that you could imagine on a store shelf. We could also prototype more sophisticated electronics much faster."

Their Kickstarter campaign, which lasted from May through June 2013, raised almost US$150,000—more than triple the stated goal.

Manufacturing, Shipping, and Support

Once the campaign succeeds, more work rolls in. Williams recalls, "because of our inexperience, we underestimated how much work was still required to design for manufacture, set up a test plan the factory could follow, finalize and purchase everything on the bill of materials, design supplementary materials like packaging and manuals, test for electromagnetic compatibility, and ship to 30 different countries." She noted that during this time, "we had to re-design our board, because the microprocessor we were using suddenly shot up in price." She said that in hindsight, "a lot of this stuff looks obvious, but our experience wasn't really atypical for first-time product creators." They shipped Clyde in July of 2014—six months later than initially planned.

Williams further explained that this isn't the end. "One typically thinks that shipping is the finish line, but the reality is, that's when you start seeing a lot of customer support work." She said that because Clyde is a computational product, "you have to answer questions about the setup, the drivers—if anyone wants to plug it in to program from their computer—how the sensors work, how to assemble it, and so on." She also said that the product can get held up in customs, or sometimes packages go missing and need to be replaced. These things are time consuming for the first few months, which is difficult because, as Williams puts it, "customer support is never really an engineer's favorite task."

Furthermore, she said that although it's great to already have customers in place at the idea stage, it also adds a lot of pressure. "Most backers are really supportive. They back campaigns not just to acquire an end product, but to engage with the process of invention and design and production. So you can get great feedback from people regarding what features they want you to prioritize." However, Williams explained that "even if people are nice, you feel the weight of your promise to give them what they want, and you feel pretty awful about every delay."

Viable Niche Products

Crowdfunding can really change how we make ubiquitous computing products. It will allow niche products to become economically viable. According to Williams, "1,100 was a pretty good number to produce. If we'd been required to do a minimum order quantity of 10,000, it just wouldn't have happened, but this way, about 1,000 people get a neat little product that powerfully appeals to them."

It's exciting to see that this could bring the market nearer to research. Williams said that "this new set of capabilities makes the boundary between research and product development a lot more porous. You could conceivably do a run of a hundred or so devices for a research project, and deploy them in the wild for validation." This can help reduce the risk. "Universities and industrial research labs spin off research projects into start-ups sometimes; crowdfunding and small-run manufacturing can allow you to validate these projects before taking a big risk on them."

Move Fast

With crowdfunding, inventors make their ideas public long before they have a product. Are there precautions one can take to avoid competition? According to Williams, "protecting our IP wasn't a huge deal for us, but it can be for others. At $150,000, we were below the threshold of interest for most copycats to try to reverse engineer our product—if we'd raised more than half a million, I'd have been a lot more worried."

Also, they made their product open source, because preventing copying wasn't a huge priority. For those who need to prevent copying, she understands that patents don't always work and offered two suggestions. The first was to "move fast. Invent new things faster than people can copy your old things." Second, she said "don't make incredibly simple pure-hardware products. Those are the easiest things to copy." She went on to add that "the hardest thing for a competitor to steal is a fantastic user community."

New Ideas

And what's next? "Manufacturing Clyde to fulfill our Kickstarter campaign was a marvelous learning experience. Once we finished that project, we realized it had given us a ton of ideas for tools that we wished we'd had during the manufacturing process." Williams explained that "effective collaborative tools for manufacturing just didn't seem to exist yet, at least not in any form usable to the increasing number of start-ups and small-scale creators that need them."

So she's now working on an exciting new project for a hosted Bill of Materials management tool (see fabule.com). The Bill of Materials is a keystone document that is an absolutely crucial point of communication between a creator and their manufacturer at all stages of production—from prototyping to sourcing to assembly and testing and subsequent manufacturing runs. According to Williams, "Right now, everyone is using Excel, spending hours doing tedious and error-prone revisions and maintaining multiple parallel copies for different purposes and different audiences." With their new product, they will offer a means for collaboratively using a Bill

INNOVATIONS IN UBICOMP PRODUCTS

Figure 3. The Minuum onscreen keyboard. This text-input method reduces a full-size keyboard down to a single row of keys.

of Materials. Inventors and designers receive product guidance in terms of how to list and specify the materials they need, and some information is automatically filled in. Manufactures receive the information in the "right" format and can then add comments and suggest changes. The aim is to minimize the effort needed to create the lists and to reduce errors during collaboration.

TRUONG: CREATING AN ONSCREEN KEYBOARD

Truong had a successful Indiegogo campaign to fund parts of the development of a new type of onscreen keyboard: Minuum (see Figure 3). As Truong explains, "Minuum is a text-input method that reduces a full-size keyboard down to a single dimension, or row, of keys. For onscreen keyboards, this reduces the amount of space that would be taken up by the input method, giving more real estate to the rest of the application." (For more information, see http://minuum.com.)

The challenge is letting users still type quickly, because, as Truong says, "it becomes harder for them to precisely hit keys on this reduced-sized keyboard." He thus created a product with a backend that has "a disambiguation engine that predicts what the user is typing without requiring the user to always hit the exact keys in the word that they are typing." He explains that "by reducing text entry to being simply selection of keys placed on a single row, this same mode of typing can be carried over to a variety of other platforms and devices."

Benefits of Crowdfunding

Besides the financial benefits of crowdfunding, Truong sees several other reasons that might not be obvious at first. "One advantage with crowdfunding is that it is full of people who want to support new ideas. In a sense, these are people who are likely to be early adopters of the technology."

A second advantage is "the buzz that it provides for the product." Truong says that for Minuum, during the crowdfunding campaign, "YouTube videos about the product were watched by over a million people." Then the media "caught wind of the campaign and helped to promote awareness as well."

Finally, Truong says that crowdfunding provides "feedback about the product/idea from the potential users while the development of the product is still occurring. That can help to shape the product."

From Idea to Product

The product came out of working with students in an HCI course at the University of Toronto. The initial ideas were centered on a gesture-based keyboard. This project showed that by using a disambiguation approach, it was possible to handle the imprecisions well. According to Truong, "using an iterative design process over the course of a year, we continued to develop, evaluate, and refine the prototype in numerous ways, including dramatically improving the disambiguation algorithm and supporting more input gestures." He said that the most interesting part of the work "was the disambiguation engine's ability to support imprecise typing." He explained that he had "attempted another start-up a few years earlier on the 1Line Keyboard [where he] stumbled into an interesting user problem: how to give the user more screen space while typing." The Minuum keyboard embodies these two concepts.

One of the reasons they decided to go with crowdfunding was to get additional funding for doing the development. "Basically, we had been working in stealth mode.... It wasn't a finished product, but it was close. This meant that we could afford to ask for not a huge amount of money from the crowdfunding campaign to complete the product."

Timing is difficult. Even for running a campaign, many things need to be prepared: "We had to start to put together a website; develop the concept video; bring on people to help with marketing, communications, and press releases; and so on. I remember we had set the launch date to be early in the year, but we ended up delaying it because getting all these materials ready took a significant effort."

Furthermore, there is a big difference between a research prototype and a product that can be released to potentially hundreds of thousands of people. As

This article originally appeared in IEEE Pervasive Computing, *vol. 14, no. 3, 2015.*

Truong describes, "coding and testing of the product, tracking of bugs… is quite important and requires a lot of resources, including time. So while it seemed like a lot of time often goes by before a product actually gets released after we hear about it initially, it's primarily because taking it from a concept or research prototype to a deliverable product still requires a significant amount of development, testing, and debugging."

User Feedback

The interaction with the customers is very tight in a crowdfunding model, and it clearly has an impact on the product. According to Truong, "user feedback at any stage of development is always valuable. The most important one that crowdfunding provides is whether potential users find the overall concept appealing." He says this differs from traditional product development, where you build the product and then "put it on the market and see if people might buy the product." Crowdfunding lets companies "get insight into whether such customers exist already." He said that during the campaign, "funders will discuss features that they are excited about and what they want and hope to have included in the product.… All of this information helps us to understand user requirements and prioritize features."

As Truong sees it, crowdfunding can change how we make ubiquitous computing products by letting us "take concepts that we have done some basic research and development on and turn them into actual products." Previously, we would have had to first secure the funding. "Even if we knew an idea could work and be turned into a product, could we find people who believed in us enough to fund that development? Crowdfunding allows for enough people who believe in the idea to share some of that cost." He said that it also helps demonstrate the market size, which helps the company grow.

Yet as Williams explains, there's still a lot of work involved. "While I think crowdfunding is giving us a lot of opportunities we didn't have before, I don't think it's exactly bringing about a utopian world of easy hardware." She says that although it's easier, it's not easy. "Manufacturing even a small run of 1,000 or so requires a skill set that's really distinct from research and prototyping in ways that we often fail to appreciate until we're deep into the process."

REFERENCE

1. A. Schmidt, "Crowdfunding—Interviews with Amanda Williams and Khai Truong," blog, 1 June 2015; http://albrecht-schmidt.blogspot.com/2015/06/crowdfunding.html.

Descaling Your Scrum

Jim Coplien, Scrum Foundation

Every now and then I must remove scale from my tea kettle. It accumulates with use and slows the heating of the water over time. It may change the flavor of the water a bit but doesn't make a lot of difference other than creating lime.

The same basic principle applies when businesses scale their organizations. A successful business accumulates more and more staff when there is enough money to go around. Many firms then descale when hard times come. *Angry Birds* developer Rovio Entertainment laid off 14 percent of its staff in October 2014, cutting back to 700 employees. It's hard to not compare the company to rival Supercell, which earned five times Rovio's revenue with only 132 staff members.

Now, scaling agile is all the rage, perhaps because large late-adopter companies are now coming on the Scrum scene with the misconception that they must paint Scrum onto all of their thousands of engineers. And such companies rarely stand still but instead are always looking to grow.

Meanwhile, of course, a topic like scaling agile is a newly drilled well of consulting, training, and yes, even certification.

Why do so many companies scale up instead of lean down? Often it's because they're afraid the competition might outperform them. In addition, in the contemporary software market, there is an unspoken principle that more features are better.

When many enterprises realize great profits, they share the wealth by hiring more employees. Many of the new workers end up taking roles that don't add value to the production process. In our book *Organizational Patterns of Agile Software Development*, Neil Harrison and I call these "deadbeat roles." Our research found that these roles are common in many large organizations. In fact, at least one of my Japanese clients calculates managers' bonuses in part on the basis of the number of people reporting to them. This type of incentive for scaling up is the antithesis of lean.

David Graeber, professor of anthropology at the London School of Economics and Political Science, says that most jobs in a modern economy

 Published by the IEEE Computer Society 2376-113X/15/$31.00 © 2015 IEEE

represent deadbeat roles. He claims they don't add real value but just keep people employed in an economy that has become so efficient as to not require "full-time" workers.

Scaling staff can make sense for endeavors in which work is perfectly able to be partitioned. If 10 people can transport one stone for the pyramid, then 100 can transport 10 stones.

Coordinated work, on the other hand, has a sweet spot beyond which hiring more people generates more coordination costs than the additional horsepower can cover. Adding people to work on one system means more teams, which means more handoffs between teams. Thus, 20 people can't sort a deck of cards faster than five people can. Intellectual work in complex domains requires coordination that lowers the sweet spot, and the sweet spot is already very low in software development.

Most organizations—like my Japanese client—start small and grow to feed more mouths as they give into social pressures and the belief that there is power and success in growth. But small is beautiful. If you start small, stay small. You should be very afraid if a consultant offers to help you scale.

I'll continue these thoughts in the next *ComputingEdge* issue.

Jim Coplien is a partner with the Scrum Foundation, a provider of certified Scrum training and consulting. He is also a partner with Gertrud & Cope, a Danish company that specializes in agile and lean software development. Coplien was a pioneer in practical object-oriented design in the early 1990s and is an author and trainer in the areas of software design and organizational improvement.

CLOUDERA, INC. is recruiting for our San Francisco, CA office: Engineering Manager: manage & mentor a team of software engineers for professional & technical growth including having weekly 1:1s, doing career development, managing expectations & performing quarterly performance reviews. Mail resume w/job code #33224 to: Cloudera, Attn.: HR, 1001 Page Mill Rd., Bldg. 2, Palo Alto, CA 94304.

PROGRAMMER ANALYST: Design, develop, test & implement software applications using knowledge in C++ ,Matlab ,OBIEE 10.1.x or Later, Oracle BI Publisher, Informatica Power Center 8.x, Oracle10g/9i, PL/SQL, UNIX, UNIX Shell Scripting, BI Apps 7.9.6.x, DAC, SQL Developer, SQL Server 2005 and Windows 2003 Server. Must be willing to travel & reloc. Reqs MS comp sci, eng or related. Mail resumes to Nartal Systems, Inc. 2650 Route 130, Suite # E, Cranbury, NJ 08512.

PROGRAMMER ANALYST. - Design, develop, deploy, test & implement application s/w in Agile environment utilizing knowledge of C#,VB.NET, ASP.NET 5.0, MVC 5.0 Javascript, Angular JS,WCF, EF6, Winforms, WPF, Visual Studio 2013, Sql Server 2008, T-Sql, MS Visio 2010, Windows Server 2008, Microsoft IIS 7.0. Must be willing to travel and reloc. Reqs MS in comp sci, eng or rel. Mail resumes to Code Ace Solutions Inc. 50 Cragwood Road, Ste 217, South Plainfield, NJ 07080.

SIEMENS PLM SOFTWARE INC. has an opening in Maryland Heights, MO for Applications Developer to lead & manage large projects & develop highly scalable OO applications using Core Java and J2EE technologies. Email resumes to PLMCareers@ugs.com & refer to Req#145417. EOE.

ENGINEER: Broadcom Corporation, the leading provider of highly integrated complete system-on-a-chip solutions for digital and satellite cable set-top boxes, cable and DSL modems, residential gateways, high speed transmission area networking, home and wireless networking, cellular and terrestrial wireless communications, VoIP gateway and telephony systems, broadband network processors, and server solutions seeks all levels of Engineers/Scientists in Irvine, CA, Santa Clara, CA, San Jose, CA, Sunnyvale, CA San Diego, CA, Matawan, NJ, Chandler, AZ, Duluth, GA, Andover, MA, Edina, MA, Federal Way, WA, Horsham, PA and Austin, TX: Test (ENG472), Product Engineering (ENG476), RF/Wireless (ENG484), Hardware Development (ENG502), Electronic Design (ENG506), IC Design (ENG507), Software Development (ENG510), Software Applications (ENG514), Software Systems (ENG516), Software Quality Assurance (ENG518), Systems Design (ENG520), Firmware (ENG521), Product Applications/Systems Integration (ENG584), DSP (ENG509), Configuration/Release (ENG519), Packaging (ENG524), Design (ENG537), Field Applications (ENGS87), Layout Design (ENG0313), Product/Prod. Line Manager (ENG586), Systems/Database Administrator (ENG641), Applications Programmer (ENG6555), and Process Development (ENG526), Sales/Business Development (ENG567), IT Security Analyst (ENG658); Business Systems Analyst (ENG646); Product Development (ENG558); Engineering Systems Analyst (ENG533) Compliance Engineer (ENG463); and Manager Enterprise Application (ENG347) (Oracle, SAP, Baan, etc). Education/experience requirements vary by position/level. Some positions may require domestic and/or international travel. Must have unrestricted right to work in U.S. Mail all resumes to HR Ops Specialist, 5300 California Avenue, Bldg. 2 #22108-B, Irvine, CA 92617. Must reference job code.

PROGRAMMER ANALYST - design, develop, test & implement application s/w utilizing knowledge of C#, Microsoft Sharepoint Server 2013, Windows Sharepoint Services(WSS) 3.0, Sharepoint designer 2013,2010, Oracle, Microsoft SQL Server, Visual Studio.NET 2010, WCF Services, Web services, XML, XSLT, XSD, XPATH, Unix & Windows 00/NT. Must be willing to travel & reloc. Reqs MS in comp sci, eng or rel. Mail resumes to Strategic Resources International, Inc. 777 Washington Rd, Suite 2, Parlin, NJ 08859.

BMC SOFTWARE INC. has the following openings: Staff Sales Operations Analyst, Req#15001478 in Houston, TX to develop new reporting solutions for Global Services; and Principal Product Developer Req#15001479 in San Jose, CA to perform product design and development in relation to CLM. Mail resumes to Attn: Req # of interest, Olivia Delgado, BMC Software, 91 East Tasman Drive, San Jose, CA 95134-1618.

SOFTWARE ENGINEER (MULT. Openings) sought by Nexius Insight, Inc. in Reston, VA w/a MS in Info Systms or rltd. Work on Bus. Intelligence tools, incl Tableau for Data visualization. Work on Dbase dvlpmt, incl writing queries. Load Data. Prep web servers, incl installation of Java, Apache Tomcat servers, & Java prgmg. Resp for deployment of web based UT prgm in Sprint project. Work on data manipulation & dvlpmt for different demo projects. Mail resumes to Nexius Insight, Inc., 1301 Central Expressway S., Ste 200, Allen, TX 75013 Attn: HR.

DIR, PRACTICE SVCS. (NY, NY & locs throughout the US) Mnge team of Svc Arctcts & Cnsltnts charged w/dsgn & dlvrng access control & DLP solutions. Mnge staffing efforts & annul targets & act as SME for tech & sales inquiries. REQ: Bach Deg or for equiv in CS or Bus or rel field + 5 yrs of prog exp in job &/or rel occup. Will accept a Master's deg or for equiv in CS or Bus or rel field + 3 yrs of exp in job &/or rel occup. Must have exp w/CA DataMinder solution def, archtre, dsgn & implmntn; CA Privileged Identity Mgr solution def, archtre, dsgn & implmntn. Engaging in sales process invlvng access control & data loss prevention solutions; Mngng solution specific def, archtre, dsgn & implmntn of access control & data loss prevention solutions; Freq visits to unanticipated client sites; Wrk fr home anywhere in US. Send resume to: Althea Wilson, CA Technologies, One CA Plaza, Islandia, NY 11749, Refer to Requisition #103841.

 Published by the IEEE Computer Society 2376-113X/15/$31.00 © 2015 IEEE

PROGRAMMER ANALYST - design, develop, test and implement Oracle EBS Applications utilizing knowledge of Sql, PL/Sql, Workflow, BI Publisher, OAF/ADF, java, Spring, Hybernate, Unix and modules INV, WIP, BOM, ASCP, DEMANTRA, GL, OAH, FA, AR, TCA, PO and OMr.Must be willing to travel & reloc. Reqs MS in comp sci, eng or rel. Mail resumes to Nitya Software Solutions Inc. 9690 South 300, Ste 319, Salt Lake City, Utah 84070.

PROGRAMMER ANALYST: design, develop, test & implement application s/w utilizing knowledge of n-tiered distributed software applications, EDI, ASP. NET, MVC, C#, LINQ to SQL, LINQ to application, Stored Procedures, JavaScript, J Query, HTML5, CSS, and AJAX, Angular JS, Telerik, Web Services, SOAP, WSDL, XML, XSLT processing, AGILE, Test Director, Unix & Windows 00/NT. Must be willing to travel & reloc. Reqs MS in comp sci, eng or rel. Mail resumes to Strategic Resources International, Inc. 777 Washington Rd, Suite 2, Parlin, NJ 08859.

SENIOR CONSULTANT F/T (Fishkill, NY) Position involves travel to various unanticipated worksites up to 100% of the time anywhere in the United States. Must have Master deg or the foreign equiv in Comp Sci, Comp Engg, or related with one (1) yr of exp designing, building, or fixing & supporting integration interfaces that meets business requirements using SAP BW/BI, Business Objects Web Interface reports, Crystal reports, BEx Query designer, Cross module expertise in SAP FI/CO, FI-SL, FI-GL, LO, HR, SCM and CRM in fairly large size environments. Provide comprehensive consultation to business unit & IT management and staff at the highest technical level on all phases of application. Respond timely to issues reported. Responsible for Full Life Cycle Implementation includes GAP analysis, requirement gathering, blue printing, development, testing and reporting of Business Objects suite of products. Provide leadership in recommending and implementing continuous process improvement, education and training requirements to management staff. Send resume: Novisync Solutions, Inc., Recruiting (SM), 300 Westage Bus Ctr Dr, Ste 350, Fishkill, NY 12524.

SAILPOINT TECHNOLOGIES INC. has an opening in Austin, TX for Software Engineer to maintain & integrate SailPoint IdentityNow with existing apps. Interested applicants mail resumes to Attn: A. Krupa (SP14), SailPoint, 11305 Four Points Drive, Bldg 2, Ste 100, Austin, TX 78726.

CALIFORNIA STATE UNIVERSITY, EAST BAY
FACULTY EMPLOYMENT OPPORTUNITY
DEPARTMENT OF COMPUTER SCIENCE

FULL-TIME TENURE-TRACK OAA Position No. 15-16 CS-DATA/CLOUD/CORE-TT (2 Positions)

THE UNIVERSITY: California State University, East Bay is known for award-winning programs, expert instruction, a diverse student body, and a choice of more than 100 career-focused fields of study. The ten major buildings of the Hayward Hills campus, on 342 acres, contain over 150 classrooms and teaching laboratories, over 177 specialized instructional rooms, numerous computer labs and a library, which contains a collection of over one million items. The University also has campuses in Contra Costa County, Online, and in Oakland, California. With an enrollment of approximately 13,000 students and 600 faculty, CSUEB is organized into four colleges: Letters, Arts, and Social Sciences; Business and Economics; Education and Allied Studies; and Science. The University offers bachelor's degrees in 50 fields, minors in 61 fields, master's degrees in 37, and 1 doctoral degree program. http://www20.csueastbay.edu/

THE DEPARTMENT: The Department of Computer Science has over 20 full-time faculty members, with a wide range of backgrounds and interests. The faculty is committed to teaching its undergraduate and Master's level students. In a typical quarter, the Department will offer over 30 undergraduate and about 20 graduate classes. Classes are offered both in day and evening. Classes are generally small, with many opportunities for faculty-student contact. The Department offers a variety of degrees: B.S. in Computer Science (with possible options in Networking and Data Communications, Software Engineering, or Computer Engineering), and M.S. in both Computer Science and Computer Networks. Currently, there are more than 350 undergraduate majors and over 350 students in the M.S. programs.

DUTIES OF THE POSITION (2 positions currently available): Teaching courses at B.S. and M.S. levels, curriculum development at both levels, and sustaining a research program. Please note that teaching assignments at California State University, East Bay include courses at the Hayward, Concord and Online campuses. In addition to teaching, all faculty have advising responsibilities, assist the department with administrative and/or committee work, and are expected to assume campus-wide committee responsibilities.

The ideal candidate for this position is able to:

1. Teach a wide range of computer science courses including most or all of the core subject matter at both the undergraduate and graduate level. (considering all areas of computer science, capable of teaching in emerging areas).
2. Support offerings for undergraduate C.S. students including teaching courses, developing the undergraduate curriculum, and engaging undergraduate students in research.
3. Support offerings for graduate C.S. students – teaching courses, guiding M.S. theses, developing the graduate comprehensive examination, etc.
4. Advise Computer Science students.
5. Participate in departmental activities such as curriculum development, assessment, outreach, etc.
6. Develop and continue ongoing research activities, service and leadership.

RANK AND SALARY: Assistant Professor. Salary is dependent upon educational preparation and experience. Subject to budgetary authorization.

DATE OF APPOINTMENT: Fall Quarter, 2016

QUALIFICATIONS: Applicants must have a Ph.D. in Computer Science by September 2016. Applicants who can teach undergraduate and master's level courses in most or all of the core subject matter in computer science. Candidates should demonstrate experience in teaching, mentoring, research, or community service that has prepared them to contribute to our commitment to diversity and excellence. Additionally, applicants must demonstrate a record of scholarly activity. This University is fully committed to the rights of students, staff and faculty with disabilities in accordance with applicable state and federal laws. For more information about the University's program supporting the rights of our students with disabilities see: http://www20.csueastbay.edu/af/departments/as/

APPLICATION DEADLINE: The deadline for applications is October 31, 2015; review of applications will begin November 1, 2015. The position, however, will be considered open until filled. Please submit a letter of application, which addresses the qualifications noted in the position announcement; a complete and current vita at https://my.csueastbay.edu/psp/pspdb1/EMPLOYEE/HRMS/c/HRS_HRAM.HRS_CE.GBL

Additionally, please email graduate transcripts, 3 letters of recommendation, 3 references, a statement of teaching philosophy, and evidence of teaching and research abilities to the Computer Science Search Committee to this email: cssearch@mcs.csueastbay.edu.

A detailed position announcement is available at: http://www20.csueastbay.edu/about/career-opportunities/

NOTE: California State University, East Bay hires only individuals lawfully authorized to work in the United States. All offers of employment are contingent upon presentation of documents demonstrating the appointee's identity and eligibility to work, in accordance with the provisions of the Immigration Reform and Control Act. If you are considered as a finalist for the position, you may be subject to a background check.

As an Equal Opportunity Employer, CSUEB does not discriminate on the basis of any protected categories: age, ancestry, citizenship, color, disability, gender, immigration status, marital status, national origin, race, religion, sexual orientation, or veteran's status. The University is committed to the principles of diversity in employment and to creating a stimulating learning environment for its diverse student body.

TECHNICAL LEAD F/T (Poughkeepsie, NY) Position involves travel to various unanticipated worksites up to 100% of the time anywhere in the United States. Must have Bach deg or the foreign equiv in Engg, Electronic Engg, Electrical & Electronics Engg or related w/5 yrs of progressive exp in managing & leading a team of 3 developers in application analysis, design, development, implementation and testing of Java/J2EE-based software applications through full product development life cycle and release process. Provide project plan estimation and rollout strategy in collaboration with the project manager. Mentor the junior team members by providing technology knowledge transition. Create design documents, data conversion documents, technical specifications, class diagram flowchart based on requirements session. Take ownership of the system and programming documentation. Manage production tickets workload among developers and resolve them in timely fashion. Provide subject matter expertise and implement the code using following tools/technologies: Java/J2EE, Servlets, Java Server Pages, Hibernate, Enterprise Java Beans, AJAX methodology, Web Services, JDBC, Oracle, SQL Developer, IBM Websphere, IBM Rational Application Developer and Eclipse. Send resume: Indotronix Int.l Corp., Recruiting (RC), 331 Main St, Poughkeepsie, NY 12601.

PROGRAMMER ANALYST - design, develop, test & implement application s/w utilizing knowledge of C#, Microsoft BizTalk Server 2010, BAM, Oracle, MS SQL Server, Visual Studio.NET 2010, WCF Services, Web Services, XML, XSLT, XSD, XPATH, Unix & Windows 00/NT. Must be willing to travel & reloc. Reqs MS in comp sci, eng or rel. Mail resumes to Strategic Resources International, Inc. 777 Washington Rd, Suite 2, Parlin, NJ 08859.

R&D MANAGER sought by GN Hearing Care Corporation (dba GN Otometrics North America) in Schaumburg, IL. Manage & dvlp all research & dvlpmt functions & staff w/in North American R&D Team. Serve as project mgr for all projects running & in pipeline. Direct s/ware team incl. Architects, S/ware Engrs, H/ware Engrs, & oversee dvlpmt of all s/ware for Otometrics' vestibular & audiological measurement products. Reqs: Master of Engg or foreign equiv. deg in Comp Sci & Engg, + 2 yrs exp. dvlpg & supporting new s/ware applications & products for vestibular & audiological measurement using: Agile Product Lifecycle Mgmt s/ware; DevSuite Application Lifecycle Mgmt s/ware; Grand Avenue s/ware for medical products; Visual Studio; Passolo; Perforce; OTOsuite Vestibular balance suite & OTOsuite for knowl sharing purposes. To apply, send resumes & cvr ltr to Dan Connelly, Manager, HR at dconnelly@gnhearing.com.

CLOUDERA, INC. is recruiting for our Palo Alto, CA office: Solutions Consultant: Work with Cloudera customers to install & implement CDH platform & develop solutions on the Cloudera infrastructure. Mail resume w/job code #36233 to: Cloudera, Attn.: HR, 1001 Page Mill Rd., Bldg. 2, Palo Alto, CA 94304.

BUSINESS ANALYSTS in McLean, VA sought by established IT firm. Qualified candidates will have Master's degree in Comp. Sci./Electrical Eng. field and 12 months as Systems Analyst, Programmer Analyst, or related position. Will accept candidates with Bachelor's degree in Comp. Sci./Electrical Eng. field and 5 years of relevant experience in the mentioned occupations. Experience with databases and data warehousing concepts required. Position subject to relocation to various unanticipated office and client sites throughout U.S. Send resumes to: HR Department, Supremesoft Corporation., 8201 Greensboro Dr., Ste. 300, McLean, VA 22102.

APPLICATION SUPPORT MANAGER. Bimbo Bakeries USA, Inc. has openings for the position of Application Support Manager in Ft. Worth, TXto develop annual audit plan. Requires BS + 5 yrs exp. Apply & submit resume to: https://careers.bimbobakeriesusa.com /opportunities. Search for Job ID14989.

EXPEDIA, INC. currently has openings for the following opportunities in our Bellevue, WA office (various/levels/types): • Software Engineers: (728.SWE-AUG) Design, implement, and debug software for computers including algorithms and data structures. • Database Developers: (728.DBD-AUG) Coordinate changes to computer databases, test and implement the database applying knowledge of database management systems. • Reporting Analysts: (728.1557) Formulate and apply mathematical modeling and other optimizing methods to develop and interpret information that assists management with decision making. • Oracle Test Analysts: (728.1399) Create and execute test cases, and report and track test execution and defect metrics using test and defect management tools. • Managers, Engineering: (728.585) Responsible for architecture, design, construction, testing, and implementation of software. • Web Analysts: (728.1383) Formulate and apply mathematical modeling and other optimizing methods to develop and interpret information. • BI Developers: (728.1571) Participate in sourcing, organizing, maintaining, and standardizing large volumes of data through development of innovative tools, reporting dashboards, and well-organized. • Data Scientists: (728.1418) Apply advanced analytic techniques such as machine learning, data mining, and statistical modeling to design and implement mathematical models and algorithms to solve marketing problems. • Directors, Technology: (728.299) Build and manage a geographically distributed technology through subordinate managers and direct reports who support technology needs of core brand or function. 10% Travel to various unanticipated sites throughout the United States and internationally required.

EXPEDIA, INC. currently has openings for the following opportunities in our San Francisco, CA office (various/levels/types): • Software Engineers: (728.SWE-AUG-SF) Design, implement, and debug software for computers including algorithms and data structures. Send your resume to: Expedia Recruiting, 333 108th Avenue NE, Bellevue, WA 98004. Must reference position & Job ID# listed above.

HOTWIRE, INC. currently has openings for the following opportunities in our San Francisco, CA office (various/levels/types): • Software Engineers: (728.SWE-AUG-HOT) Design, implement, and debug software for computers including algorithms and data structures. • Business Analysts (Operations Research Analysts): (728.1603) Formulate and apply modeling and other optimizing methods to develop and interpret computer data sets. • Senior Release Engineers: (728.1700) Deploy and coordinate releases, patches/fixes/configuration changes, and updates to multiple Hotwire application environments. Send your resume to: Hotwire/Expedia Recruiting, 333 108th Avenue NE, Bellevue, WA 98004. Must reference position & Job ID# listed above.

SR. DATA ANALYST. Pattonair USA, Inc. has openings for the position of Sr. Data Analyst in Ft. Worth, TX to perform inventory based data set-ups assuring data integrity & periodic data validation. Requires Masters + 2 or Bach. + 5 yrs exp. Apply & submit resume to: http://www.pattonair.com/EN/Careers/Pages/default.aspx select "Vacancies" option then select "Americas".

WEBSPHERE ADMINISTRATOR - Anlze, dsgn, dvlp, implmnt, suprt & mantain s/w utilizing knowledge of Oracle, SQL Server, Java, J2EE, Web Sphere, Web Logic & Unix. Must have knowledge in WAS ND installation and Clustering. Must be willing to travel & reloc. Reqs MS in comp sci, sci, business Eng, or rel. Mail resumes to: HR, Parsetek Inc, 14117 Robert Paris CT Chantilly, VA, 20151.

PROGRAMMER ANALYST: Analyze, develop, implement, and test software applications utilizing knowledge of SQL, PL/SQL, Oracle DB, Oracle Applications(11i/R12) , reports & form builder (6i/10g), SDLC, OA framework, Oracle Workflow, SQL * Loader, GAP Analysis, FTP, Test Procedures, Cases, Scripts and Plans. Preparing the AIM documents. Must be willing to travel & relocation. Requires MS Comp Science, Engineering or related. Mail resumes to Nitya Software Solutions Inc. 9690 South 300, Ste 319, Salt Lake City, Utah 84070.

EVENTBRITE, INC. is looking for Principal Software Engineers in San Francisco, CA to develop iOS and Android applications. Resume to HR, Job# EB09, Eventbrite, Inc. 155 5th St Fl 7, San Francisco, CA 94103.

GLOBLZTN ARCHTCT (Austin, TX) Gthr & anlyze localization info. Crte detailed dsgns based on reqs & resolve dsgn flaws. Align cross func teams to sol archtctre. Cert high lvl & detailed dsgn docs crted by other archtcts & advise dvlpmnt grps on tech matters. REQ: 5 yrs exp in job &/or rel occup. Must have exp w/ Engng in archtctrl dsgn of glbl enterprise IT sol across mult systms; MS SharePoint Search or FAST Search; MS Bus Connectivity Svcs Connector Frmwwrk, 3rd pty plug-ins such as BAInsight, SharePoint Custom claim provider, & SharePoint Client Object Model; SharePoint content processing & query rules; SharePoint content mgmt sys; Full life IT dvlpmnt cycle; MS SharePoint; Prvdng tech guidance to project teams.: Send resume to: Althea Wilson, CA Technologies, One CA Plaza, Islandia, NY 11749, Refer to Requisition #104141.

JUNIPER NETWORKS

Juniper Networks is recruiting for our Sunnyvale, CA office:

Software Engineer #34550: Design, implement, test and maintain software features for Juniper's networking operating system (JUNOS).

Services Management Specialist Staff #32201: Act as a designated advocate for post-sale account support coordination. Work closely with the customer and multiple internal organizations to ensure effective implementation of Customer Support.

ASIC Engineer #28304: Work on verification of ASIC chip used for switches and routers. Perform test environment design, test planning and test development for block level and chip level logic units.

Technical Mktg. Spec. Sr. Staff #21315: Perform technical marketing related work for Juniper's switching product line focused on datacenter products and solutions.

Software Engineer #30529: Use routing knowledge in the management of routes across various next hop types and optimize jemalloc, kevent, and other FreeBSD libraries to enhance routing infrastructure for maximizing scale and performance of the various supported routing protocols.

Technical Support Engineer #35156: Provide in-depth diagnostics and root-cause analysis for network impacting issues on Juniper routing products (Internet backbone routers) to large Internet Service Provider and/or enterprise customers.

Test Engineer #31928: Develop test plans to verify functionality/Requests for Comments (RFCs) implemented for particular features of BGPM MPLS, and EVPN routing protocols. Execute test plans to identify any issues in implementation and ensure that the routing protocols adhere to RFCs or functional specs.

Hardware Engineer #33093: Contribute in a team oriented centralized Service Interface (SI) organization performing system signal integrity design with exposure to new and different cutting edge technologies. Perform analysis and design, and understand tradeoffs in designing interconnect solutions ranging from chip to chip, board to board, backplane and chassis to chassis interconnect.

Software Engineer #33959: Design software modules on large distributed (multi-core) platforms such as Linux/FreeBSD/netBSD in embedded environments. Implement network security software such as SSL proxy, web security, application security and application service components using network/system/cloud security technology including Firewall, UTM, IPS, ALG, AV, VPN, SSL, PKI, etc.

Software Engineer #33162: Gather system requirements from product managers, customers, development engineers, system testers, and other customer advocates. Translate requirements into high quality code for security data plane and maintain high quality software and associated artifacts.

Software Engineer #35185: Design and develop tools to establish a stable and efficient development environment, and release processes for scalable application deployment. Implement tools and processes required to establish an efficient and productive development environment.

Software Engineer #30239: Define routing requirements for all of company's products covering all routing protocols, including IGP, BGP, Multicast, and MPLS signaling protocols, and services including Layer 3 VPNs, Layer 2 VPNS, and VPLS.

ASIC Engineer #5029: Perform physical design implementation of ASICs, including block level and fullchip floor planning, placement and routing, timing closure and physical verification. Handle physical design implementation of blocks and partitions.

Software Engineer #29449: Develop web-based, high performance and large scale network application platforms and suites that maximize the value of enterprise and the service-provider customers' investment in network infrastructure. Perform business requirement analysis and high level software design.

Technical Support Engineer #18196: Provide Level-3 technical support on company campus and data-center products. Document all actions taken toward resolving customer reported issues on company products in contract tracking database.

Juniper Networks is recruiting for our Durham, NC office:

ASIC Engineer #35188: Create test specifications for ASIC design and sub-system verifications. Create test benches by using System Verilog and UVM language tools, and create tests to ensure correct ASIC RTL operations.

Mail single-sided resume with job code # to
Juniper Networks
Attn: MS 1.4.251
1133 Innovation Way
Sunnyvale, CA 94089

Apple Inc. has the following job opportunities in Cupertino, CA:

Software Engineer Applications (Req# 9GASY7) Des & dev SW sys to sup exist & new product features in cloud & big data comp space. Travel req. 20%.

ASIC Design Engineer (Req#9QE3KK). Respon for validation & characterization of analog IP used in silicon chips for mobile apps.

Development Engineer (Req#9FKMNB). Detect and respond to information security threats against Apple.

Software Engineer Applications (Req# 9M4U5P). Des & dev OSX/iOS apps & frameworks for Identity Management Systems.

Software Engineer Applications (Req# 9SV6BQ). Des & dev software for retail point-of-sale systems.

Software Engineer Applications (Req# 9TP4QG) Resp for dsgn, architect, & implmnt of Apple's Contact Center operation and mgmt tools solutions.

Software Engineer Applications (Req# 9NRTNR). Supp & maintain custom developed frameworks & infrastructure.

Software Engineer Systems (Req# 9F4VZ5). Design & develop test & calib infrastructure for various Biometric Sensor.

Software Development Engineer (Req# 9F4SKJ) Des & dvlp Cellular SW features.

Software Engineer, Applications (REQ#9LA364). Des, dev & sup softw for gift card sys'ms .

Software Engineer Systems (REQ #9MHVD9). Respon for design, develop & support of iCloud Report enterprise web app.

Software Development Engineer (REQ# 9JANWR). Ensure 24/7 availb of Siri Hadoop based backend.

Software Development Engineer (REQ#9QCQCH). Perfm end-to-end dev of web app proj's for multi Operations groups.

Software Engineer Applications (REQ# 9JPUG8). Research, des, dev, impl't, & debug big data reporting platform.

Firmware Engineer (REQ#9VJSXN) Des & dvlp Firmware/SW for embedded accessories.

Systems Design Engineer (REQ# 9EZ28B). Evaluate the latest iPad, iPhone and iPod HW systems. Travel req'd 30%.

Software Engineer Applications (REQ# 9F7TNG) Des & dvlp rich web apps.

Systems Design Engineer (Req#9F-DW2H) Des & dvlp Over-the-Air (OTA) measrmnt sys.

IS&T Technical Project Lead (REQ#9NMSLP). Maintain portfolio of Busi Intel tools that enable SDE business team for reporting.

Software Development Engineer (REQ#9Q7KYV). Dev & write test plans for Maps services' features.

Software Quality Assurance Engineer (REQ#9P92AU). Provide end-to-end test automation solutions for key pltfrms & apps.

Hardware Development Engineer (REQ#9G9TBL) Develop hardware and verify Mac/iOS systems.

ASIC Design Engineer (Req#9E5UH2) Read specs of HW blocks & write the test plan to test the design.

Software Engineer Applications (REQ#9RDTGU) Prep functional spec during proj req's stage & conceptualize optimal & scalable sol's.

Sr. Software Engineer (REQ#9RF28L) Design and develop enterprise business applications.

Software Engineer Applications (REQ#9LKTV5) Des & dev web based soltn's for customer facing, online support apps.

Software Engineer Applications (REQ#9KT2LY). Des, dev, & deply SW apps for data-warehsng & busin intel projs in Apple IST Mktg Dept.

Hardware Development Engineer (REQ#9R34UN). Dev & qual adv lith ion batt tech for prtble pwr cons apps. Travel reqd: 20%.

Senior Software Engineer (REQ# 9SKVA6) Des & dev SW for cust sys group in service mngmnt domain.

Test Engineer (REQ#9D2VJF) Ops lead RF test engineer resp for planning & execution of New Product Intros to worldwide mfg sites. Travel req. 30%.

Engineering Project Lead (REQ# 9TG32D) Dev, create, implmnt, & support the web app dev of Sales Training App using large-scale & high-perf objct-oriented internet techs.

Software Engineer Applications (REQ#9KPQWT) Dsgn, dev, & deploy data warehouse & analytics solutions for multiple bus. groups at Apple.

Software Development Engineer (REQ#9U33P6) Conduct wide variety of TTS activities, incl linguist analysis, stat modlng, signal prcssng, & voice dev.

Engineering Project Lead (Req# 9JXUFM) Lead end-to-end support exp worldwide for 1+ feature-sets w/ in iCloud, Apple ID, iMessage, FaceTime, or Game Center.

Software Development Engineer (Req#9NM33C). Resp for the bring-up & devlpmnt of next gen Macintosh platforms.

Mechanical Design Engineer (REQ#9EHN9N). Set priorities & determine risk mitgtn plans using an undrstndng of tech details of cross-functional issues & risks in manfctrng. Travel req'd 30%.

Software Engineer Applications (REQ#9EYW47). Design and develop PKI and cryptographic services.

Senior Software Engineer (Req# 9P346W). Design, build & test server-driven mobile e-commerce app sw on the Apple iOS platform.

Hardware Development Manager (Req# 9EYU6S) Lead & grow a team resp for dvlping robust validation solutions & automation infrstructre for touch prdcts. Travel req'd 20%.

Mechanical Design Engineer (Req#9B-N3BB). Supp prod dvlpmnt by driving the dvlpmnt of 1 of the key process

dvlpmnt focus areas across iPod and iPhone. Travel req'd 25%.

Systems Design Engineer (REQ# 9D9VWS). Dev object-oriented SW used in the manufacturing processes of Apple products. Travel req'd 25%.

ASIC Design Engineer (REQ# 9DLTBA). Responsible for physical design and implementation of partitions.

Hardware Development Engineer (Req#9GQ4MC). Research, design, develop & launch next-gen Sensing Technologies. Travel req 20%.

Software Development Engineer (REQ#9PFR9T). Design & implement device drivers for peripheral devices across all iOS HW platforms.

Software Development Engineer (REQ#9EP3PA). Dsgn, implmnt, & maintain compiler support for debugging app & sys SW.

Software Development Engineer (REQ#9U55BR). Dsgn & dev low-level pwr mgmt SW.

Software Engineer Systems (REQ# 9D5QU5). Rsrch, dsgn, dev, & debug OS kernels, device drvrs, framewrks, & libraries on embedded systs.

Software Development Engineer (REQ#9RGVGH) Dvlp & dbg boot FW & HW fr nxt gen. Mac pdcts.

Hardware Development Engineer (REQ#9FV4TT). Respon for design, develop & validation of radio baseband hw circuits & sys for wireless comm dvcs.

Engineering Manager (REQ#9DCTDF). Design mech & elec interconn comp. & prod. for iPhone and iOS devices. Travel required 15%.

Software Engineering Technical Writer (REQ#9N8UQR). Create instructional documentation to develop software platforms.

Software Development Engineer (REQ#9L5M8T). Dsgn & dvlp OS lvl netwrkng SW acrss range of prdcts.

Software Engineer Applications (REQ#9E5VLE). Research & dev large-scale Cloud-based productivity app suite.

Virtual Machine Engineer (REQ# 9DPNZF) Dev & maintain all aspects of JavaScript Core JavaScript Virtual Machine, including the interpreter, just-in-time compiler, garbage collector, & runtime library.

Software Engineer Applications (REQ#9NVTE7). Des & dev SW sol's for world wide POS systems.

Software Development Engineer (REQ#9H3TRU). Design and optimize the Swift compiler.

Software Engineer Applications (REQ#9MFUUZ). Des, dev, implmnt, maintn & opr lg scale distrib'd sys.

Information Systems Engineer (REQ# 9QE3HD). Wrk w/key global & regional biz users, BPRs & ext'd IS&T teams to drive & implement strategic SCM sol's @ Apple.

Software Engineer Systems (REQ# 9SL38V). Dsgn, dev, & support highly available & high-performance entrprse Hadoop solutions & admin of Hadoop clusters for iCloud Reprtng.

Software Development Engineer (REQ#9UFPPB). Des, implem, & sup tools & compon's for local data qual monitoring & verification.

Software Engineer, Systems (REQ# 9S4QDH). Des dev & sup highly avail & high perform enterprise Hadoop sol's & admin of Hadoop clusters for iCloud Reporting.

Software Development Engineer (REQ#9D2NDG). Perform SW eng'g specific to translation & localization of Siri & other cloud-based SW user interfaces into foreign languages.

Software Development Engineer (REQ#9A639A) Bld, anlyze, & test SW to imprve prfrmnce of Siri srvce.

Firmware Engineer (REQ#9LUTHN) Dsgn & dvlp firmware for wrlss audio prdts.

Software Quality Assurance Engineer (REQ#9PUMNS) Des crte & exec't test suites & test cases to qualify Maps Core services

Hardware Development Manager (REQ#9E52CA). Des & dev display panels & integrate Apple's prod's.

Operations Engineer (Manufacturing) (REQ#9E8QUP). Des & dev fixtures for electronics assembly & measurement. Travel req'd 40%.

Software Engineer Applications (REQ#9XJVVW). Des, implmnt, & debug specific modules of a SW sys to allow users to interact w/large datasets on a server via a variety of client apps running on MacOS, iOS & other platforms.

Software Development Engineer (REQ#9D2SVD) Dsgn & dvlp low lvl SW for cntrlling pwr mgmt. features of new chipsets & platforms.

Apple Inc. has the following job opportunity in Austin, TX:

Systems Engineer (REQ#9H7T5L) Build & tblsht comp serve sys.

Apple Inc. has the following job opportunity in San Francisco, CA:

Software Development Engineer (REQ#9E2PBJ). Create SW test plans & exec black/white box fnctnl & perf tests.

Refer to Req# & mail resume to Apple Inc., ATTN: L.M. 1 Infinite Loop 104-1GM Cupertino, CA 95014.

Apple is an EOE/AA m/f/ disability/vets.

ADVERTISER INFORMATION • AUGUST 2015

Advertising Personnel

Debbie Sims: Advertising Coordinator
Email: dsims@computer.org
Phone: +1 714 816 2138 | Fax: +1 714 821 4010

Chris Ruoff: Senior Sales Manager
Email: cruoff@computer.org
Phone: +1 714 816 2168 | Fax: +1 714 821 4010

Advertising Sales Representatives (display)

Central, Northwest, Far East:
Eric Kincaid
Email: e.kincaid@computer.org
Phone: +1 214 673 3742
Fax: +1 888 886 8599

Northeast, Midwest, Europe, Middle East:
David Schissler
Email: d.schissler@computer.org
Phone: +1 508 394 4026
Fax: +1 508 394 1707

Southwest, California:
Mike Hughes
Email: mikehughes@computer.org
Phone: +1 805 529 6790

Southeast:
Heather Buonadies
Email: h.buonadies@computer.org
Phone: +1 201 887 1703

Advertising Sales Representatives (Classifieds & Jobs Board)

Heather Buonadies
Email: h.buonadies@computer.org
Phone: +1 201 887 1703

Hitachi Consulting Corp.

The following openings are for Dallas, TX & various unanticipated worksites throughout the U.S. & require up to 100% domestic travel to various & unanticipated client/project worksites: Program/Project Manager/Req#29592 work w/ business & technical teams to drive critical business initiatives & deliver large enterprise projects in retail commerce & e-commerce domain; **Software Engineer/Req#29594** design & implement end-to-end solution in consumer goods/life sciences industries; **Functional Lead-SCM Mgmt/Req#29595** provide functional/technical expertise related to solution offerings to support deliverables and/or project objectives for Oracle Applications SCM domain; **Software Engineer/Req#29599** design & implement end-to-end solution in ERP solution architecture; **Service (Solution) Architect/Req#29604** execute Oracle implementation, testing & support projects & consult re: best Oracle supply chain option; **Service Architect/Req#29606** provide functional/technical expertise related to solution offerings to support deliverables and/or project objectives & develop, design & implement enterprise level, large scale, scalable apps using .NET C# & ASP.Net; **Architect/Req#29608** analyze, design & gather business reqs & develop e-commerce solutions using WebSphere Commerce Serve & Oracle; **Senior Consultant/Req#29610** solution design & perform SAP ABAP development & implementations; **Database Administrator/Req#29611** applications DBA for Oracle EBS 11i & R12; **Software Engineer/Req#29613** design & implement end-to-end solution in ERP solution architecture; **Software Engineer/Req#29596** design & implement end-to-end solution in ERP solution architecture; **Software Engineer/Req#29605** design, develop, test & deploy end-to-end solutions for implementation, upgrade & support engagements in relation to Oracle ERP solution architecture; **SharePoint Architect/Req#29607** design, develop & implement scalable & extendable customized SharePoint enterprise solutions; **Programmer Analyst/Req#29597** design & conduct changes in application systems related to apps programming in Oracle SOA suite; **Programmer Analyst/Req#29598** application programming in Oracle Financials; **Software Engineer/Req#29600** design of end-to-end technical solutions & software development/engineering; **Software Engineer/Req#29601** design & implement end-to-end solution in ERP solution architecture & Oracle EBS; **Software Engineer/Req#29602** design & implement end-to-end solution in ERP solution architecture; **Software Engineer/Req#29612** design & implement end-to-end solution using Microsoft .Net development; **Software Engineer/Req#29615** design & implement end-to-end solution involving custom development solutions using Microsoft technologies; **Project Manager/Req#29616** implement Agile (scrum) development methodology & MS web application design & development; **Senior Consultant/Req#29617** solution architect for web apps using MS based technologies; **Software Engineer/Req#29618** design end-to-end technical solution in ERP solution architecture; **Programmer Analyst/Req#29621** involved in analysis, design & build phases of projects in relation to Oracle Fusion, Oracle Apps R12 & 11i & Oracle Financials; **Software Engineer/Req#29619** design of end-to-end technical solutions & develop, test & deploy solutions for implementation, upgrade & support engagements involving ERP solution architecture; **Manager, Software Engineering/Req#29620** leading cross functional teams in global delivery model or NShore model & managing enterprise-level large scale projects/products; **Senior Consultant/Req#29622** analyze & gather customer's business reqs & define business data reqs & create reports to integrate w/ the SAP non/SAP data; **Senior Consultant/Req#29623** senior system analyst for ERP-SAP implementation; **Software Engineer/Req#29624** design & implement end-to-end solution w/ functional & technical component w/in the MS technology stack; **Software Engineer/Req#29625** design & implement end-to-end solutions w/ technical deployment involving ERP Solution Architecture; **Senior Consultant/Req#29626** analysis, architecture, design & development of data warehouse & BI solutions & MS BI full life cycle implementations; **Business Analyst/Req#29627** assess the data quality for patterns, value ranges, record completeness & conformity to data standards using various profiling strategies & facilitate workshops for understanding data flows & lifecycle, data owners & data quality mgmt. processes to define data governance methods & procedures; **Senior Consultant/Req#29628** develop business solutions utilizing MS SharePoint & MS SQL Server & BI tools; **Senior Consultant/Req#29629** analyze & gather customer's business reqs & develop & review functional & technical specifications & interfaces using MS technologies; **Business Analyst/Req#29630** implement Oracle ERP systems & analyze customer business process & design the solution in Oracle ERP environment; **Manager/Req#29631** develop strategies, drive execution & planning of the support mechanism for roll out of e-commerce apps; **Software Engineer/Req#29633** analyze specification & design of end-to-end technical solutions in SAP; **Software Engineer/Req#29632** design & implement end-to-end solution & Enterprise Architecture integration.

The following opening is for Chicago, IL:

Software Engineer/Req#29614 design of end-to-end technical solutions including install & upgrade of Oracle EBS from 11i to R12 & requires up to 75% domestic travel.

Interested applicants mail resumes to Attn: Reuben Job, Hitachi Consulting, 7979 Gateway Boulevard, Suite 220 Newark, CA 94560 & refer to Job/Req# of interest.

CLASSIFIED LINE AD SUBMISSION DETAILS: Rates are $425.00 per column inch ($640 minimum). Eight lines per column inch and average five typeset words per line. Send copy at least one month prior to publication date to: Debbie Sims, Classified Advertising, *Computer* Magazine, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720; (714) 816-2138; fax (714) 821-4010. Email: dsims@computer.org.

In order to conform to the Age Discrimination in Employment Act and to discourage age discrimination, *Computer* may reject any advertisement containing any of these phrases or similar ones: "...recent college grads...," "...1–4 years maximum experience...," "...up to 5 years experience," or "...10 years maximum experience." *Computer* reserves the right to append to any advertisement without specific notice to the advertiser. Experience ranges are suggested minimum requirements, not maximums. *Computer* assumes that since advertisers have been notified of this policy in advance, they agree that any experience requirements, whether stated as ranges or otherwise, will be construed by the reader as minimum requirements only. *Computer* encourages employers to offer salaries that are competitive, but occasionally a salary may be offered that is significantly below currently acceptable levels. In such cases the reader may wish to inquire of the employer whether extenuating circumstances apply.

Help build the next generation of systems behind Facebook's products.

Facebook, Inc.

currently has openings in
Menlo Park, CA (various levels/types):

Data Engineer, Analytics (4219J)
Responsible for data warehouse plans for a product or a group of products. Design, build, and launch new data models in production and new data extraction, transformation and loading processes in production.

Data Engineer (4566J)
Design, develop, test and launch new reports and dashboards into production, and provide support to reports and dashboards running in production.

Data Scientist (830J)
Apply your expertise in quantitative analysis, data mining, and the presentation of data to see beyond the numbers and understand how our users interact with our core products.

Mail resume to: Facebook, Inc. Attn: SB-GIM, 1 Hacker Way, Menlo Park, CA 94025. Must reference job title and job# shown above, when applying.

Ooyala Inc. has openings in our **Santa Clara, CA** location for:

Software Engineer (6688.33) Build interactive data visualizations using the latest web technologies, including D3 & CoffeeScript, working with terabyte-scale datasets; **Associate Product Specialist (6688.75)** Design & build a new tool (billing & tooling) that will optimize packages for hundreds of customers worldwide; **Associate Product Specialist (6688.52)** Design & build a new tool (billing & tooling) that will optimize packages for hundreds of customers worldwide, further defining Ooyala as a trusted partner;

Ooyala Inc. has openings in our **Plano, TX** location for:

QA Manager (6688.80) Build & lead a team of QA Engineers & Software Developers in Test responsible for delivering world class products using agile practices in a SaaS environment & at web scale; **Backend Developer (6688.76)** Review requirements & translate them to high level design.

Send resume to HR, 4750 Patrick Henry Drive, Santa Clara, CA 95054.
Must ref. job code above when applying.

Linkedin Corp.

Linkedin Corp. has openings in our **Mtn View, CA** location for: **Software Engineer (All Levels/Types) (6597.962, 6597.1032, 6597.959, 6597.1254, 6597.1332, 6597.901, 6597.286, 6597.662, 6597.791, 6597.1356, 6597.582, 6597.978, 6597.1020)** Design, develop & integrate cutting-edge software technologies; **Senior Manager, Software Engineering (6597.128)** Create & own the strategic roadmap for the team; **Engineering Manager (6597.1194)** Lead a team of Software Engineers in conducting architecture, design, & implementation work; **Associate Web Developer (6597.792, 6597.646)** Own the front-end development for one or more products; **Manager, Software Engineering (6597.553)** Work across the entire stack from the data layer to the distributed service layer & the front-end presentation tier to optimize the user experience on mobile devices; **Sr. Information Security Engineer (6597.579)** Protect infrastructure, applications, & members by identifying new vulnerabilities & responding to existing vulnerabilities within the organization; **Senior Product Manager (6597.1012)** Conduct analysis of the competitive environment, customers, & product metrics to determine the right feature set to drive engagement & usage on LinkedIn; **Information Security Engineer (6597.941)** Responsible for identifying new vulnerabilities & responding to existing vulnerabilities within the organization.

Linkedin Corp. has openings in our **Sunnyvale, CA** location for: **Software Engineer (All Levels/Types) (6597.928, 6597.770, 6597.683, 6597.1306)** Design, develop & integrate cutting-edge software technologies; **Test Engineer (6597.929)** Design & develop automated test suites, continuously design creative ways to break software, & identify potential bugs; **Senior Database Engineer (6597.596)** Evaluate & implement of new functionality, performance tuning, root-cause analysis, & issue resolution across complex database environments; **Manager, Software Engineering (6597.567)** Create & deliver compelling products & end user experiences in the search & analytics domain with focus on building software platforms; **Manager, Database Engineering (6597.75)** Administer Oracle database systems & related environments from development to production; **Systems Engineer (6597.854)** Improve Operations Teams' efficiency by developing, identifying, & deploying tools to improve the manageability & support ability of the production, staging, reporting, & corporate environments; **Manager, Systems Engineering (6597.288)** Lead a team of up to 6 Solutions Architects & Developers to design, develop & integrate cutting-edge software technologies; **Associate Web Developer (6597.992)** Own the front-end development for one or more products.

Linkedin Corp. has openings in our **San Francisco, CA** location for: **Software Engineer (All Levels/Types) (6597.1075, 6597.818)** Design, develop & integrate cutting-edge software technologies; **Technical Consultant (6597.953)** Analyze clients' business needs & recommend the appropriate technology solutions; **Technical Services Manager (6597.915)** Oversee the technical support of company customers; **Quality Assurance Engineer (6597.1347)** Work with agile team members in estimating test effort using SCRUM & Kanban methodologies. Position allows for telecommuting; **Technical Solutions Architect, Sales Systems (6597.839)** Responsible for the analysis, design, & development of sales systems projects in conjunction with other members of Sales Systems teams. Limited travel required to other company worksites in the Bay Area.

Linkedin Corp. has openings in our **Calabasas, CA** location for: **Enterprise Data Warehouse Architect (6597.1339)** Build a leading edge enterprise data warehouse encompassing the entire life cycle, including data integration, transformation, logical & physical design, security, backup, & archival strategies implementing industry best practices. **Software Engineer (All Levels/Types) (6597.1342)** Design, develop & integrate cutting-edge software technologies.

Please email resume to: 6597@linkedin.com. Must ref. job code above when applying.

2016

Richard E. Merwin Distinguished Service Award

CALL FOR AWARD NOMINATIONS
Deadline 15 October 2015

ABOUT THE MERWIN AWARD
The highest level volunteer service award of the IEEE Computer Society for outstanding service to the profession at large, including significant service to the IEEE Computer Society or its predecessor organizations.

ABOUT DICK MERWIN
Dick Merwin was a pioneer in digital computer engineering who participated in the development of the ENIAC, MANIAC, and STRETCH computers. Despite a busy and productive technical career, Merwin found time to be active in professional societies, including the IEEE Computer Society, ACM and AFIPS. His generosity of spirit and genuine helpfulness was an important element in the progress of the computer profession.

AWARD
A bronze medal and $5,000 honorarium are awarded.

PRESENTATION
The Richard E. Merwin Award is presented at the IEEE Computer Society's Annual Awards Ceremony.

REQUIREMENTS
This award requires 3 endorsements.

NOMINATION SUBMISSION
Nominations are being accepted electronically
www.computer.org/web/awards/merwin

RANGACHAR KASTURI
2015 Richard E. Merwin Distinguished Service Award

For exemplifying a true volunteer spirit and a commitment to excellence through significant and continuing contributions supporting the vision and mission of the IEEE and the IEEE Computer Society.

AWARDS HOMEPAGE
www.computer.org/awards

CONTACT US
awards@computer.org

Move Your Career Forward

IEEE Computer Society Membership

Explore All of the Related Resources on Software

Build Your Knowledge

IEEE Software

The authority on translating software theory into practice, this bimonthly magazine presents pioneering ideas, expert analyses, and thoughtful insights to help software professionals keep up with rapid technology change.

IEEE Transactions on Software Engineering

This bimonthly journal focuses on well-defined theoretical results and empirical studies with real potential to impact software construction, analysis, and management.

Create Connections

Technical Council on Software Engineering

TCSE encourages the application of engineering methods and principles in developing software, and works to increase professional knowledge of techniques, tools, and empirical data to improve software quality. The council cosponsors conferences and develops proposals for IEEE software engineering standards.

Advance Your Career

Software Engineering Knowledge Area Certificates

The 12 certificate areas assess a candidate's proficiency in understanding the concepts involved, as well as their ability to apply them in the software engineering process.
http://www.computer.org/web/education/knowledge-area-certificates

FOR DIRECT LINKS TO THESE RESOURCES, VISIT

www.computer.org/edge-aug

IEEE computer society

The Community for Technology Leaders

30th IEEE INTERNATIONAL Parallel and Distributed Processing SYMPOSIUM

MAY 23-27, 2016
CHICAGO HYATT REGENCY
CHICAGO, ILLINOIS USA

IPDPS 2016 CHICAGO

IPDPS 2016 will be held in Chicago, the third largest metropolis in the USA. It boasts a dual-hub airport system and 46 non-stop global flights daily. Situated along the shoreline of Lake Michigan, downtown Chicago (aka the Loop), where the Hyatt Regency is located, is an easy walk or cab ride to the attractions the city offers, including parks, beaches, shopping, museums, live entertainment, and the singular international cuisine of Chicago's diverse ethnic neighborhood eateries. Check the IPDPS Web pages for updates and information on workshops, the PhD Forum, and other events as the full program develops.

CALL FOR PAPERS

Authors are invited to submit manuscripts that present original unpublished research in all areas of parallel and distributed processing, including the development of experimental or commercial systems. Work focusing on emerging technologies is especially welcome. Topics of interest include:

- **Parallel and distributed algorithms**, focusing on topics such as: numerical, combinatorial, and data-intensive parallel algorithms, locality-aware and power-aware parallel algorithms, streaming algorithms, parallel algorithms in specific domains such as machine learning and network science, scalability of algorithms and data structures for parallel and distributed systems, communication and synchronization protocols, network algorithms, scheduling, and load balancing.

- **Applications of parallel and distributed computing**, including computational and data-enabled science and engineering, big data applications, parallel crowd sourcing, large-scale social network analysis, management of big data, cloud and grid computing, scientific, biological and medical applications, and mobile computing. Papers focusing on applications using novel commercial or research architectures, big data approaches, or discussing scalability toward the exascale level are encouraged.

- **Parallel and distributed architectures**, including architectures for instruction-level and thread-level parallelism; petascale and exascale systems designs; novel big data architectures; special purpose architectures, including graphics processors, signal processors, network processors, media accelerators, and other special purpose processors and accelerators; impact of technology on architecture; network and interconnect architectures; parallel I/O and storage systems; architecture of the memory hierarchy; power-efficient and green computing architectures; dependable architectures; and performance modeling and evaluation.

- **Parallel and distributed software**, including parallel and multicore programming languages and compilers, runtime systems, operating systems, resource management including green computing, middleware for grids, clouds, and data centers, libraries, performance modeling and evaluation, parallel programming paradigms, and programming environments and tools. Papers focusing on novel software systems for big data and exascale systems are encouraged.

WHAT/WHERE TO SUBMIT

Authors will need to register their paper and submit an abstract by October 9, 2015 and then submit full versions by October 16, 2015. More details on submissions and instructions for submitting files are available at www.ipdps.org. All submitted manuscripts will be reviewed. Submitted papers should NOT have appeared in or be under consideration for another conference, workshop or journal.

GENERAL CHAIR

Xian-He Sun
(Illinois Institute of Technology, USA)

PROGRAM CHAIR

Jeffrey K. Hollingsworth
(University of Maryland, USA)

PROGRAM VICE-CHAIRS

- **Algorithms:**
  Ümit V. Çatalyürek
  (Ohio State University, USA)
- **Applications:**
  Darren Kerbyson
  (Pacific Northwest National Laboratory, USA)
- **Architecture:**
  Andrew A. Chien
  (University of Chicago, USA) &
  Hank Hoffman
  (University of Chicago, USA)
- **Software:**
  Karen Karavanic
  (Portland State University, USA)

- KEYNOTES & TECHNICAL SESSIONS
- WORKSHOPS & PHD FORUM
- COMMERCIAL PARTICIPATION

Details at www.ipdps.org

IMPORTANT DATES

August 15, 2015
Proposals for New Workshops

October 9, 2015
Submit Abstract

October 16, 2015
Submit Paper

December 18, 2015
Author Notification

After December 18, 2015
Deadlines for Paper Submissions to Workshops

SPONSORED BY:

IEEE computer society

IN ASSOCIATION WITH:

- ACM SIGARCH
- IEEE Computer Society Technical Committee on Computer Architecture
- IEEE Computer Society Technical Committee on Distributed Processing