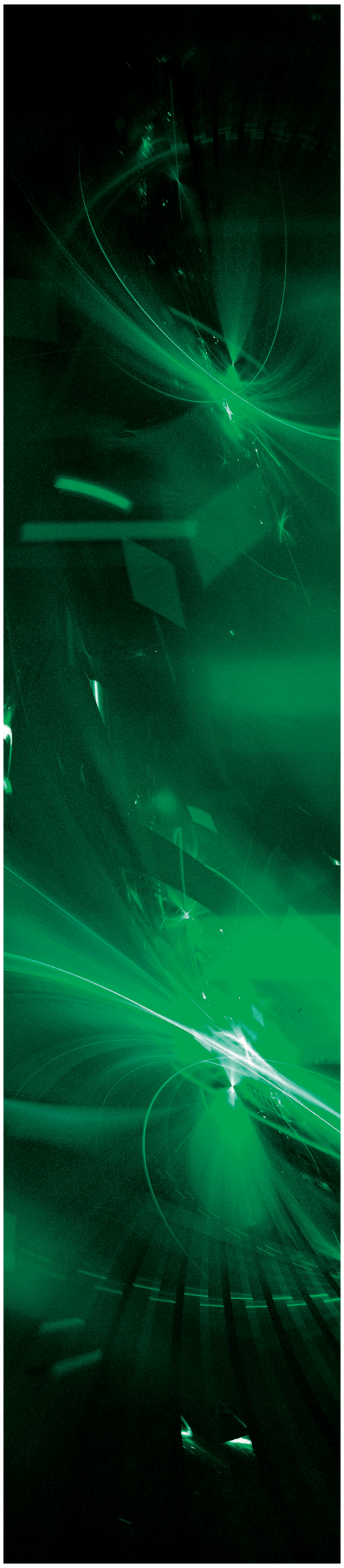


The Big Bang: 25 Years of Software History

"Big Bang" is a popular term: chroniclers of almost anything like to use it to describe Day One. The particulars of the event, however, often don't live up to the implications of the term.

But for the purpose of placing the past quarter-century of software history into perspective, let's be bold. Within a few months on either side of *IEEE Software*'s January 1984 debut, what we might arguably call the modern era of software, computing, and networking (and perhaps even the modern global economy) came to fruition almost at once. For instance, in 1983, Bell Labs researcher Bjarne Stroustrup contributed C++ to the programming lexicon, and the Domain Name System was invented.

This time line is our admittedly incomplete and imperfect snapshot of selected events representing the advance of software production, engineering, and theory—just enough to provide some context for the times recent and bygone. If you have any comments (omitted milestones, corrections), write to us at software@computer.org.

IEEE Software's past editors in chief have added a few comments to provide some context for the time period of their terms as EIC. We've also asked some of the discipline's leading practitioners what they think most profoundly influenced the science and practice of software development, and what influenced the influencers.

1984

- Fred Cohen, creator of the first virus, publishes *Computer Virus—Theory and Experiments.*
- The US Defense Department awards Carnegie Mellon University the contract to establish a Software Engineering Institute.
- Richard Stallman leaves MIT and begins the GNU Project.
- Apple unveils the first Macintosh.

1985

- Bjarne Stroustrup releases the first commercial implementation of C++.
- Aldus develops PageMaker for the Mac, enabling widespread desktop publishing.
- Microsoft releases the first Windows operating system, signaling it was serious about computing for the masses.

1986

- Microsoft goes public.
- The SCSI (Small Computer Systems Interface) specification is accepted as an ANSI standard.
- Fred Brooks publishes *No Silver Bullet.*
- The first OOPSLA conference is held.
- The *Wall Street Journal* article helps popularize the concept and term "CASE" for computer-aided software engineering.

1987

- *Peopleware* by Tom de Marco and Timothy Lister is published.
- Larry Wall introduces Perl.

1988

- Grad student Robert Tappan Morris writes code to gauge the Internet's size; it becomes known as the first Internet worm.
- Barry Boehm publishes "A Spiral Model of Software Development and Enhancement" in *Computer.*

Software Leaders Cast Their Votes

Greg Goth

The time line we present in this issue cannot claim to be complete. To help round out the picture, we decided to ask some of software's most accomplished practitioners about the things they thought transformed the discipline, the industry, and the world between 1984 and today. We set one ground rule—they had to propose someone else's work rather than their own. Herewith, their observations.

KENT BECK

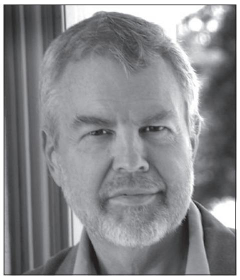
Founder of Extreme Programming and author of *Extreme Programming Explained;* kentb@earthlink.net

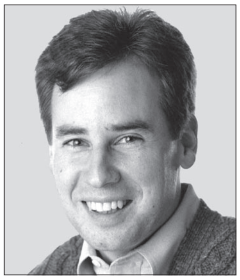
Programmers are people, 1987. I wanted to nominate the publication of *Object-Oriented Software Engineering: A Use-Case Driven Approach* by Ivar Jacobsen, but it was published in 1982. It rocked my world by suggesting I view systems first through the users' eyes. Focusing on 1984 on, I nominate *Peopleware* by Tom DeMarco and Tim Lister. The idea that programmers are people was shocking to me. I've never thought about software development the same since reading it. There are many clever software engineering ideas that would work wonderfully, if only programmers were computers instead of people. Alas ….

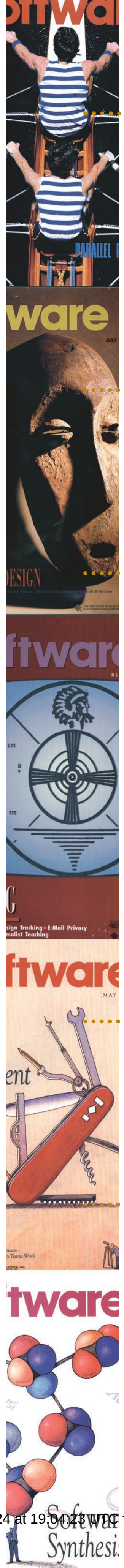
continued on p. 9 …

Reflections from *Software's* Past Editors in Chief

Here, at our request, all the former editors in chief think back to their goals during their terms and what they thought were the most profound events, occurrences, and advances in software during that time. Some of their recollections are on pp. 10–14. —ed.







1989

- Tim Berners-Lee writes *Information Management: A Proposal,* the foundational document of the World Wide Web, and solicits comments at CERN.
- Watts Humphrey writes *Managing the Software Process.*

1990

- Alan Emtage develops the first Internet search engine, Archie, at McGill University.

1991

- The Object Management Group releases Corba 1.0.
- Linus Torvalds modestly announces his new project, a free operating system, which becomes Linux, and initiates the "Bazaar" style of software development.
- Guido van Rossum releases Python.
- The SEI publishes version 1.0 of the CMM for Software (SW-CMM).
- Tim Berners-Lee posts the World Wide Web on the alt.hypertext newsgroup.

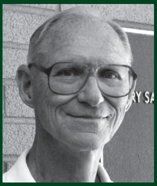
1992

- The US amends the National Science Foundation Act to allow commercial use of the Internet.

1993

- The Hillside Group discusses expanding the concept of patterns in software design.
- The National Center for Supercomputing Applications, at the University of Illinois, releases Mosaic 1.0.

continued from p. 7 ...



BARRY BOEHM

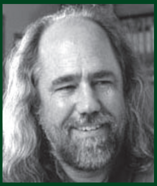
TRW Professor of Software Engineering and director of the Center for Systems and Software Engineering, University of Southern California; boehm@usc.edu

Software Capability Maturity Model, 1987. When the SW-CMM emerged as a Software Engineering Institute technical report in 1987, most of its recommended processes and practices had been well known ever since the 1956 and 1961 publication of key papers on the SAGE [Semi-Automatic Ground Environment] air-defense system. No rocket science was involved; it was simply a "just do it" checklist. But for decades, similar "just do it" advice was largely ignored via excuses such as "we're different," "it's too expensive," and "we don't have time."

But the SW-CMM got many organizations to "just do it," owing to strong packaging and social-engineering practices. Its major packaging strength was to organize the key process areas into maturity levels. One of its social-engineering strengths was to establish a given maturity level (initially Level 2, later Level 3) as a necessary qualification for bidding on software contracts. A second strength was in creating a funded community of practice by requiring Level 3 organizations to fund corporate Software Engineering Process Groups. Not every "just-do-it" had positive effects (described later), but overall the SW-CMM significantly reduced the level of sloppy software engineering practice worldwide.

Architectural mismatch, 1995. One of the more seductive recent software composition images is that of the Web 2.0 mashup, in which new Web applications are created by mashing together various existing Web apps. This often works when the Web applications share compatible assumptions about the nature of control, data, and users, but often doesn't work when they don't. David Garlan highlighted this phenomenon of architectural mismatch in a 1995 *IEEE Software* article, which describes how his team tried to mash together four components with incompatible architectural assumptions, turning an expected six-month, one-person-year project into a two-year, five-person-year disappointment.

Separation of concerns. Another seductive software development approach involves the principle of *separation of concerns*, expressed in the SW-CMM by the statement "Analysis and allocation of the system requirements is not the responsibility of the software engineering group but is a prerequisite for their work." A 2006 *Systems Engineering* article by Mark Maier titled "System and Software Architecture Reconciliation" provides good examples of the trouble that this approach can impose on software developers. The article deals with the fundamental architectural mismatches between the physical architectures developed by hardware systems engineers and good software architectures: functional "part-of" relationships versus layered "served-by" relationships; monolithic data organization; interface data flows versus protocols; and static versus dynamic functional-physical allocation.



GRADY BOOCH

IBM Fellow and author of *Object-Oriented Analysis and Design with Applications* and the *UML Users Guide*; architecture@booch.com

David Parnas' "On the Criteria to Be Used in Decomposing Systems into Modules," 1972. (We decided to accept Grady's suggestion even though it falls outside our ground rules. —ed.) The entire history of software engineering can be characterized by rising levels of abstraction. We see this in our languages, our tools, our methods. Such abstraction is essential, for it's the primary means whereby we as humans attack the problems of complexity. Today, the typical contemporary software-intensive system is continuously evolving, distributed, concurrent—and very complex. David's work in the early 1970s represented a state change in the way we attack complexity, namely, by information hiding and abstraction. David's work, concurrent with work in abstract data types by Mary Shaw and Joseph Goguen, represented a seminal contribution to the software development method, leading the way for the now-contemporary generation of object-oriented languages and methods, which are all mainstream.

MICHAEL CUSUMANO

SMR Distinguished Professor of Management & Engineering Systems, Massachusetts Institute of Technology, and author of *The Business of Software*; cusumano@mit.edu

Microsoft's iterative approach, 1989. In 1989, Microsoft adopted a more structured iterative approach centering around daily builds, immediate bug fixes, and short milestone releases. These practices, initially in the Excel 3 project headed by Chris Peters, are now common in software engineering around the world. They're especially important for accommodating late design changes and responding better than older waterfall-ish methods to the requirements of fast-paced markets and rapidly evolving technologies. If Microsoft hadn't mastered these techniques, it never would have shipped Windows 3.1 in 1992 or blockbuster graphical applications such as Office in the early 1990s, and it probably would have collapsed as a company.

continued on p. 11 ...

Reflections

Bruce Shriver

President of Genesis 2 and EIC 1984–1987

As the inaugural EIC of *IEEE Software*, I faced a unique challenge: to provide content that would be an interesting, timely, and relevant blend of practical and research articles—in short, to develop a differentiating identity for *IEEE Software*. This required devoting time, thought, and resources to numerous "infrastructural" areas. One was to build a pool of referees who understood the nature and quality of material that I hoped to publish (don't underestimate the effort it takes to get quality reviews!). Other goals were to solicit contributions from a wide range of practitioners and researchers; appoint a small, working editorial board and department editors; develop author and reviewer guidelines; and identify appropriate special-issue topics and solicit proposals for consideration.

It's interesting to reflect on the highlights and challenges in the 1984 time frame, when the inaugural issue appeared. The Japanese Fifth-Generation Computer Project had recently been announced, and, among other things, logic programming and work in intelligent systems were being widely investigated—as were object-oriented programming languages, relational-database tools and methodologies, and general-purpose distributed systems. On the other hand, software quality, testing, and the maintenance and evolution of large legacy systems had become significant problems, and reliable software was difficult to develop.

The Internet was being used more widely in industry and academia. In 1985, I initiated a major effort to handle submissions electronically. I believe *Software* was one of the first professional-society periodicals to track submissions and reviewers this way. We now take the Internet for granted in article and reviewer processing, just as we do online access to articles, but this generally wasn't the case in the mid-'80s.

Ted Lewis

Executive director of the Center for Homeland Defense and Security at the US Naval Postgraduate School and EIC 1988–1991

My main challenge as EIC was to strike a balance between research papers and practical topics. Accordingly, we grew several specialized columns during that time while maintaining a high level of sophistication in the research papers. Also, *IEEE Software* was a "specialty magazine" at the time, far less mainstream than it is today. I tried to broaden its appeal, somewhat following the model of *Computer* magazine but with the spotlight on software.

continued on p. 12 ...

1994

- The "Gang of Four" releases *Design Patterns: Elements of Reusable Object-Oriented Software*.
- Marc Andreessen and Jim Clark found Netscape.
- The Standish Group publishes the first edition of the controversial Chaos reports documenting the prevalence of software project failures.
- Netscape releases its first browser.

1995

- The first wiki, Ward Cunningham's WikiWikiWeb, debuts.
- Microsoft releases Windows 95.
- The first Apache server, v0.6.2, is released.
- Java is released at Sun World Conference.
- Rasmus Lerdorf creates PHP/FI, the precursor to PHP.
- Yukihiro Matsumoto releases Ruby.
- Netscape and Sun Microsystems announce JavaScript, the precursor to the ECMAScript standard and several dialects of Web scripting languages.

1996

- The W3C releases the initial draft of XML.
- Gartner introduces the term "service-oriented architecture" (SOA).

1997

- The Object Management Group adopts UML as a standard.
- Eric Raymond presents "The Cathedral and the Bazaar" at Linux Kongress.
- Gregor Kiczales and his team at Xerox PARC introduce aspect-oriented programming.
- Eric Gamma and Kent Beck write the basics of JUnit while flying over the Atlantic.

1998

- SGML Open changes its name to Oasis to reflect an expanded scope, including XML and other related standards.
- The US Naval Postgraduate School offers the world's first doctoral program in software engineering.
- The IBM Software Group begins developing what would become the Eclipse platform.
- Sergey Brin and Larry Page present Google in "The Anatomy of a Large-Scale Hypertextual Web Search Engine" and founded Google.

1999

- Kent Beck writes first book on XP, *Extreme Programming Explained.*
- IBM senior executives decide to fully support Linux on the company's servers.
- Sun Microsystems releases J2EE 1.2, the first public edition of Java Platform Enterprise Edition.
- Y2K fever spreads worldwide, causing many (not all) institutions to spend money and effort to avoid anticipated catastrophe.

2000

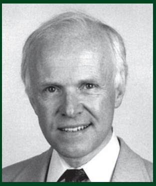
- The Y2K bug turns out to be not such a big deal, after all.
- The first international XP conference is held.
- Roy Fielding discusses Web services in his doctoral dissertation, *Architectural Styles and the Design of Network-Based Software Architectures.*
- The OMG starts work on model-driven architecture.
- Microsoft releases C#.
- Martin Fowler publishes *Refactoring: Improving the Design of Existing Code.*

2001

- The W3C holds the first workshop on Web services.
- The Agile Manifesto is released.
- The first agile development conference, XP Universe, is held.
- The Spybot worm emerges.
- Microsoft launches the first release of the .NET framework.

2002

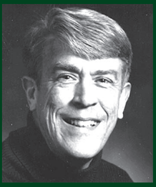
continued from p. 9 ...



TOM DEMARCO
Principal of the Atlantic Systems Guild and coauthor of *Peopleware: Productive Projects and Teams*; tdemarco@systemsguild.com

"Nichification," mid-1980s. At some point in the middle of the decade, the software industry began to fragment into niches. Prior to this time, we'd all been largely homogenous. The holy grail for progress had been viewed as the development of a so-called higher-order language, one that would be as grand a step beyond the second-generation compiled languages as they had been beyond assembly language. It never happened. Once this false goal was discarded, real progress was possible.

Apple's Applications Store, 2008. Apple's concept of a brokered market for independently developed modular applications could begin a sea change in the way software is produced, priced, and marketed. So far it's just toy apps for the iPhone and iPod, but look out when applications for desktops and laptops begin to use the same vehicle and when open source protocols for application suites begin to be promulgated.



ROBERT GLASS
Publisher and editor, the *Software Practitioner*, and a visiting professor at Griffith University; bob@robertlglass.com

Orders-of-magnitude fallacy. The thing I remember most clearly from 25 years ago was the belief, prevalent almost everywhere in the field, that we were about to turn a significant corner and improve software productivity by what was called at the time "orders of magnitude." Somehow, many believed, we would be producing software 10 or more times faster as the result of the new technologies that everyone was touting. I was even interviewed for a job in which producing that breakthrough was to be my charter. I turned the job down, on the grounds that neither I nor anyone else could produce what they wanted.

I'm happy to say that, by the end of the 1980s, we software professionals had begun to realize

continued on p. 13 ...

Reflections

... continued from p. 10

At the beginning of my tenure, the idea of automated design via CAD/CAM-like tools was on the ascendancy. Today we take these tools for granted and, in fact, have agreed on a unified software design notation and toolset. So, 20 years ago, software was maturing from a coding discipline into a design discipline. However, so much was yet to come: agile programming, Extreme Programming, and rapid-development environments, for example, had still not gotten on our radar screen.

Carl Chang

Professor and chair of the Department of Computer Science, Iowa State University, and EIC 1991–1994

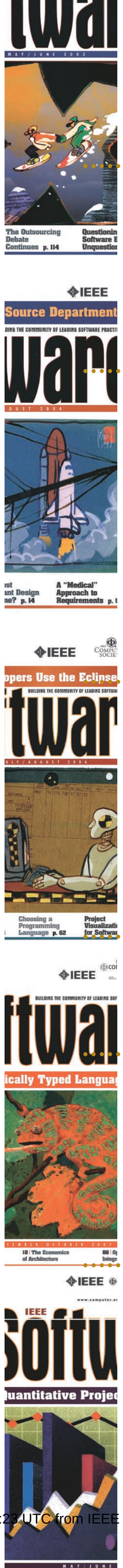
My goal was to foster technology transfer by publishing innovative content that addressed readers from both camps—researchers who are doers, and practitioners who are thinkers. My first step involved balancing the composition of the editorial board, choosing half from industry and half from academia. In 1991, I established an Industrial Advisory Board, a new and unprecedented group of advisors for the Computer Society. The two boards interacted very well, and our joint annual meeting gathered 30–35 people annually. Each two-day meeting resulted in solid editorial development plans, which significantly improved magazine content and made it more relevant to readers. We also conducted the first-ever Computer Society reader survey (perhaps first in the IEEE as well) in 1991. The message we got from more than 200 returns was loud and clear: feed the readers with content that's cutting-edge yet relevant to their professions.

Two of the most profound advances during my tenure involved the Capability Maturity Model (CMM) and requirements engineering.

We published two articles (point-counterpoint style) on the SEI's CMM in the July 1991 issue. We continued from there, regularly covering process improvement topics and experience reports. As a result, we received many letters from readers. Debates ranged from "Are capability evaluations just wishful thinking?" to "There is more than one way to measure process maturity." Clearly, *IEEE Software* played a pivotal role in shaping the SEI's maturity model early on.

IEEE Software also played a central role in fostering requirements engineering (RE), a relatively young area of study. When Alan Davis and I looked for contributions in 1991, we found there weren't enough reports in the emerging area. We decided to launch the IEEE International Conference on Requirements Engineering (ICRE), holding the first one in April 1994. We also decided to shrink-wrap the March 1994 issue of *IEEE Software* together with the ICRE proceedings for conference attendees. This was a starting point for fostering RE as an emerging discipline (in the software engineering sense). After the fifth ICRE in 2002, it merged with the IEEE International Symposium on Requirements Engineering, thus creating the IEEE International Requirements Engineering Conference. Finally, we had a robust community in which RE researchers and practitioners could interact. This enthusiasm has continued, and RE 2008 was just held in September in Barcelona.

continued on p. 14 ...



2003

- SOAP becomes a W3C recommendation on exchanging data in XML over the Internet.

2004

- IBM forms the Eclipse Foundation.
- David Heinemeier Hansson releases Ruby on Rails.
- The first O'Reilly Media Web 2.0 Conference is held.

2005

- Jesse James Garrett coins the term Ajax to stand for Asynchronous Javascript and XML.
- The Internet reaches one billion users.

2006

2007

- IBM Rational announces the Jazz project to support collaborative software development.

2008

- *IEEE Software* celebrates its 25th birthday.

continued from p. 11 ...

that that elusive goal was in fact unreachable. No matter how much we wished it so, nothing in the present or visible in the future was going to produce that kind of benefit.

And then, when the 1990s came around, people began predicting that we were going to achieve order-of-magnitude improvements in the *quality* of our software. Ah, but that's a story for another time!

WATTS HUMPHREY

Leader of the Capability Maturity Model, Personal Software Process, and Team Software Process programs, Software Engineering Institute, and author of *Managing the Software Process*; watts@sei.cmu.edu

Coda opens a door, 1987. In scaling-up systems, control logic has been manageable, but data has always been a problem. As the number of users of shared files increase linearly, their possible interactions increase as the square. The systems recovery and cleanup workload then increase exponentially. In 1987, Mahadev Satyanarayanan and James Kistler started a long-term effort to develop the Coda file system, which introduced optimistic replication coupled with distributed cache management and user control. By eliminating the need for resolving all potential conflicts centrally, mobile distributed-computing networks can now be fast and reliable, even with unreliable networks and components.

People CMM, 1995. Software people rarely agree on much, but they do agree that human capability is most important in producing great software. Although this is universally recognized, it gets scant attention in computer science. What makes some people good at software development and others merely so-so? The first orderly effort to address this subject was in 1995 when Bill Curtis and his SEI colleagues William Hefley and Sally Miller introduced the People Capability Maturity Model. If this framework motivates an orderly attack on the human issues in computer science and software engineering, it could start another software revolution.

RALPH JOHNSON

Computer science professor, University of Illinois, and coauthor, *Design Patterns: Elements of Reusable Object-Oriented Software*; johnson@cs.uiuc.edu

Linux leads to open source's coming of age, 1991. The most important event in the creation of open source was the release of Linux in 1991, not because Linux was so great (it's one of many Unix clones, and BSD advocates argue that it isn't even the best open source one) but because of the process Linus Torvalds developed for it. This process, described well by Eric Raymond in "The Cathedral and the Bazaar," has made open source an important part of the software industry. It has led to both standardization (for operating systems) and creativity (scripting languages and Web development toolkits). Recently, it has had a big impact on academic software engineering research, because academics can now study the development process of large, widely used software systems without signing nondisclosure agreements.

DIOMIDIS SPINELLIS

Associate professor in the Department of Management Science and Technology and director of the Information Systems Technology Laboratory, Athens University of Economics and Business, and author of *Code Quality: The Open Source Perspective*; dds@aueb.gr

Scripting languages gain significant mind-share in the late 1980s. At the time, many programmers were already using special-purpose scripting languages, such as the Unix shell, awk, and Rexx. However, the distribution of Perl (in 1987) and Tcl (a year later) as open source software stimulated their widespread adoption, their porting to low-cost platforms, and a drive for piling features onto them.

Perl's popularity led to the development of an archive containing more than 13,000 modules today and to a cycle of software reuse and ever more-specialized contributions. Later, the flexibility of scripting languages fueled innovation on the fledging Web, with many groundbreaking applications such as Wikipedia relying on them. Indirectly, scripting languages' success contributed to the adoption of a bytecode interpretation portability layer, garbage collection, and extensive runtime libraries for Java and C#.

GUIDO VAN ROSSUM

Creator of Python; guido@python.org

Open source gets named, 1998. Invented earlier that year for the public release of the source code for Netscape Navigator, the term was selected by vote at an event organized by Tim O'Reilly. The leaders of many significant open source projects were present, and Eric Raymond successfully argued for using "open source" in preference over the politically charged term "free software." The term's widespread use was virtually guaranteed by a press conference held the evening after the vote.

... continued from p. 12

Alan M. Davis

Professor, College of Business, University of Colorado at Colorado Springs, and EIC 1995–1998

My primary goal was to see how useful we could make *IEEE Software* for practitioners. The idea was to transfer as many "ripe fruits" to practice as possible: a combination of best practices (that is, get more practitioners to do things the way that the most effective practitioners did them) and best short-term research (that is, get more practitioners to adopt the most practical research results).

As an industry, we don't track practices particularly well, let alone best practices. How successful have we (all the thousands of people who have contributed to *IEEE Software*) been?

Research is a funny thing. For it to be good, it has to be risky. And risky means that most of it will never get to practitioners' "real world." So, the magazine has an almost impossible mission: to somehow sift through the plethora of "stuff" to find the one-in-a-thousand idea that practitioners could and should adopt.

I'd like to believe that research has changed considerably in 25 years, but I don't think it has. Many of the research papers in 1995 could appear as well in the latest volume—all they'd need to do is to freshen up terminology. Most software engineering research is still weak in comparison to other fields. Most papers still just report "I invented something new" with little demonstration of practicality. Papers that try to bridge this gap tend to apply their "new idea" to a student project or an exemplar. Few researchers dare to apply their ideas to real industry problems. The reason is simple: the problems are tough. They don't actually fit the assumptions researchers make so that their research approach will work; unfortunately, this is exactly what makes real-world software development so difficult.

Regarding practice, I sense that the software industry has changed its practices considerably in the past 25 years. The length of the software development life cycle has decreased dramatically, from a few years to a few months. I suspect that "incremental development" results more from project managers learning from each other and their past mistakes than from any published paper. The success of shorter life cycles also results naturally from globalization of and growing competition within the software industry and from the ubiquity of software.

There's a growing bimodal distribution of power and control in the software industry. We see the widespread adoption of techniques that empower software developers to make more decisions, including (remarkably!) business decisions; we also see a tendency to have fewer CIOs in industry. On the other hand, many implementers of so-called process improvement try to centralize control and authority.

Software development is also becoming specialized. Twenty-five years ago, most companies (regardless of industry) hired their own IT personnel. Now, the trend appears to be the opposite: most nonsoftware companies no longer see IT as a core competency; instead, they outsource it as part of their infrastructure or overhead.

Finally, seemingly more attention is being given to measuring process than measuring progress. I can't explain this; it seems counter-evolutionary to me. I hope it's just a fad.

Steve McConnell

CEO and chief software engineer, Construx Software, and EIC 1999–2002

My overriding goal was to fulfill the magazine's mission: to "build the community of leading software practitioners." New ideas in software come from both practice and academia, so I tried to encourage contributions from practically oriented academics and from theoretically oriented practitioners.

The years I was EIC were exciting times in the software world. The dot-com bubble reached its maximum and then burst. The Internet evolved from being a place for toy-sized applications to become a serious focus for major software development initiatives. Open source became a major focus. Y2K came and went. Agile programming got its start. It was a very busy time!

Warren Harrison

Professor of computer science, Oregon State University, and EIC 2003–2006

When I became EIC, I was stunned to learn that 44 percent of the readers identified themselves as corporate or technical management, and more than 22 percent were "senior developers," likely involved in leading small groups of programmers. In essence, that was 66 percent in management. Fearing that *IEEE Software* was on its way to becoming *IEEE Software Management*, I was determined to move the magazine back toward the center and address the concerns of the industry's everyday foot soldiers (a move that my predecessor, Steve McConnell, had already begun). I note with pleasure that only 46 percent of the magazine's readers now identify themselves as management.

No doubt the most profound change during my tenure as EIC was the legitimization of agile methods. The Agile Manifesto was written in 2001, and most of the agile methods popular at that time were developed in the late 1990s. When I took over the magazine in January 2003, agile methods were viewed as revolutionary techniques developed by mavericks who were tweaking the nose of the well-entrenched waterfall life cycle. By the time I stepped down in December 2006, agile methods were well entrenched in the industry and had gained widespread acceptance.