Gems of Software Engineering Wisdom

34 years of IEEE Software history in 1000 quotes

Editor: Željko Obrenović

Gems of Software Engineering Wisdom

34 years of IEEE Software history in 1000 quotes

Željko Obrenović

This book is for sale at http://leanpub.com/se-wisdom

This version was published on 2018-03-09

Contents

CONTENTS

CONTENTS

Foreword

This book contains a selection of 1000 quotes from IEEE Software[1]. IEEE Software is a leading software engineering magazine with a very rich history. Since 1984, many of the leading software engineering professionals have contributed to IEEE Software. The content is based on the material from the IEEE Software history website[2]. Visit this site to find more details about IEEE Software and its history.

Selected quotes are not intended to serve as a comprehensive overview of the history of software engineering. Rather, the book is designed as a "coffee table book". It is intended for casual reading, offering interesting, educative, thought-provoking, and sometimes controversial quotes.

Zeljko Obrenovic

[1] https://publications.computer.org/software-magazine/

[2] https://obren.info/ieeesw

1984


IEEE
Software
SYSTEM DESIGN
Also in this issue:
Ada to Silicon
Message/Object Programming
Knowledge and Database Management
Verification and Validation
IEEE COMPUTER SOCIETY
PREMIERE ISSUE
JANUARY 1984

“Many of the **challenges facing the software industry** today are a direct result of our **insatiable appetite** for new computer-based systems applications. Others confront us simply because we have not managed to successfully solve a large number of **problems that we ourselves created many years ago**.”

---

*Bruce D. Shriver, **From the Editor-in-Chief**, IEEE Software, January 1984.*[3]

[3] DOI: 10.1109/MS.1984.233385

"There probably isn't a best way to build the system or even a major part of it. Much more important is to **avoid choosing a terrible way** and to **have a clear division of responsibilities** among the parts."

Butler W. Lampson, ***Hints for Computer System Design***, *IEEE Software, January 1984.*[4]

[4] DOI: 10.1109/MS.1984.233391

“Designing a computer system is very **different from designing an algorithm**: … the requirement - is less precisely defined more complex, and more subject to change; the system has much more internal structure - hence, many internal interfaces; and the measure of success is much less clear. The designer usually finds himself flondering in a sea of possibilities, unclear about how one choice will limit his freedom to take other choices or affect the size and performance of the entire-system.”

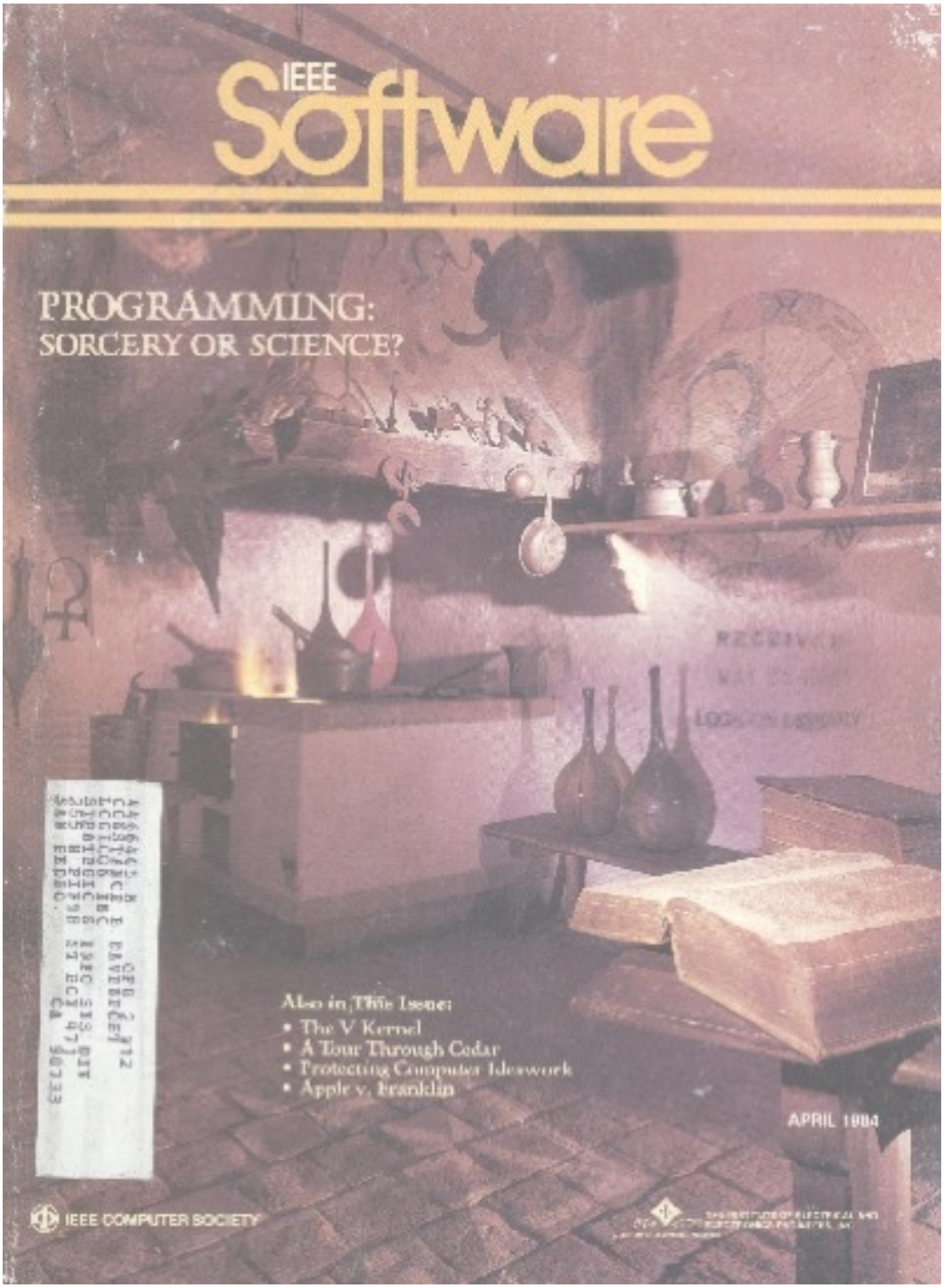
Butler W. Lampson, ***Hints for Computer System Design****, IEEE Software, January 1984.*[5]

[5] DOI: 10.1109/MS.1984.233391

“Booch offers a software design methodology, which he calls ‘**object-oriented design**’ in contrast to earlier popular methods he designates as either **functional** or **data-oriented**. In contrast to suggestions that we either identify a program’s principal function and describe it in the top box in a hierarchy chart or carefully identify the patterns of data and their flows, Booch suggests, ‘Define the problem, develop an informal strategy; formalize the strategy.’”

Peter G. Anderson, ***Review of Grady Booch’s book ‘Software Engineering with Ada’****, IEEE Software, January 1984.*[6]

[6] DOI: 10.1109/MS.1984.233391


IEEE
Software
PROGRAMMING:
SORCERY OR SCIENCE?
Also in This Issue:
• The V Kernel
• A Tour Through Cedar
• Protecting Computer Ideswork
• Apple v. Franklin
APRIL 1984
IEEE COMPUTER SOCIETY

“One does not use structural engineering analysis to build a **sandcastle**. But neither does one choose the prize-winning builder of sandcastles as architect for **a tower block of offices** in a city.”

Sir Charles Antony Richard Hoare, ***Programming: Sorcery or Science?***, *IEEE Software, April 1984.*[7]

[7] DOI: 10.1109/MS.1984.234042

“I believe that in our branch of engineering, above all others, the academic ideals of **rigor and elegance will pay the highest dividends** in practical terms of reducing costs, increasing performance, and in directing the great sources of computational power on the surface of a silicon chip to the use and convenience of man.”

Sir Charles Antony Richard Hoare, ***Programming: Sorcery or Science?****, IEEE Software, April 1984.*[8]

[8] DOI: 10.1109/MS.1984.234042


IEEE
Software
RECEIVED
AUG 6 1 1984
LOGICON LIBRARY
CAPITAL-INTENSIVE
SOFTWARE TECHNOLOGY
JULY 1984
Also in this issue
Built-In Quality in Large Systems
Software Renewal: A Case Study
Dynamic Task Scheduling
Global Naming in Distributed Systems

“The greater speed of technical change means that **capital investment must be recovered more quickly** and that enhancement and evolution consume proportionately more resources than in a slowly changing technology. This contributes to the fact that **maintenance and enhancement** are the **dominant costs** in the software life cycle today.”

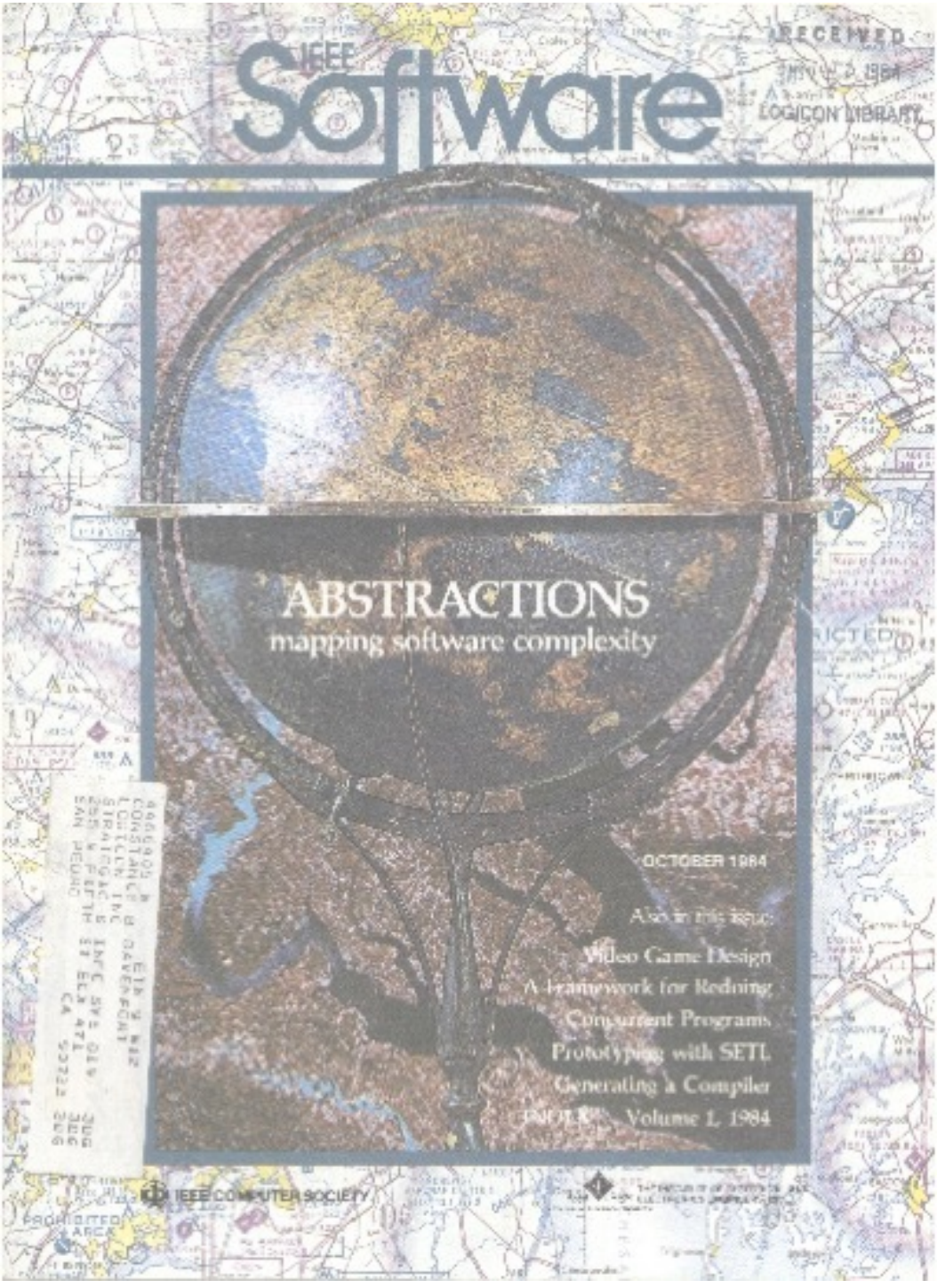
Peter Wegner, ***Capital-Intensive Software Technology***, *IEEE Software, July 1984.*[9]

[9] DOI: 10.1109/MS.1984.234384

“Periods of **rapid technological change** require **more innovation** and **greater risks** than periods of stability.”

Peter Wegner, ***Capital-Intensive Software Technology Conclusion***, *IEEE Software, July 1984.*[10]

[10] DOI: 10.1109/MS.1984.234706


IEEE
Software
RECEIVED
LOGICON LIBRARY
ABSTRACTIONS
mapping software complexity
OCTOBER 1984
Also in this issue:
Video Game Design
A Framework for Redoing
Concurrent Programs
Prototyping with SETL
Generating a Compiler
Volume 1, 1984
IEEE COMPUTER SOCIETY

"An **abstraction is a simplified description**, or specification, of a system that **emphasizes** some of the system's details or properties while **suppressing** others. A good abstraction is one that **emphasizes details that are significant** to the reader or user and suppresses details that are, at least for the moment immaterial or diversionary."

Mary Shaw, ***Abstraction Techniques in Modern Programming Languages****, IEEE Software, October 1984.*[11]

[11] DOI: 10.1109/MS.1984.234384

1985


IEEE
Software
JANUARY 1985
On Formalism in Specifications
Also in this issue:
Algorithm Animation
Application Generators
Software Engineering as a Profession
Languages for Developing Information Systems
Human-Computer Interfaces
IEEE COMPUTER SOCIETY

"The use of **formal notation** does not, however, preclude that of **natural language**. In fact, mathematical specification of a problem usually leads to a better natural-language description. This is because formal notations naturally lead the specifier to **raise some questions** that might have remained unasked, and thus unanswered, in an informal approach."

Bertrand Meyer, ***On Formalism in Specifications****, IEEE Software, January 1985.*[12]

[12] DOI: 10.1109/MS.1985.229776

IEEE Software
MARCH 1985

Specifications

Make thee an ark of gopher wood; rooms shalt thou make in the ark, and shalt pitch it within and without with pitch.

And this is the fashion which thou shalt make it of: the length of the ark shall be three hundred cubits, the breadth of it fifty cubits, and the height of it thirty cubits.

A window shalt thou make to the ark, and in a cubit shalt thou finish it above; and the door of the ark shalt thou set in the side thereof; with lower, second, and third stories shalt thou make it.

Anna, a Specification Language for Ada

Other Ada articles in this issue:
- Transformation Tools
- Debugging
- Programming-in-the-large
- Testing

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

“In essence, **programming-in-the large** involves the two complementary activities of modularization and interface control. **Modularization** is the identification of the major system modules and the entities those modules contain, where entities are language elements that are given names, such as subprograms, data objects, and types. **Interface control** is the specification and control of the interactions among entities in different modules.”

Alexander L. Wolf, Lori A. Clarke, Jack C. Wileden, ***Ada-Based support for programming-in-the-Large****, IEEE Software, March 1985.*[13]

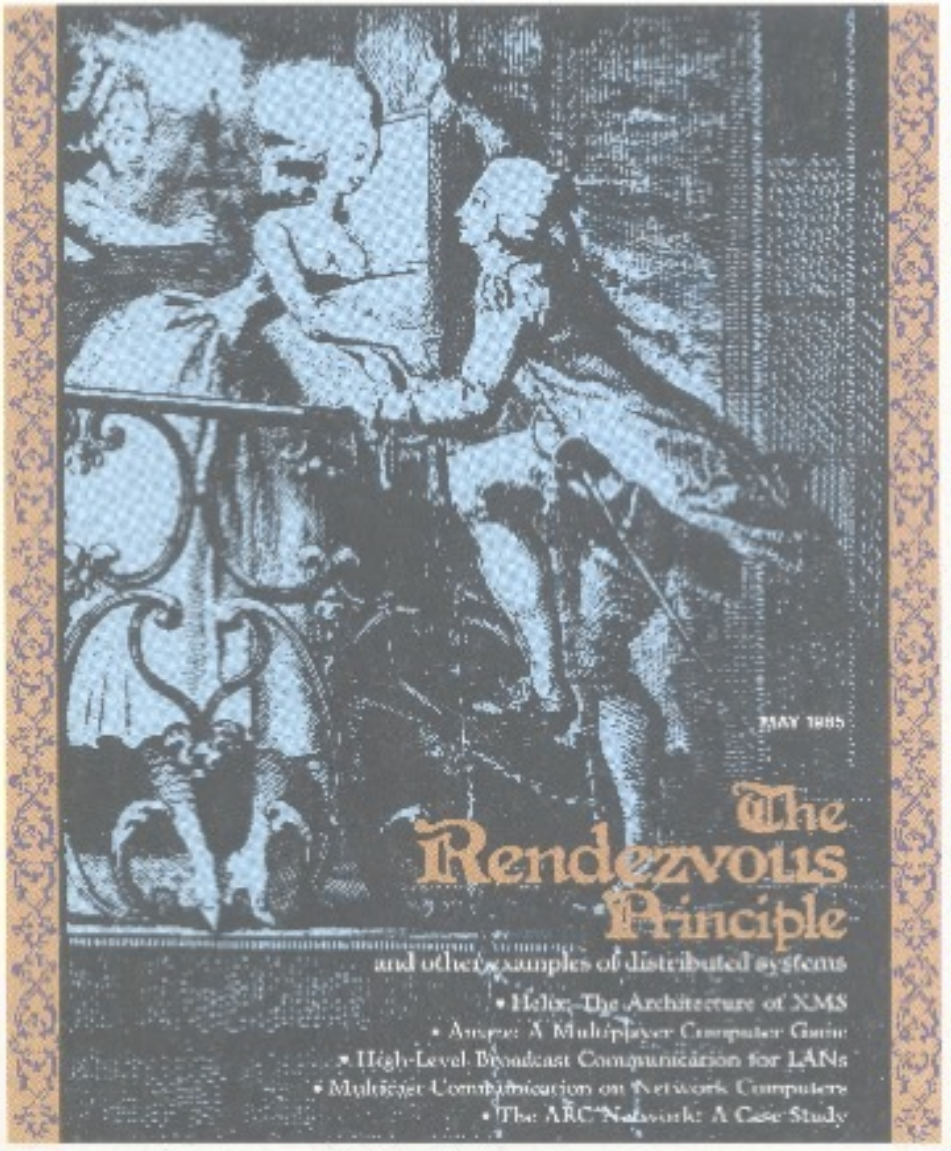
[13] DOI: 10.1109/MS.1985.230352

“Engineers may be able to design a **better interface** if they take into account the **control structures** underlying the **interface syntax**.The **syntactic**, **semantic**, and **protocol** aspects of the interface each have their **own ‘complexity.’**.”

T.E. Lindquist, ***Assessing the Usability of Human-Computer Interfaces***, *IEEE Software*, *March 1985.*[14]

[14] DOI: 10.1109/MS.1985.230052

IEEE Software



MAY 1985

The Rendezvous Principle

and other examples of distributed systems

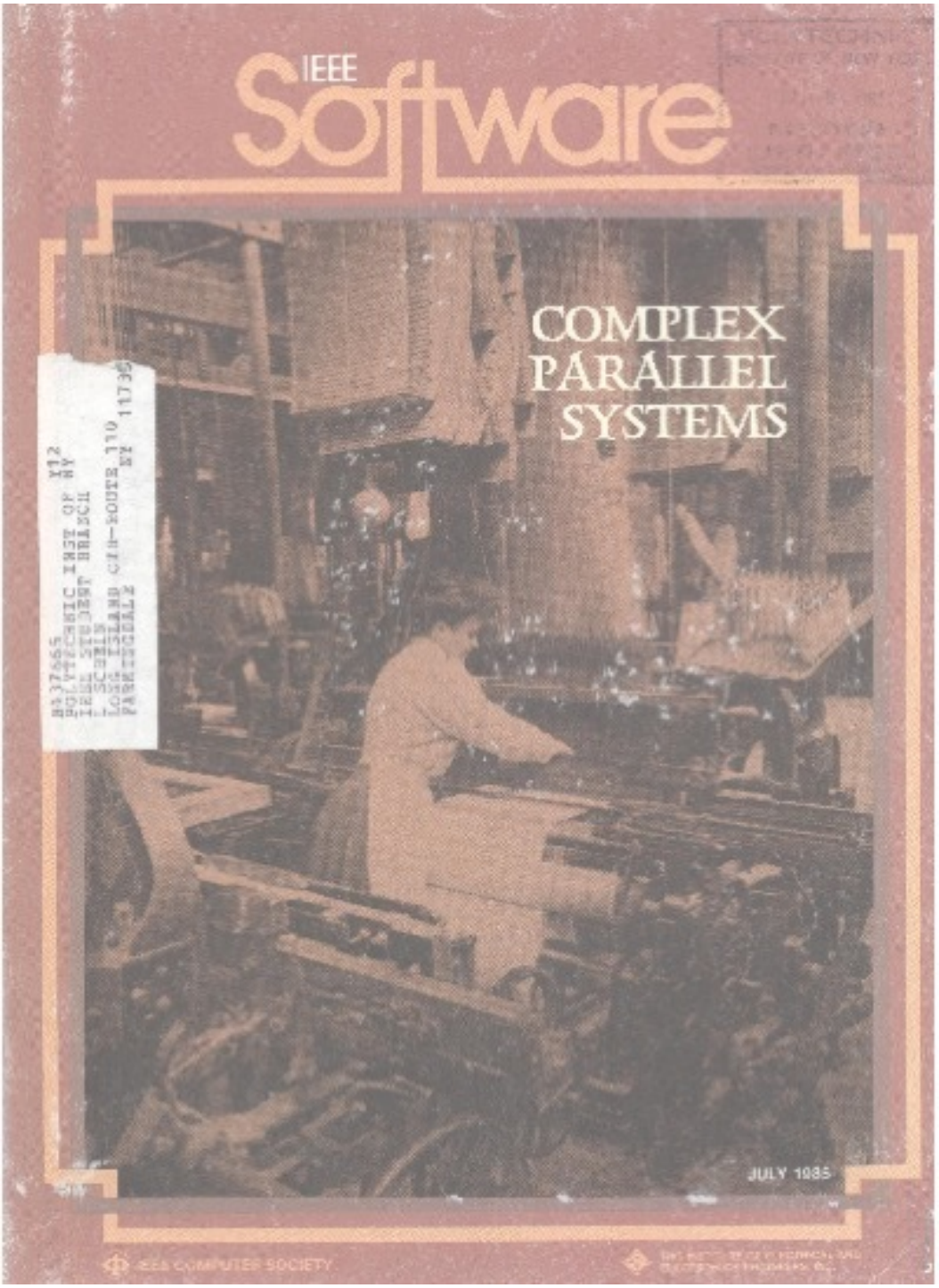
- Helix: The Architecture of XMS
- Amoeba: A Multiplayer Computer Game
- High-Level Broadcast Communication for LANs
- Multicast Communication on Network Computers
- The ARC Network: A Case Study

IEEE COMPUTER SOCIETY

"The **lack of a complete theoretical basis** for distributed computing systems need not inhibit the development of useful systems. Even without such a basis, many **technical advances have been made by individuals**, who then **share them with others**, who in turn accept useful concepts and add further innovations."

Stephen F. Lundstrom, Duncan H. Lawrie, ***Experiences with Distributed Systems****, IEEE Software, May 1985.*[15]

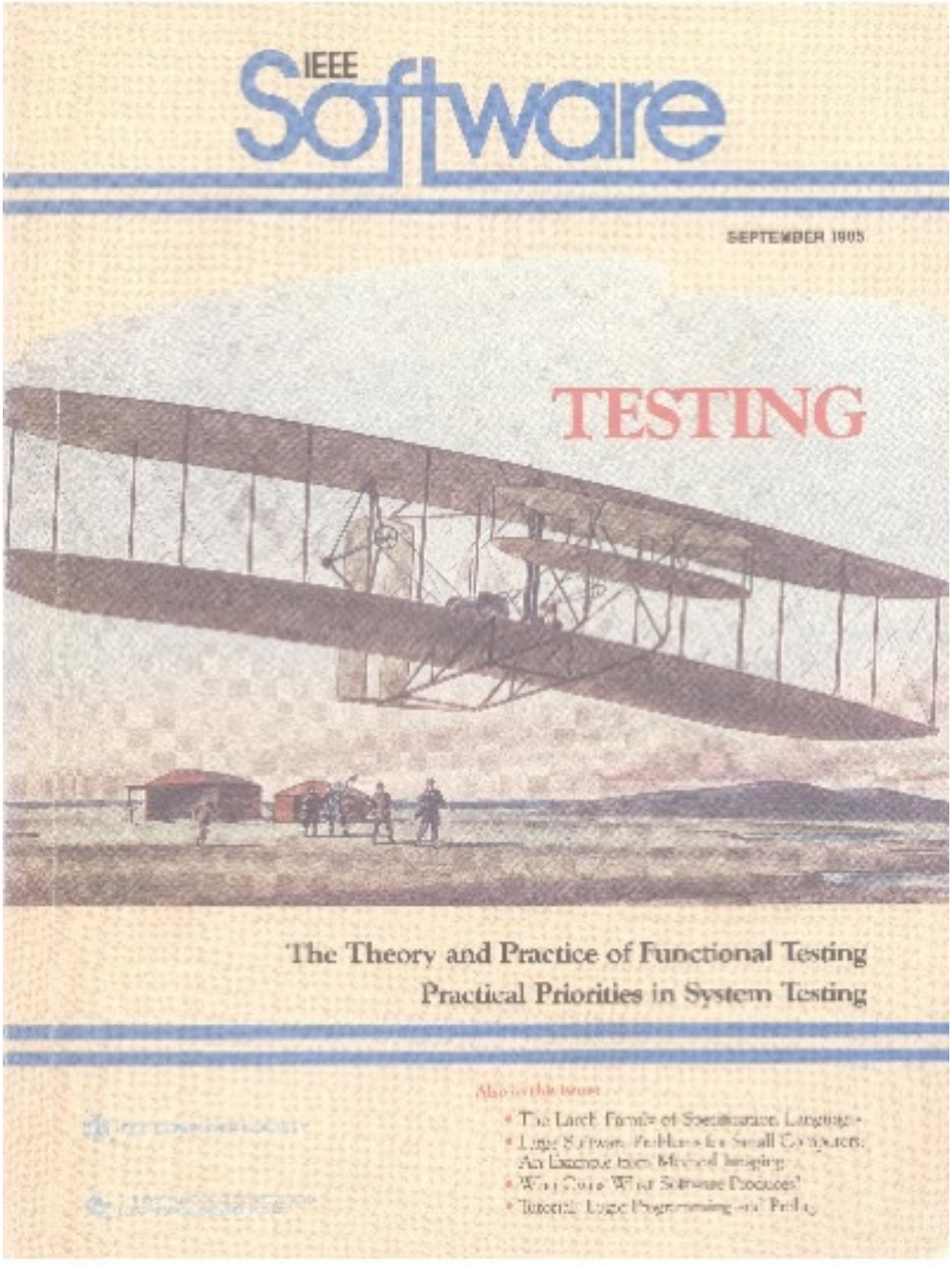
[15] DOI: 10.1109/MS.1985.230692


IEEE
Software
COMPLEX
PARALLEL
SYSTEMS
JULY 1985
IEEE COMPUTER SOCIETY

“System designers should provide for such practical issues as **deadlock** avoidance, **scheduling**, necessary and sufficient primitives to allow for **synchronization**, and the combination of **multiprogramming** (a single system dividing its time and resources among many jobs) and **multiprocessing** (multiple processing units operating on a single job).”

Joanne L. Martin, ***Operating Systems and Environments for Large-Scale Parallel Processors****, IEEE Software, July 1985.*[16]

[16] DOI: 10.1109/MS.1985.231051


IEEE
Software
SEPTEMBER 1985
TESTING
The Theory and Practice of Functional Testing
Practical Priorities in System Testing

“Program **testing** consists of scattered collection of **rules of thumb**, **coverage** measures, and testing **philosophies**.”

---

*William E. Howden, **The Theory and Practice of Foundation Testing**, IEEE Software, September 1985.*[17]

[17] DOI: 10.1109/MS.1985.231754

"Today we tend to go on for years, with tremendous effort to find that the system, which was not well understood to start with, does not work as anticipated. We **build systems like the Wright brothers** built airplanes-build the whole thing, push it off the cliff, let it crash, and start over again."

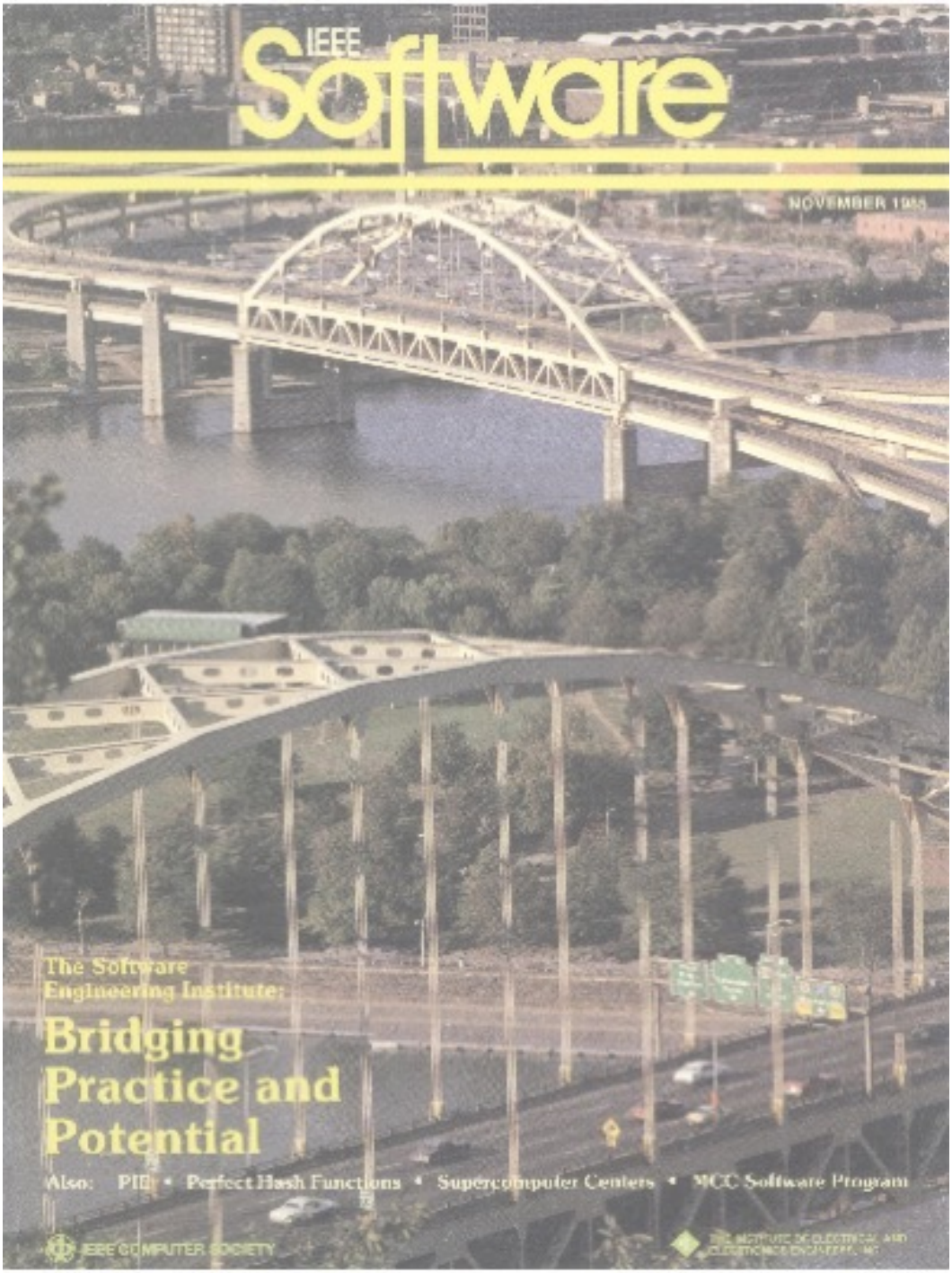
William E. Howden, ***The Theory and Practice of Foundation Testing****, IEEE Software, September 1985.*[18]

[18] DOI: 10.1109/MS.1985.231754

“When **programs rather than humans** create products, the issue of product ownership create **thorny legal problems** which the law has yet to address.”

Michael C. Gemignani, ***Who Owns What Sofware Produces?***, *IEEE Software, September 1985.*[19]

[19] DOI: https://doi.org/10.1109/MS.1985.231758

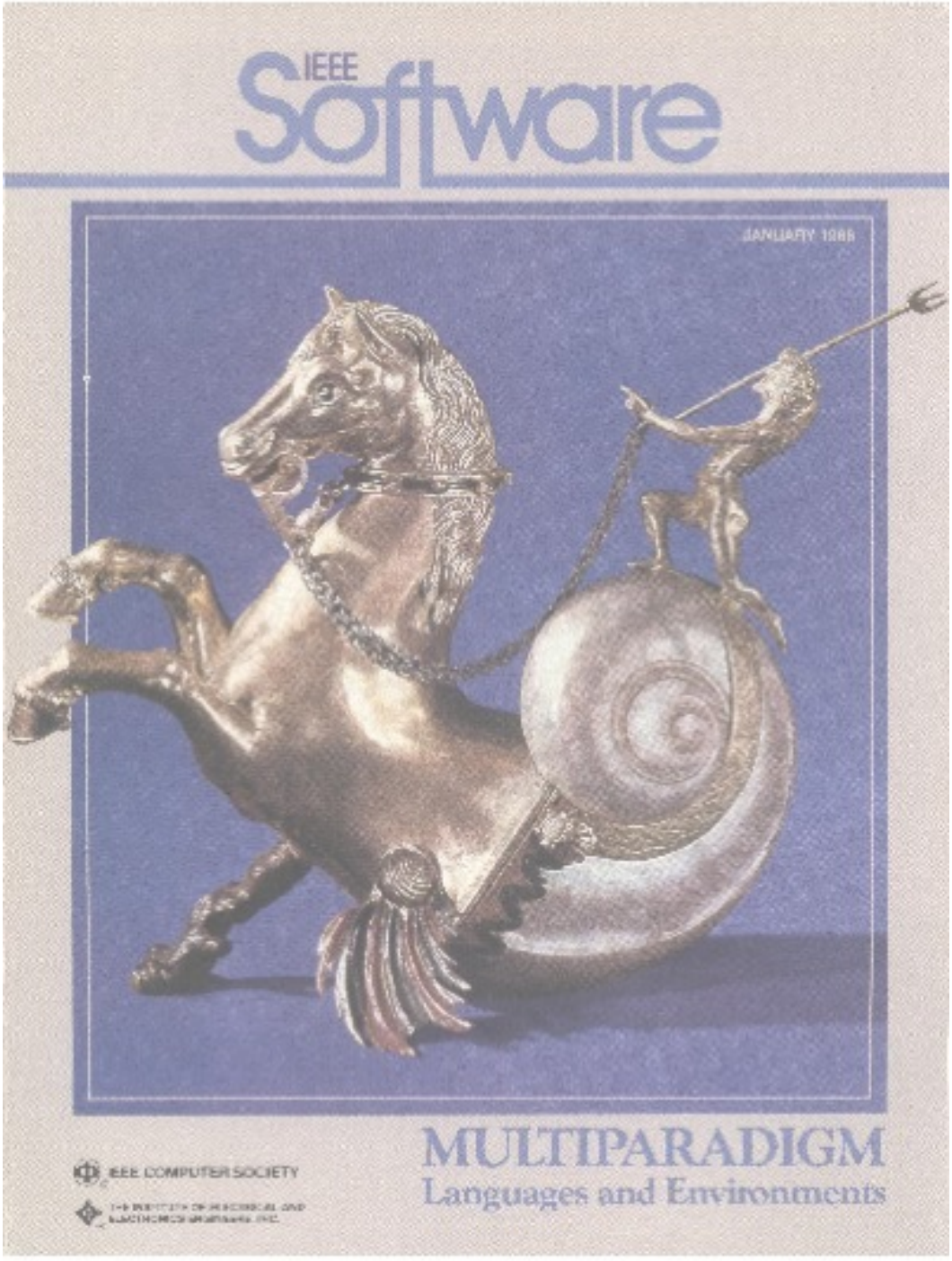

IEEE
Software
NOVEMBER 1985
The Software Engineering Institute:
Bridging Practice and Potential
Also: PIE • Perfect Hash Functions • Supercomputer Centers • MCC Software Program
IEEE COMPUTER SOCIETY

“The SEI Software Engineering Institute has established its long-range goal: Transition **new software engineering technology** into **routine common practice** to achieve significant improvements in the ability of software developers and maintainers to produce and support predictably high quality systems.”

M.R. Barbacci, Mary Shaw, ***The Software Engineering Institute: Bridging Practice and Potential****, IEEE Software, November 1985.*[20]

[20] DOI: 10.1109/MS.1985.232064

1986


IEEE
Software
JANUARY 1986
MULTIPARADIGM
Languages and Environments
IEEE COMPUTER SOCIETY

“**Multiparadigm systems** incorporating **two or more of the conventional program paradigms**. For example, the Loops system … combines features of the Lisp, functional, rule-oriented, and object-oriented paradigms.”

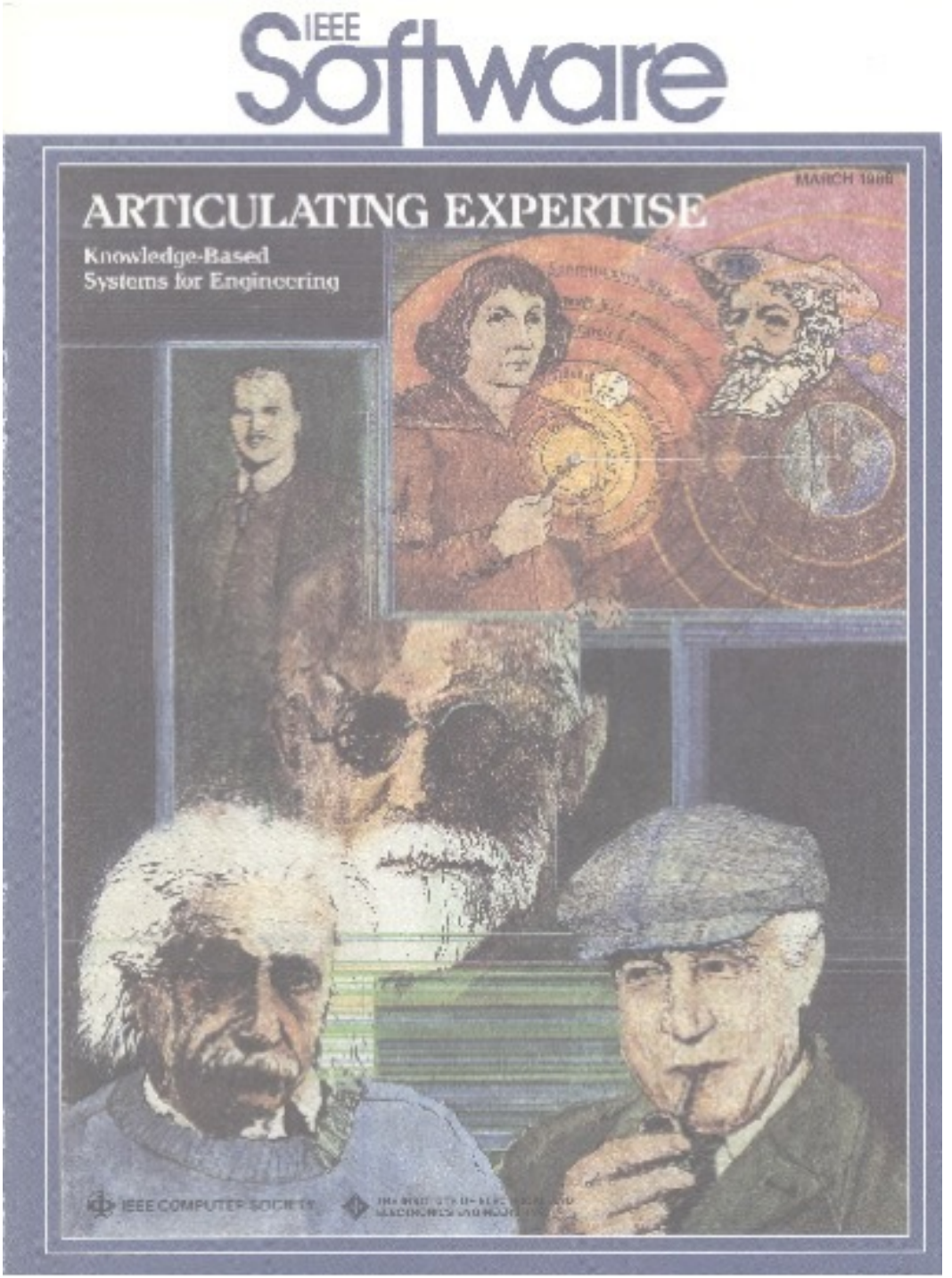
Brent Hailpern, ***Guest Editor’s Introduction Multiparadigm Languages and Environments****, IEEE Software, January 1986.*[21]

[21] DOI: 10.1109/MS.1986.232426

“The **C++** programming language was designed to make the task of **programming more enjoyable for the serious programmer**.”

Bjarne Stroustrup, ***Multiparadigm Research: A Survey of Nine Projects***, *IEEE Software*, *January 1986.*[22]

[22] DOI: 10.1109/MS.1986.232757


IEEE
Software
MARCH 1986
ARTICULATING EXPERTISE
Knowledge-Based
Systems for Engineering
IEEE COMPUTER SOCIETY

“Knowledge-based expert systems provide a programming methodology for **solving ill-structured engineering problems**. … these systems also provide a flexible software development methodology-by **separating the knowledge base** from the **inference mechanism** …”

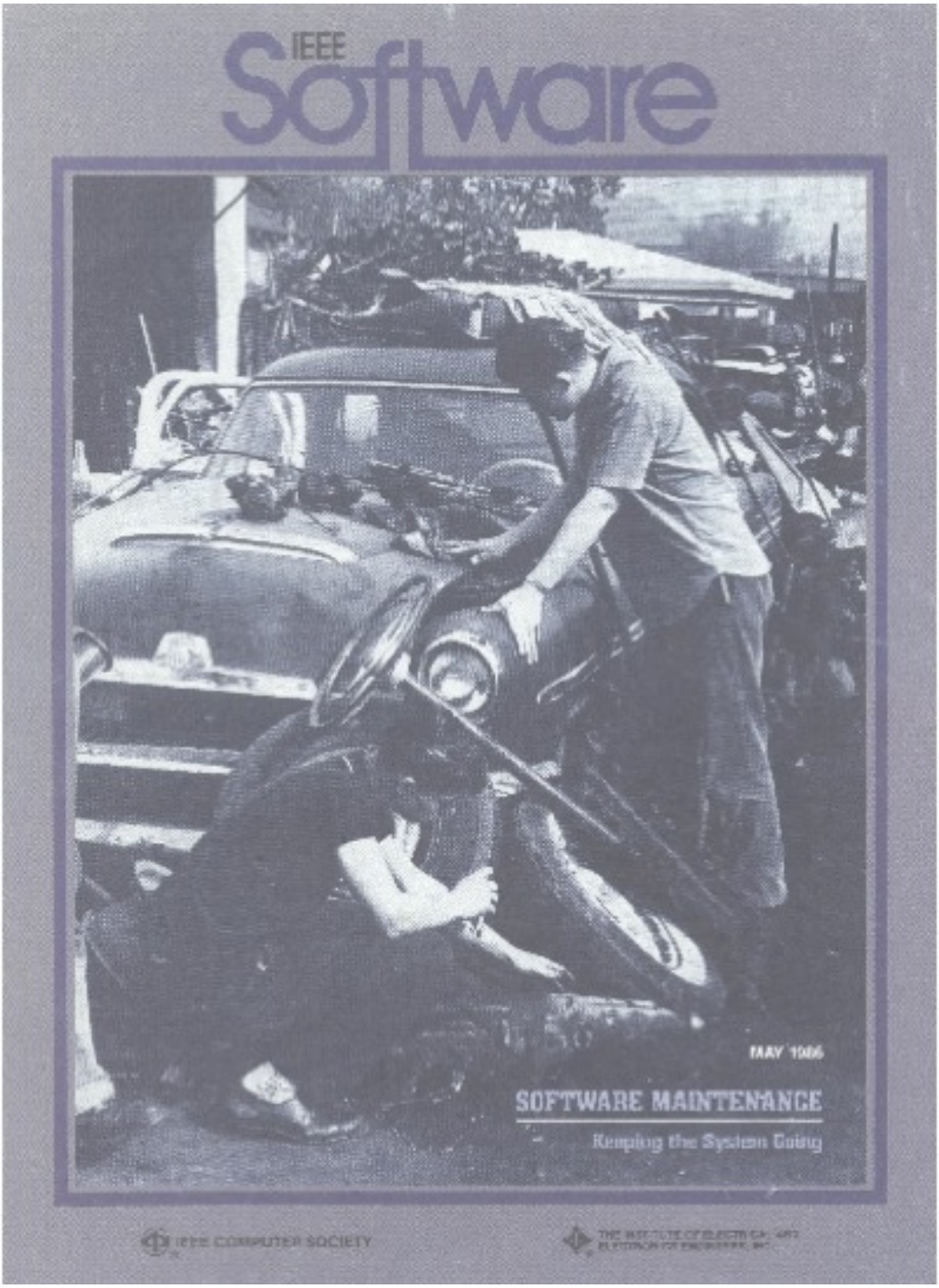
Duvvuru Sriram, Michael D. Rychener, ***Knowledge-Based Expert Systems for Engineering****, IEEE Software, March 1986.*[23]

[23] DOI: 10.1109/MS.1986.232780

“Empirically comparing **structural test coverage metrics** reveals that test sets that satisfy one metric are likely to satisfy another metric as well.”

M.D. Weiser, J.D. Gannon, P.R. McMullin, ***Comparison of Structural Test Coverage Metrics****, IEEE Software, March 1986.*[24]

[24] DOI: http://dx.doi.org/10.1109/MS.1985.230356


IEEE
Software
MAY 1986
SOFTWARE MAINTENANCE
Keeping the System Going
IEEE COMPUTER SOCIETY

“Arguing about the definition of **software maintenance** seldom leads to maintenance advances…”

---

*Robert S. Arnold, Roger J. Martin, **Software Maintenance**, IEEE Software, May 1986.*[25]

[25] DOI: 10.1109/MS.1986.233403

“**User interfaces** in the software environment are much **like spices in good recipes**; the right arrangement must be found or the food will not show its full flavor. Factors such as data availability and complexity and the size of the display must be carefully weighed and accounted for in the design of any software environment.”

Ben Shneiderman, Philip Shafer, Roland Simon, Linda Weldon, ***Display Strategies for Program Browsing: Concepts and Experiment****, IEEE Software, May 1986.*[26]

[26] DOI: 10.1109/MS.1986.233405

“One of the major challenges facing project software system managers and maintainers in the 1980’s is **how to upgrade** large, complex, embedded system, written a decade or more ago in unstructured languages according to **designs that make modification difficult**.”

Robert N. Britcher, James J. Craig, ***Using Modem Design Practices to Upgrade Aging Software Systems****, IEEE Software, May 1986.*[27]

[27] DOI: 10.1109/MS.1986.233407


IEEE
Software
JULY 1986
FIRMWARE ENGINEERING
The Interaction of Microprogramming and Software Technology
IEEE COMPUTER SOCIETY

“Computer science consists of at least three worlds - software, hardware and **firmware**. The world of firmware, developed to reduce **the semantic gap between hardware and software**, has multiple levels and intricacies, symmetries and quasi-symmetriess.”

---

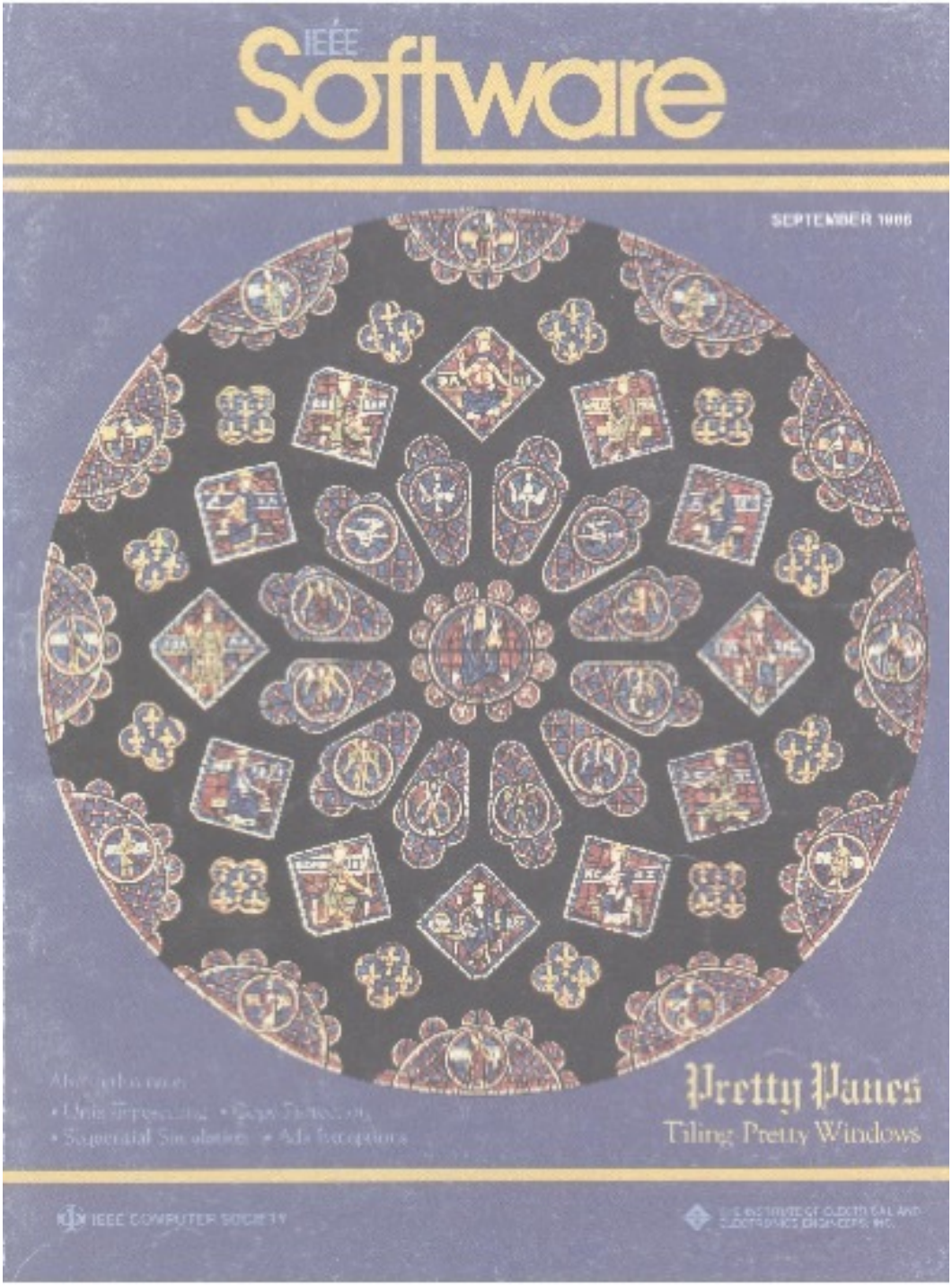
*Henry Ayling, **Escher: The Image of the Artist**, IEEE Software, July 1986.*[28]

[28] DOI: 10.1109/MS.1986.233745

“We chose IEEE Software as our forum because **firmware engineering** involves transferring theory, principle, and technique **from software to firmware**, with appropriate adaptations.”

Subrata Dasgupta, Robert A. Mueller, ***Firmware Engineering: The Interaction of Microprogramming and Software Technology****, IEEE Software, July 1986.*[29]

[29] DOI: 10.1109/MS.1986.233747


IEEE
Software
SEPTEMBER 1986
Pretty Panes
Tiling Pretty Windows
IEEE COMPUTER SOCIETY

“**Windows** serve as the **conceptual framework** for the capture, development, organization, and highlighting of information - both textual and graphic. Attention to pure **aesthetics** is an important issue in making display interfaces understandable, memorable, and appealing to users. Consequently, window aesthetics are a factor in **user satisfaction**.”

---

*Jason Gait, **Pretty Pane Tiling of Pretty Windows**, IEEE Software, September 1986.*[30]

[30] DOI: 10.1109/MS.1986.229469

“For every piece of business software sold, at least one illegal copy exists.”

Paul A. Suhler, Nadar Bagherzadeh, Miroslaw Malek, Neil Iscoe, ***Software Authorization Systems****, IEEE Software, September 1986.*[31]

[31] DOI: 10.1109/MS.1986.234396


IEEE
Software
NOVEMBER 1986
MODULA-2
Craftsmanship
at a High Level
Also . . .
Harlan Mills
on Structured
Programming
IEEE COMPUTER SOCIETY

“**Modula-2** has been adopted as the foundation for a number of **experimental systems**. The insights gained from those experiments will prove valuable to both current **designers and future users of programming systems**, no matter what the language. “

Robert P. Cook, ***Modula-2 Experiments Will Help Future Language Designs***, *IEEE Software, November 1986.*[32]

[32] DOI: 10.1109/MS.1986.234419

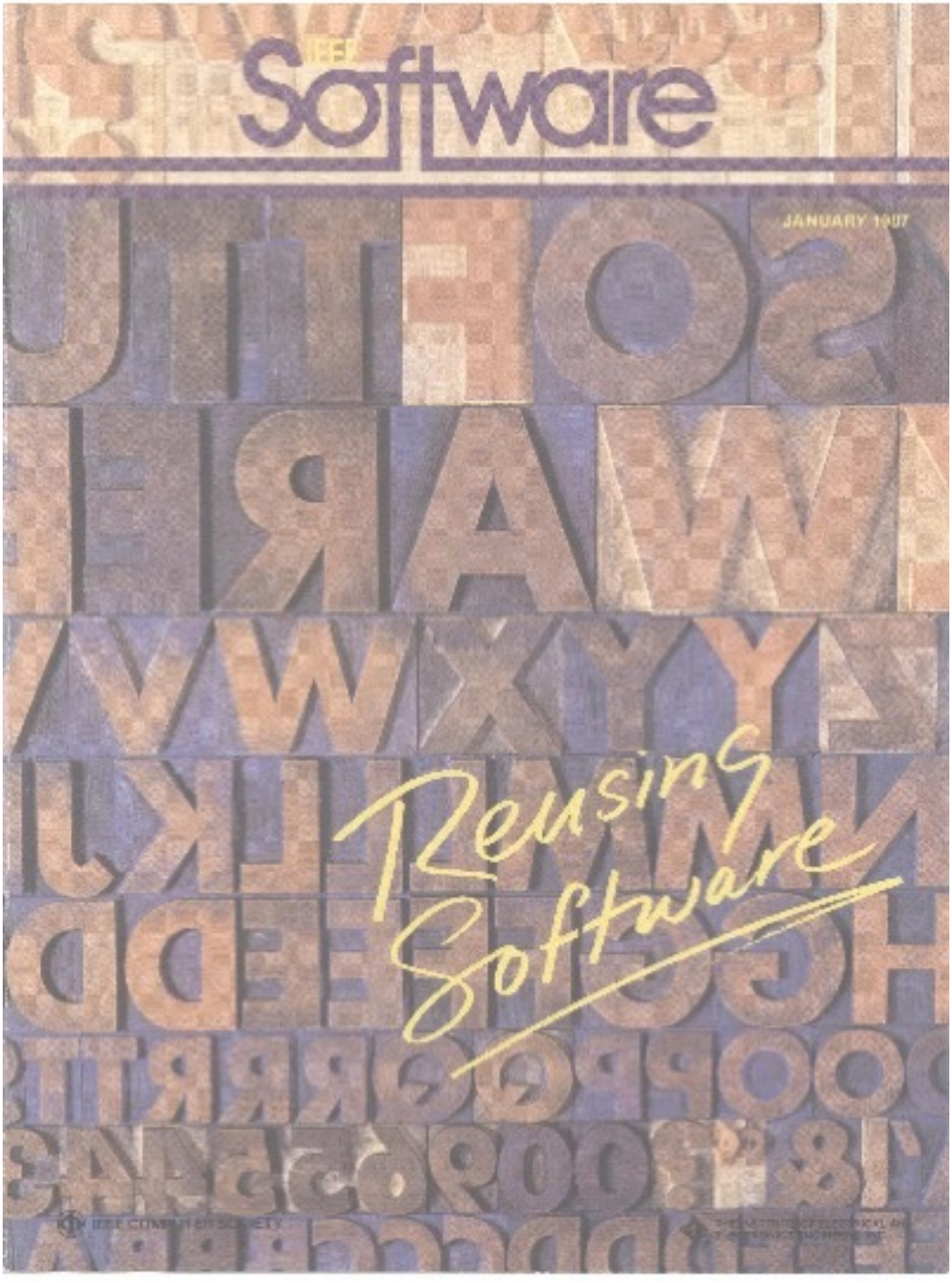
“Dijkstra’s proposal to prohibit the **GOTO** was greeted with controversy: ‘**You must be kidding!**’”

---

*Harlan D. Mills, **Structured Programming: Retrospect and Prospect**, IEEE Software, November 1986.*[33]

[33] DOI: 10.1109/MS.1986.229478

1987


IEEE
Software
JANUARY 1987
Reusing
Software

"**Reusing** and **reworking** software is not new; it has been done since the very beginnings of our industry in the early 1950s. Reuse means using an entity in a different context from that in it initially had been used. This is often called **'black-box' reuse**. When an entity is modified before it is used in the new setting, it is called 'rework' or **'white-box' reuse**."

Bruce D. Shriver, ***Editor in Chief Introduction Reuse Revisited****, IEEE Software, January 1987.*[34]

[34] DOI: 10.1109/MS.1987.229788

"To **reuse** a software component, you **first have to find it**."

Ruber Prieto-Diaz, Peter Freeman, ***Clasifying Software for Reusability****, IEEE Software, January 1987.*[35]

[35] DOI: 10.1109/MS.1987.229789

"There are two levels of **reuse** to consider: the reuse of **ideas and knowledge** and the reuse of particular **artifacts and components**."

Ruber Prieto-Diaz, Peter Freeman, ***Clasifying Software for Reusability****, IEEE Software, January 1987.*[36]

[36] DOI: 10.1109/MS.1987.229789

“**Visual programming languages** … deal with objects that do not have an inherent visual representation. This includes traditional data types such as arrays, stacks, and queues and application data types such as forms, documents, and databases. … both programming constructs and and the rules to combine these constructs should be presented visually.”

Shi-Kuo Chang, ***Visual Languages: A Tutorial and Survey****, IEEE Software, January 1987.*[37]

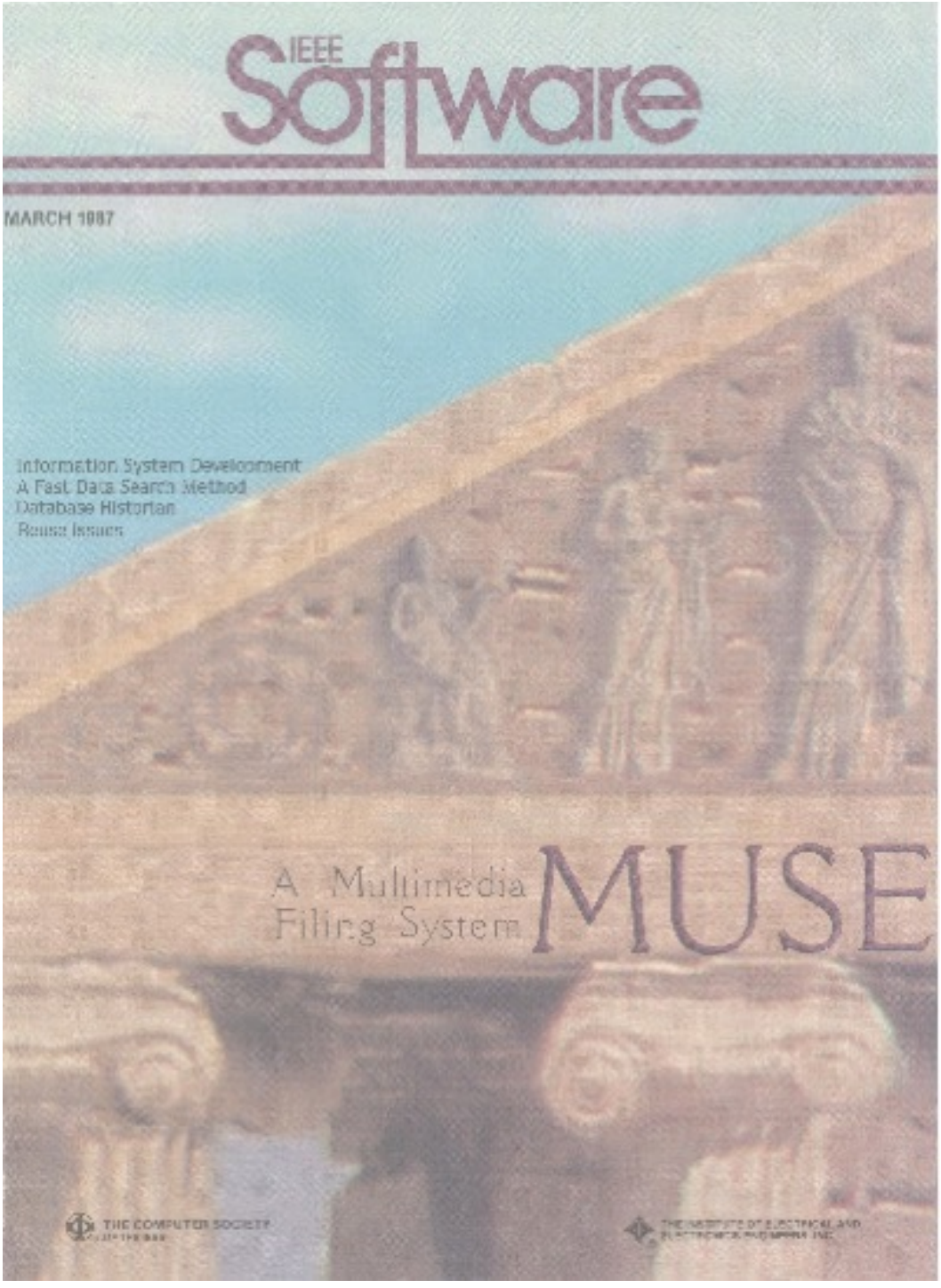
[37] DOI: 10.1109/MS.1987.229792

“**ABC** is being designed and implemented with an **integrated programming environment**. … The main design objectives … : **simplicity**, suitability for **interactive use**, and **availability of tools** for structured programming.”

---

*Steven Pemberton, **An Alternative Simple Language and Environment for PCs**, IEEE Software, January 1987.*[38]

[38] DOI: 10.1109/MS.1987.229797


IEEE
Software
MARCH 1987
Information System Development
A Fast Data Search Method
Database Historian
Reuse Issues
A Multimedia
Filing System
MUSE
THE COMPUTER SOCIETY

“New technology is changing the way we store documents. This experimental system features flexible **document retrieval**, a **distributed architecture**, and the capacity to store many very **large documents**.”

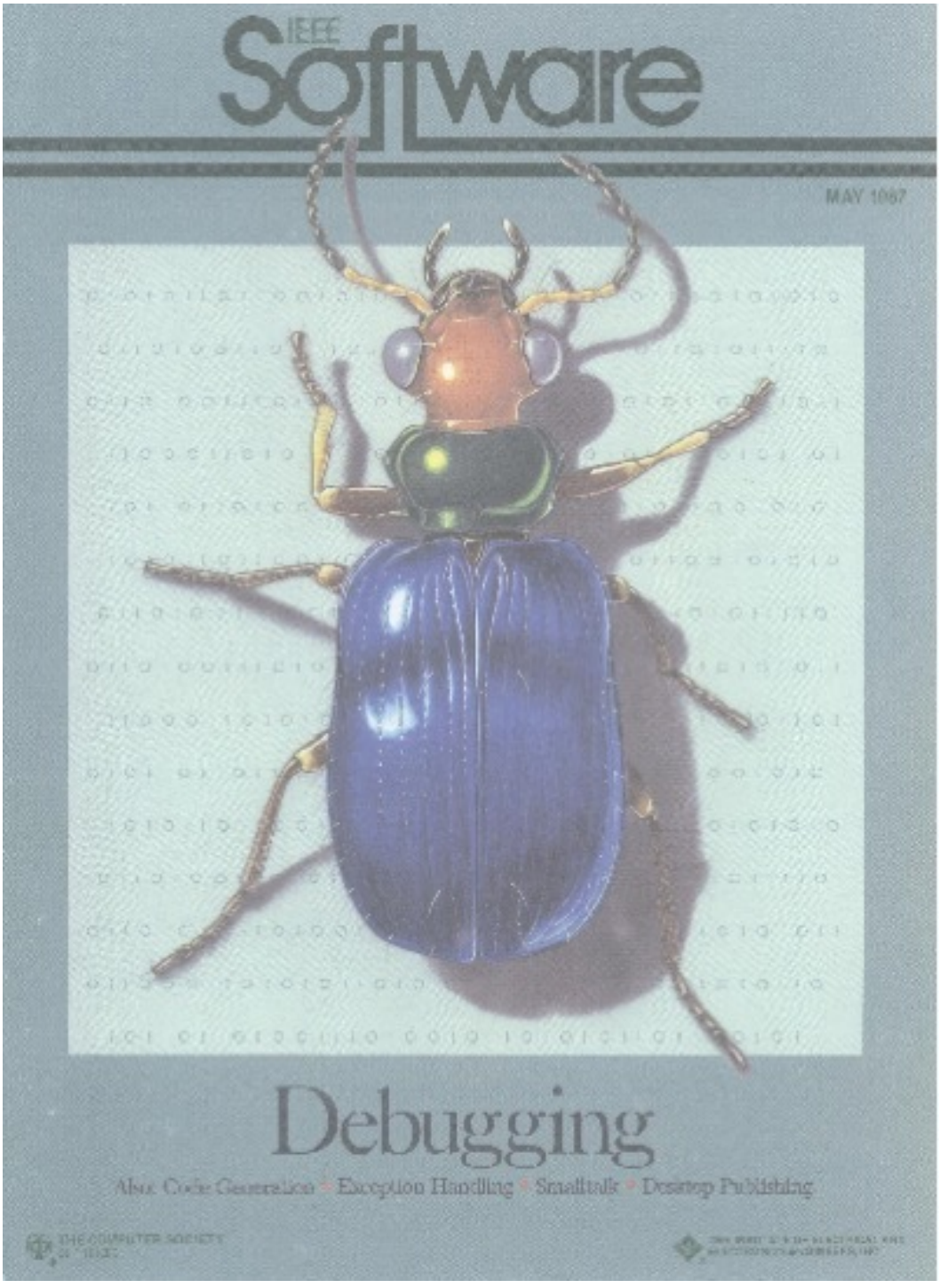
Simon Gibbs, Dennis Tsichritzis, Akis Fitas, Dimitri Konstantas, Yiannis Yeorgaroudakis, ***Muse: A Multimedia File System****, IEEE Software, March 1987.*[39]

[39] DOI: 10.1109/MS.1987.230090

“Simply being more organized will not make **the reuse problem** go away. The issues are technical, not managerial. The answers lie in **object-oriented design**.”

Bertrand Meyer, ***Reusability: The Case for Object-Oriented Design****, IEEE Software, March 1987.*[40]

[40] DOI: 10.1109/MS.1987.230097


IEEE
Software
MAY 1987
Debugging
Also: Code Generation • Exception Handling • Smalltalk • Desktop Publishing

---

“Testing Ada programs is easier with this **visual debugger** that graphically depicts what the program as doing - and how it is being done.”

Sadahiro Isoda, Yuji Ono, Takao Shimomura, ***VIPS: A Visual Debugger****, IEEE Software, May 1987.*[41]

[41] DOI: 10.1109/MS.1987.230394

“**No one likes to debug programs**, and there is no way to automate the task.”

---

*R.E. Seviora, **Knowledge-Based Program Debugging Systems**, IEEE Software, May 1987.*[42]

[42] DOI: 10.1109/MS.1987.230396

“The term debugging was first applied to **a hardware bug** - **a moth** in the circuitry of Mark II”

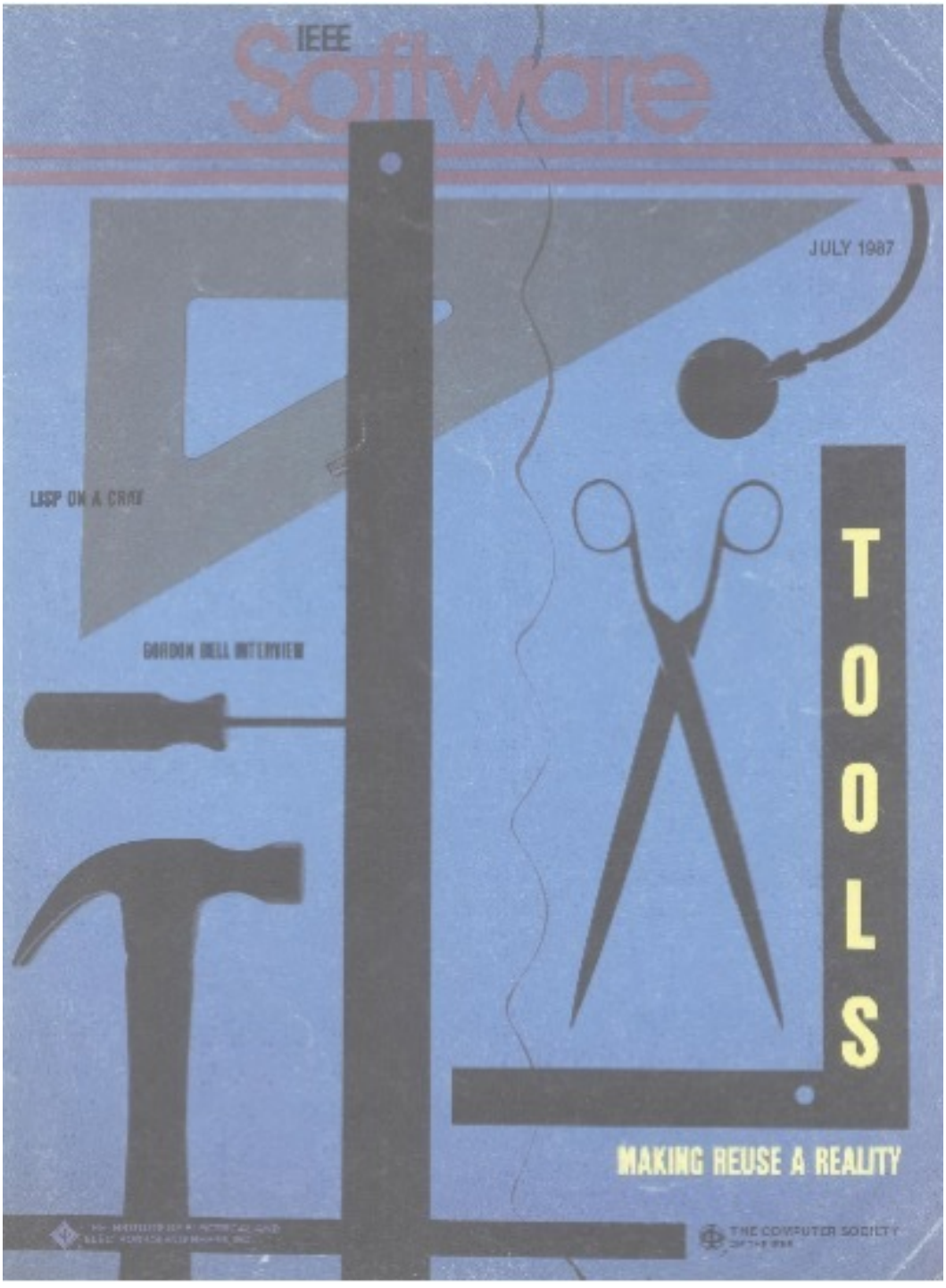
R.E. Seviora, ***Knowledge-Based Program Debugging Systems***, *IEEE Software*, *May 1987.*[43]

[43] DOI: 10.1109/MS.1987.230396

“Smalltalk promotes **fearless** programming.”

Jim Diederich, Jack Milton, ***Experimental Prototyping in Smalltalk****, IEEE Software, May 1987.*[44]

[44] DOI: 10.1109/MS.1987.230707


IEEE
Software
JULY 1987
LISP ON A CRAY
GORDON BELL INTERVIEW
TOOLS
MAKING REUSE A REALITY
THE COMPUTER SOCIETY

“People are leery about buying a **used car** for many of the same reasons programmers are reluctant to **reuse** someone else’s work.”

Will Tracz, ***Reusability Comes of Age***, *IEEE Software*, *July 1987.*[45]

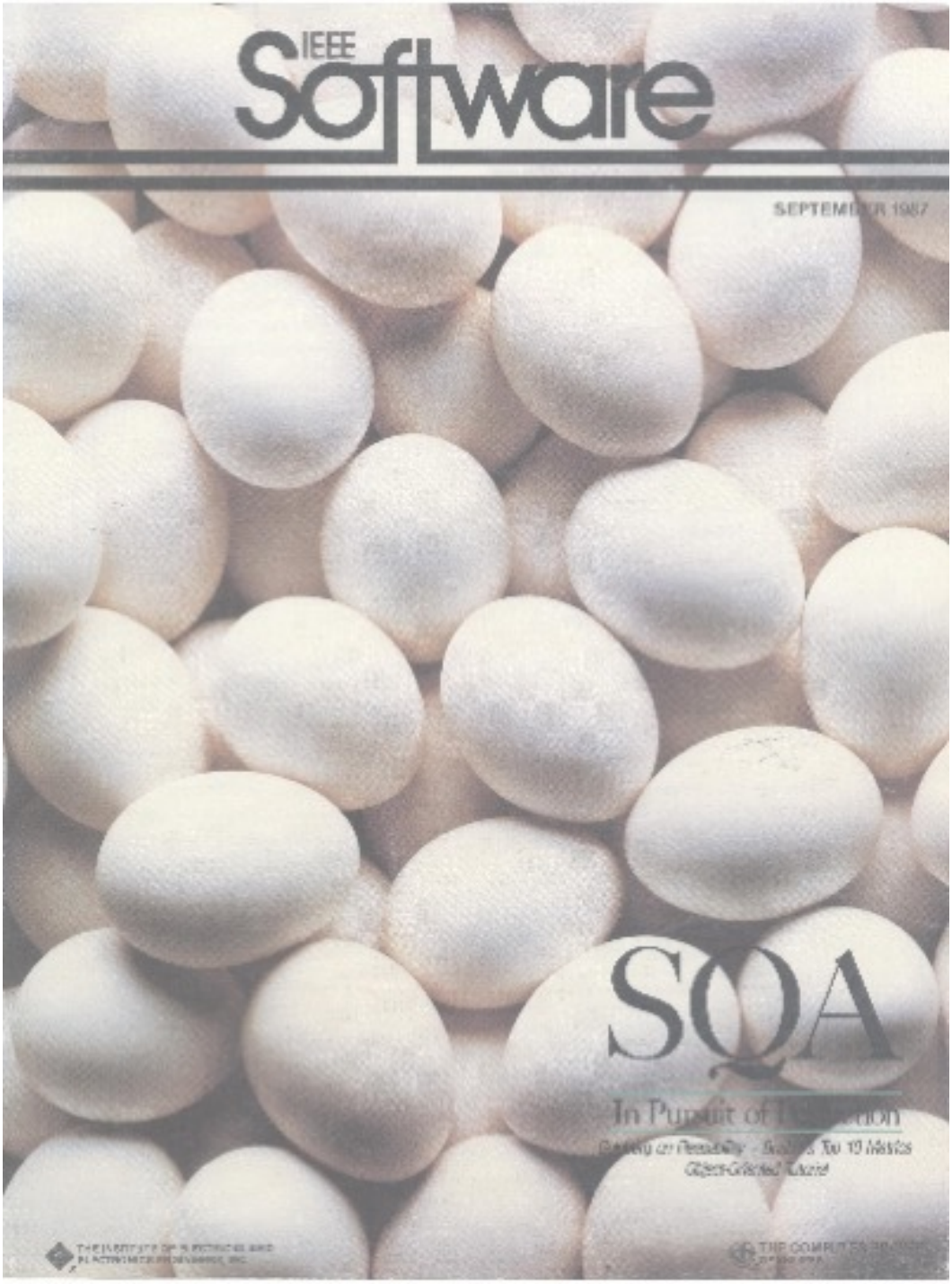
[45] DOI: 10.1109/MS.1987.231056

“An experiment asked programmers untrained in reuse to **evaluate component reusability**. They did poorly.”

---

*D.W. Embley, S.N. Woodfield, D.T. Scott, **Can Programmers Reuse Software?**, IEEE Software, July 1987.*[46]

[46] DOI: 10.1109/MS.1987.231064


IEEE
Software
SEPTEMBER 1987
SQA
In Pursuit of
Object-Oriented

“Software quality can be engineered under **statistical quality control** and delivered with better quality.”

---

*R.C. Linger, M. Dyer, H.D. Mills, **Cleanroom Software Engineering**, IEEE Software, September 1987.*[47]

[47] DOI: 10.1109/MS.1987.231413

“SQA will evolve into a broader software quality technology, shifting from a passive process to an active one, from **fault detection** to **fault avoidance**.”

F.S. LaMonica, J.P. Cavano, ***Quality Assurance in Future Development Environments****, IEEE Software, September 1987.*[48]

[48] DOI: 10.1109/MS.1987.231415

“An effective way to improve software quality is to **set measurable goals** and then manage your projects to achieve those goals. Hewlett-Packard has developed some methods to do just that.”

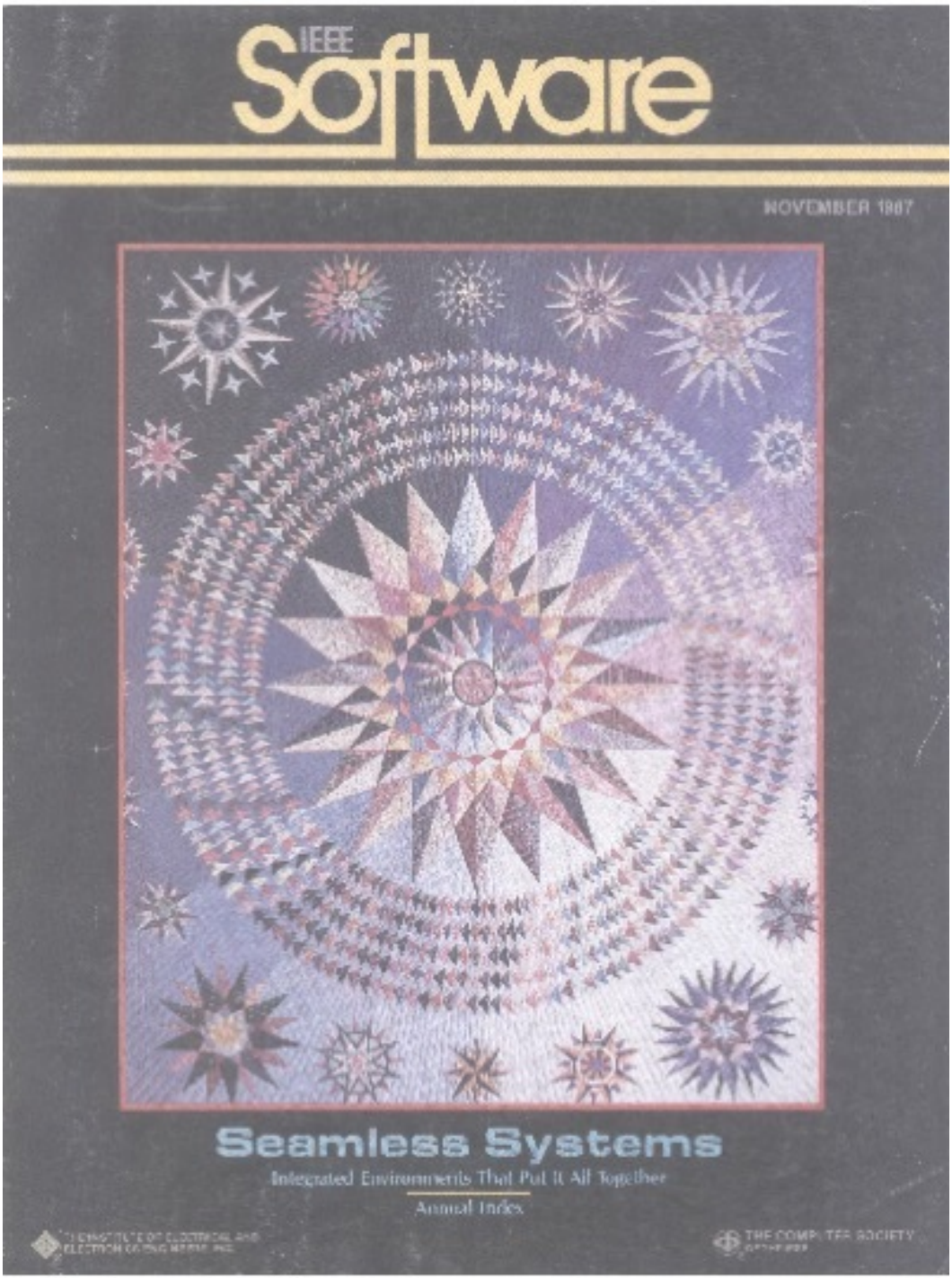
R.B. Grady, ***Measuring and Managing Software Maintenance***, *IEEE Software*, *September 1987.*[49]

[49] DOI: 10.1109/MS.1987.231417

“**Maintenance** plays a vital role in **protecting quality** as a system evolves.”

J.J. Buck, J.S. Collofello, ***Software Quality Assurance for Maintenance****, IEEE Software, September 1987.*[50]

[50] DOI: 10.1109/MS.1987.231418


IEEE
Software
NOVEMBER 1987
Seamless Systems
Integrated Environments That Put It All Together
Annual Index
THE COMPUTER SOCIETY

“How do you keep teams of programmers informed of system changes without burying them in mail messages? Make the environment responsible for **propagating changes**.”

---

*S.M. Kaplan, J. Micallef, G.E. Kaiser, **Multiuser, Distributed Language-Based Environments**, IEEE Software, November 1987.*[51]

[51] DOI: 10.1109/MS.1987.232092

“A new generation of operating system, based on **extended databases**, will supplant the original phase-sequencing and current **pipelining program composition mechanisms**.”

R.M. Baizer, ***Living in the Next-Generation Operating System***, *IEEE Software, November 1987.*[52]

[52] DOI: 10.1109/MS.1987.232097

1988


Software
JANUARY 1988
FOCUS ON PARALLEL PROGRAMMING
1988
H·A·W·A·I·I
CONFERENCE
Also: Comdex in Perspective • The Search for the Perfect Metric
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.
THE COMPUTER SOCIETY OF THE IEEE

“**Parallel programming** challenges software professionals to rethink old approaches and difficult - often controversial - choices.”

Shreekant S. Thakkar, ***Guest Editor’s Introduction: Parallel Programming—Issues and Questions***, *IEEE Software*, *January 1988.*[53]

[53] DOI: 10.1109/MS.1988.10003

“**Parafunctional programming** is based on the premise that the what (specification) and the how (implementation) are separately identifiable and maintainable system components.”

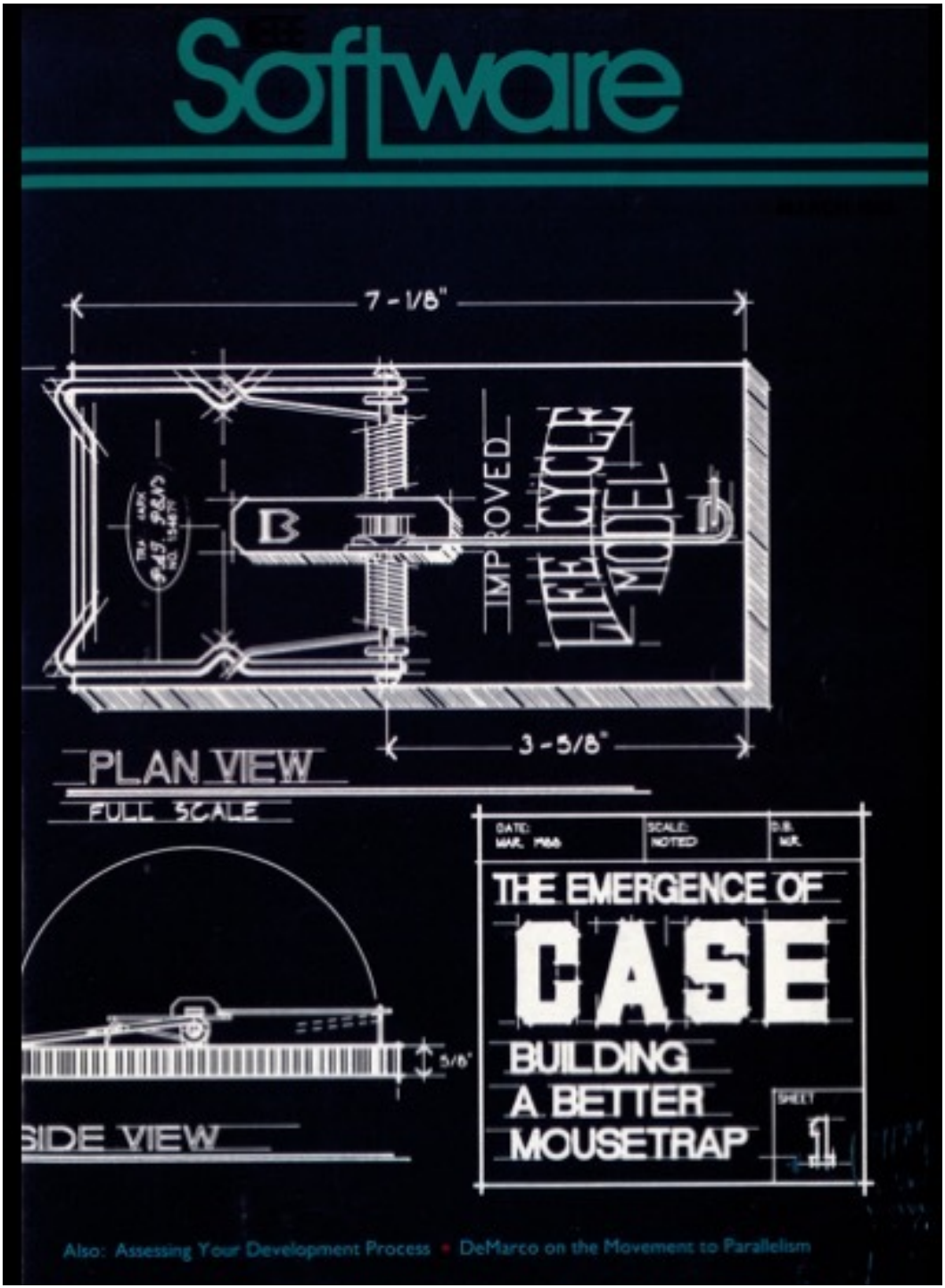
Paul Hudak, ***Exploring Parafunctional Programming: Separating the What from the How****, IEEE Software, January 1988.*[54]

[54] DOI: 10.1109/52.1994

“The IOGen **static-analysis tool** … uses a technique based on symbolic execution and produces a set of I/O pairs that represent execution paths through a program.”

Joyce R. Jenkins, Timothy E. Lindquist, ***Test-Case Generation with IOGen****, IEEE Software, January 1988.*[55]

[55] DOI: 10.1109/52.1996


Software
7 - 1/8"
IMPROVED
LIFE CYCLE
MODEL
3 - 5/8"
PLAN VIEW
FULL SCALE
5/8"
SIDE VIEW
DATE:
MAR. 1988
SCALE:
NOTED
D.B.
MR.
THE EMERGENCE OF
CASE
BUILDING
A BETTER
MOUSETRAP
SHEET
1
Also: Assessing Your Development Process • DeMarco on the Movement to Parallelism

---

“Mote than a decade after its introduction, **CASE** is emerging as a real-world technology whose **promises** are being fulfilled.”

Elliot J. Chikofsky, ***Guest Editor’s Introduction: Software Technology People Can Really Use****, IEEE Software, March 1988.*[56]

[56] DOI: 10.1109/MS.1988.10019

“Current computer-aided-software engineering (CASE) tools have several **inherent limitations** that reduce the productivity gains they can achieve … methodology constraints, administration difficulties, documentation inadequacies, and graphic-artist requirement.”

Charles F. Martin, ***Second-Generation CASE Tools: A Challenge to Vendors****, IEEE Software, March 1988.*[57]

[57] DOI: 10.1109/52.2010

“Today tools help systems analysts, so **why aren’t they widely used**? “

Charles F. Martin, ***Second-Generation CASE Tools: A Challenge to Vendors****, IEEE Software, March 1988.*[58]

[58] DOI: 10.1109/52.2010

"A **software-process maturity framework** … has been developed to provide the US Department of Defense with a means to characterize the capabilities of software-development organizations. This software-development process-maturity model reasonably represents the actual ways in which software-development organizations improve. It provides a framework for assessing these organizations and identifying the priority areas for immediate improvement. It also helps identify those places where advanced technology can be most valuable in improving the software-development process."

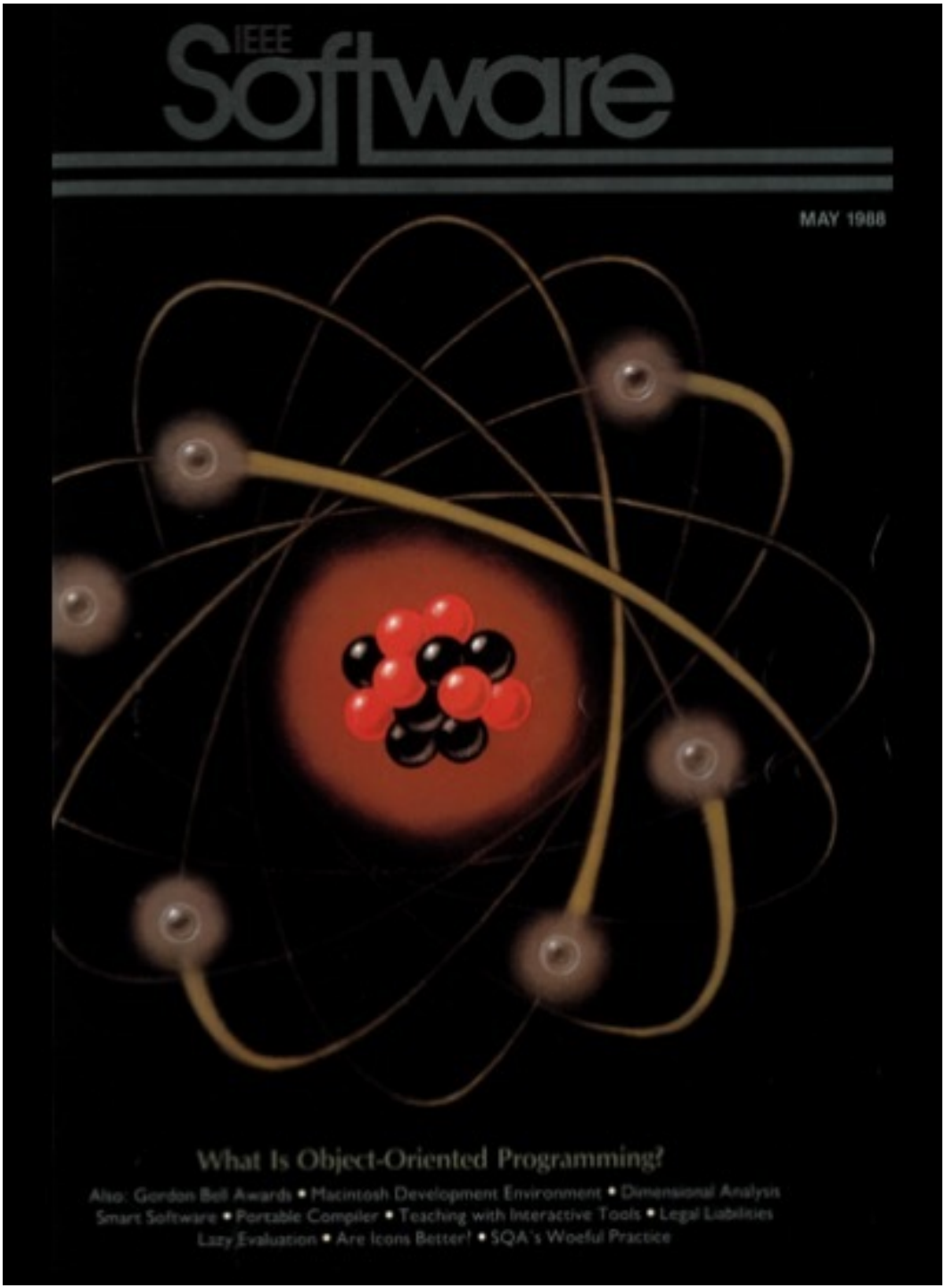
Watts S. Humphrey, ***Characterizing the Software Process: A Maturity Framework***, *IEEE Software*, *March 1988.*[59]

[59] DOI: 10.1109/52.2014

"There are several applications where a **universal-relation interface** to an existing database-management system is essential. The most obvious example is **natural-language** interface - indeed, it is hard to see how a natural-language interface could reliably use anything else, since it is unreasonable to make the user talk in terms of the **database's logical structure**."

Moshe Y. Vardi, ***The Universal-Relation Data Model for Logical Independence****, IEEE Software, March 1988.*[60]

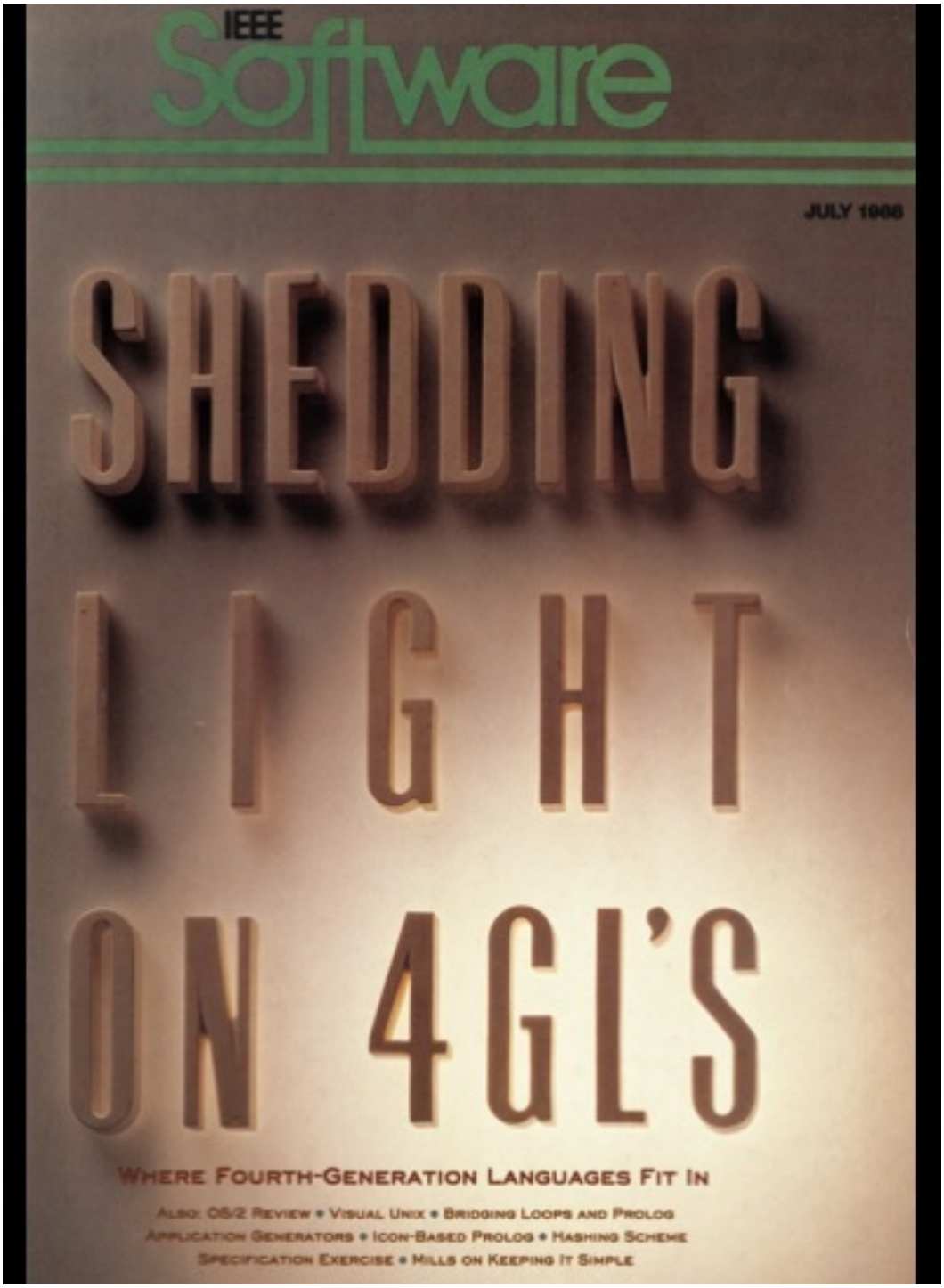
[60] DOI: 10.1109/52.2015


IEEE
Software
MAY 1988
What Is Object-Oriented Programming?
Also: Gordon Bell Awards • Macintosh Development Environment • Dimensional Analysis
Smart Software • Portable Compiler • Teaching with Interactive Tools • Legal Liabilities
Lazy Evaluation • Are Icons Better? • SQA's Woeful Practice

“The meaning of the term ‘**object oriented**’ is examined in the context of the general-purpose programming language C++. This choice is made partly to **introduce C++** and partly because C++ is one of the few languages that supports data abstraction, object-oriented programming, and traditional programming techniques. … four paradigms are examined: **procedural**, **data hiding**, **data abstraction**, and **object-oriented** programming.”

Bjarne Stroustrup, ***What Is Object-Oriented Programming?****, IEEE Software, May 1988.*[61]

[61] DOI: 10.1109/52.2020


IEEE
Software
JULY 1988
SHEDDING
LIGHT
ON 4GL'S
WHERE FOURTH-GENERATION LANGUAGES FIT IN
ALSO: OS/2 REVIEW • VISUAL UNIX • BRIDGING LOOPS AND PROLOG
APPLICATION GENERATORS • ICON-BASED PROLOG • HASHING SCHEME
SPECIFICATION EXERCISE • MILLS ON KEEPING IT SIMPLE

“Even though the code sizes were smaller with both fourth-generation tools, **Cobol was clearly superior** in performance.”

Paul J. Jalics, Santosh K. Misra, ***Third-Generation Versus Fourth-Generation Software Development****, IEEE Software, July 1988.*[62]

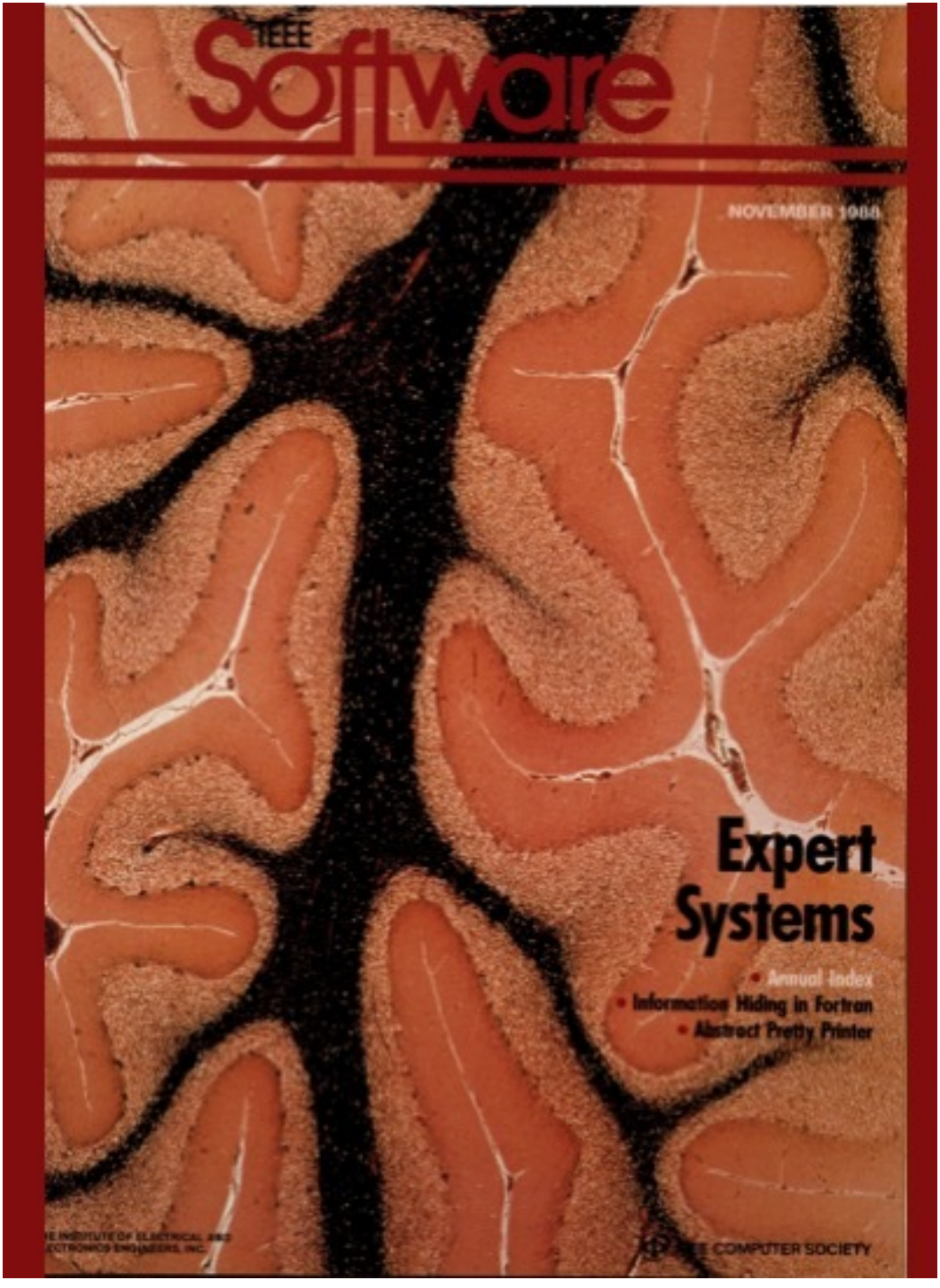
[62] DOI: 10.1109/52.17797


IEEE
Software
SEPTEMBER 1988
Real-Time
Systems
ALSO:
• COMPARING
I2 PARALLEL
FORTRAN DIALECTS
• USING PATENTS
TO PROTECT SOFTWARE
• PROGRAMMING A HYPERCUBE
MULTICOMPUTER EFFICIENTLY
• ACHIEVING LOW ERROR RATES
• STANDARDIZING USER INTERFACES
THE COMPUTER SOCIETY

“**Shared memory** requires carefully designed concurrency control, but the traditional approach, which is to embed the entire **allocate-release** implementation code in critical sections, is unsuitable for real-time applications because it results in excessively high response time.”

Ray Ford, ***Concurrent Algorithms for Real-Time Memory Management****, IEEE Software, September 1988.*[63]

[63] DOI: 10.1109/52.7940


IEEE Software
NOVEMBER 1988
Expert Systems
• Annual Index
• Information Hiding in Fortran
• Abstract Pretty Printer
COMPUTER SOCIETY

“**Expert systems** attempt to clone an expert’s problem-solving behavior in a particular knowledge-intensive domain. An expert’s domain knowledge encompasses both the **facts** that apply to a particular area and the **knowledge of how** and when to use these facts to solve a problem in that domain. “

Murat M. Tanik, Raymond T. Yeh, ***Guest Editors’ Introduction: Expert Systems****, IEEE Software, November 1988.*[64]

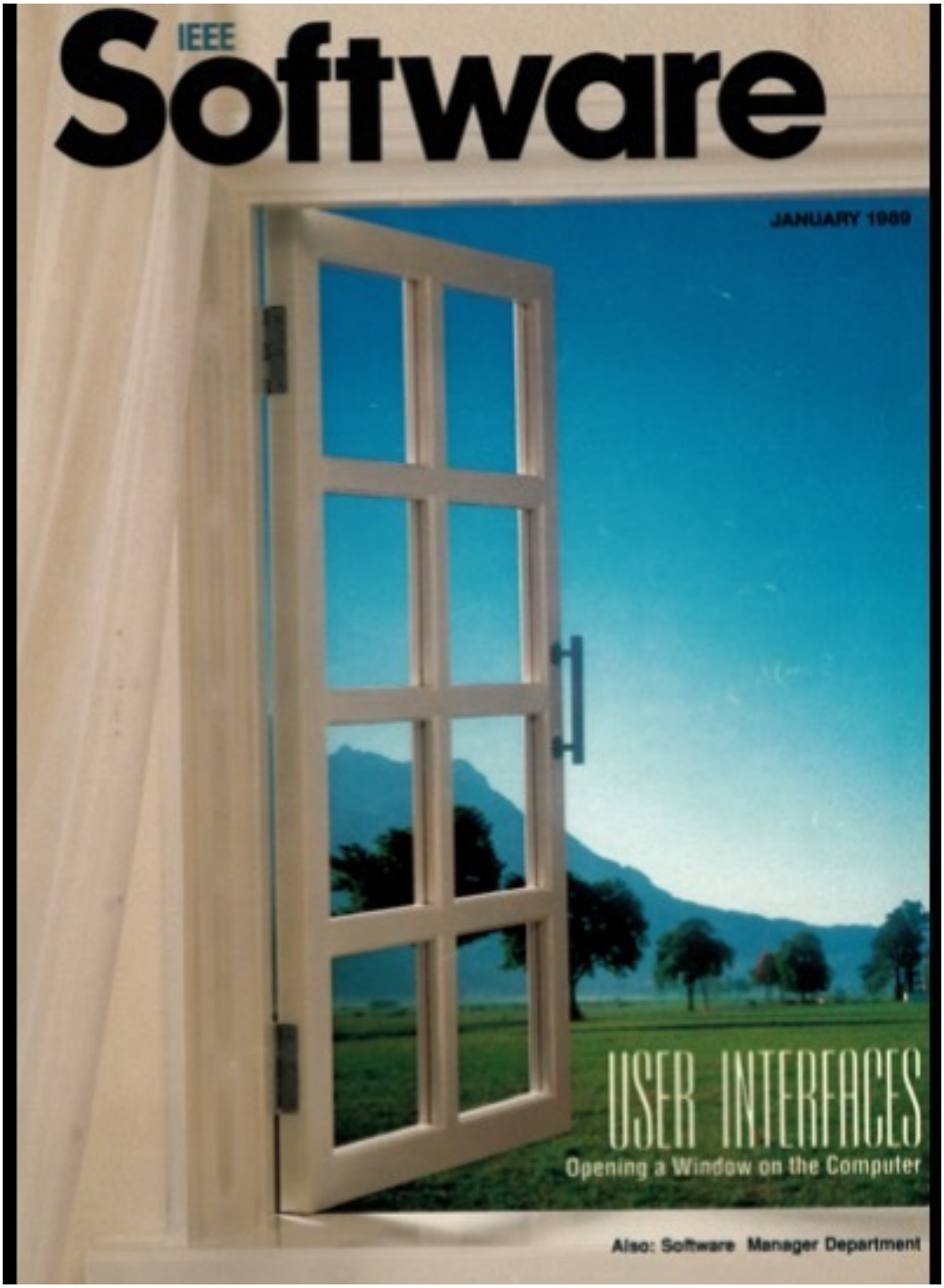
[64] DOI: 10.1109/MS.1988.10052

“It is far better to succeed by design than by default or chance. **Usability testing** tips the balance in favor of success and **reduces the risks** associated with launching a new system.”

Kathleen Potosnak, ***Recipe for a Usability Test***, *IEEE Software*, *November 1988.*[65]

[65] DOI: 10.1109/MS.1988.10054

1989


IEEE
Software
JANUARY 1989
USER INTERFACES
Opening a Window on the Computer
Also: Software Manager Department

“An **interactive system** - one with a human-computer interface - is not judged solely on its ability to compute. It is also judged on its **ability to communicate**. In fact, if users cannot communicate effectively with an interactive system, its computational ability may be inaccessible. “

Deborah Hix, ***Guest Editor’s Introduction: User Interfaces—Opening a Window on the Computer****, IEEE Software, January 1989.*[66]

[66] DOI: 10.1109/MS.1989.10004

“An overview is given of **user-interface development systems** (UIDS). … The three types are **language-based**, **graphical**, and **automatic creation interfaces**.”

Brad A. Myers, ***User-Interface Tools: Introduction and Survey***, *IEEE Software*, *January 1989.*[67]

[67] DOI: 10.1109/52.16898


IEEE
Software
MARCH 1989
THE FAR EAST
IEEE COMPUTER SOCIETY

"**Japan's Sigma** (Software Industrialized Generator and Maintenance Aids) … consists of the Sigma Center, Sigma network, and Sigma user sites. The Sigma Center will help users who are constructing development environments of programs using those environments. It will provide database services, demonstration services, and part of the network service."

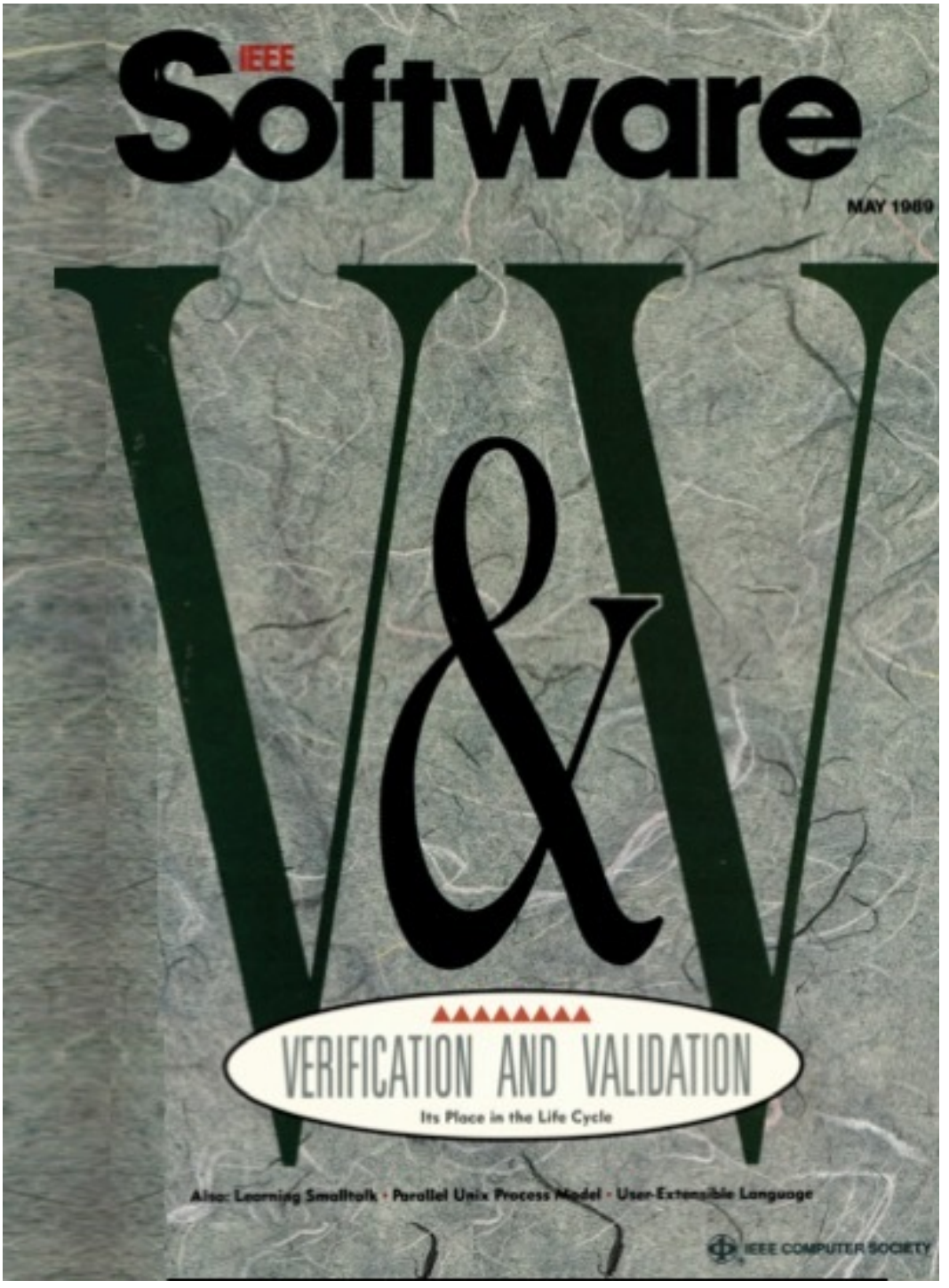
Noboru Akima, Fusatake Ooi, ***Industrializing Software Development: A Japanese Approach****, IEEE Software, March 1989.*[68]

[68] DOI: 10.1109/52.23125

“With strong government backing and direction, **Singapore** seeks to develop information technology as a growth industry and use it to promote the nation’s global competitiveness.”

Tahn Joo Chin, Kai Yuen Wang, ***Software Technology Development in Singapore****, IEEE Software, March 1989.*[69]

[69] DOI: 10.1109/52.23132


IEEE
Software
MAY 1989
V&V
VERIFICATION AND VALIDATION
Its Place in the Life Cycle
Also: Learning Smalltalk · Parallel Unix Process Model · User-Extensible Language
IEEE COMPUTER SOCIETY

“Properly applied throughout the life cycle, **verification and validation** can result in higher quality, more reliable programms.”

Dolores R. Wallace, Roger U. Fujii, ***Software Verification and Validation: An Overview****, IEEE Software, May 1989.*[70]

[70] DOI: 10.1109/52.28119

“**System testing** becomess guesswork - unless you set it up to apply **statistical analysis**. Then you can focus attention on the software’s use instead of its structure.”

---

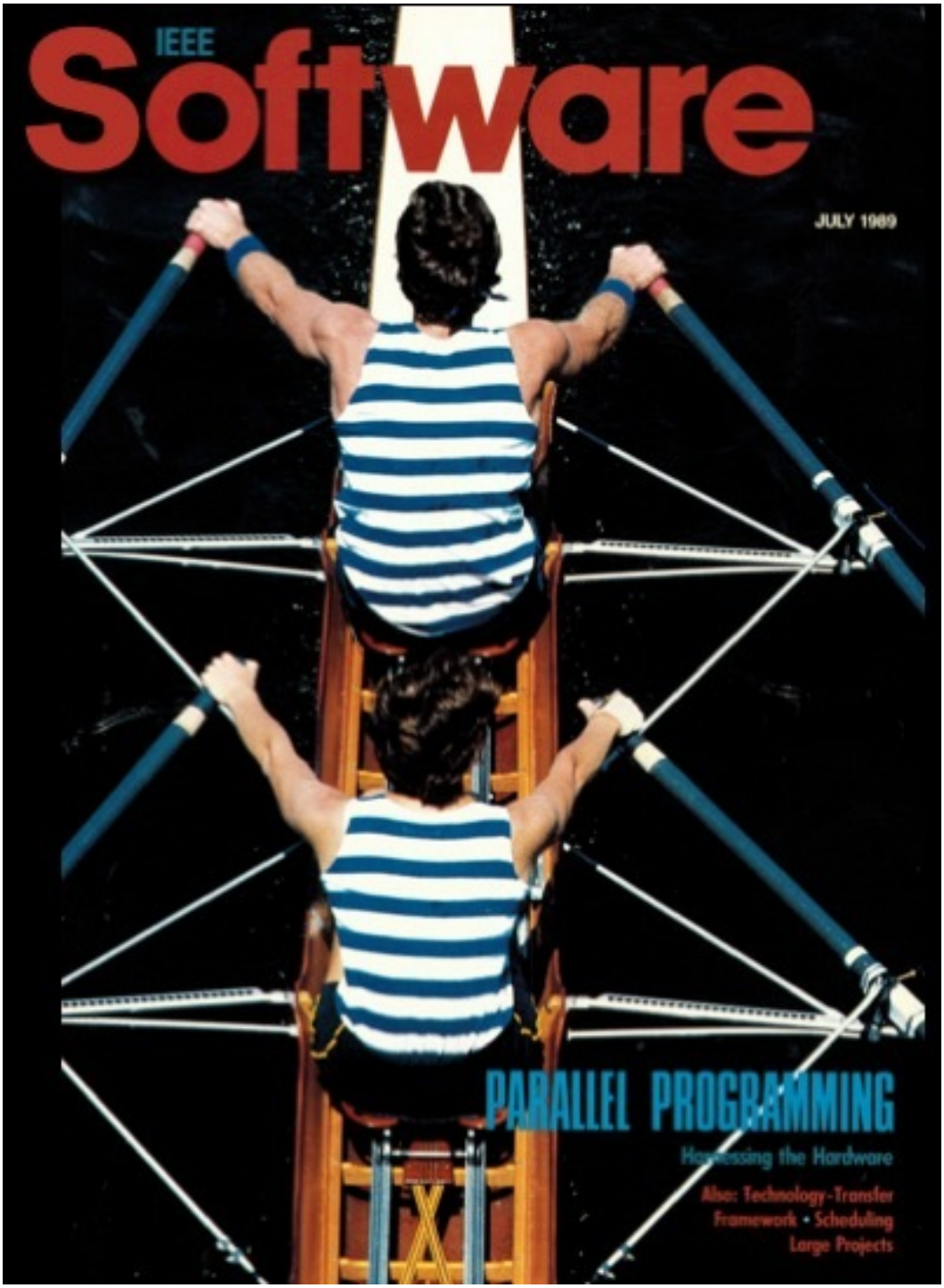
*A. Frank Ackerman, John D. Musa, **Quantifying Software Validation: When to Stop Testing?**, IEEE Software, May 1989.*[71]

[71] DOI: 10.1109/52.28120

“**Inspections** can detect and eliminate faults **more cheaply than testing**.”

A. Frank Ackerman, Lynne S. Buchwald, Frank H. Lewski, ***Software Inspections: An Effective Verification Process****, IEEE Software, May 1989.*[72]

[72] DOI: 10.1109/52.28121


IEEE
Software
JULY 1989
PARALLEL PROGRAMMING
Harnessing the Hardware
Also: Technology-Transfer
Framework • Scheduling
Large Projects

“The commercial availability of these machines poses a challenge to the software industry. What is the best way to **harness the power**?”

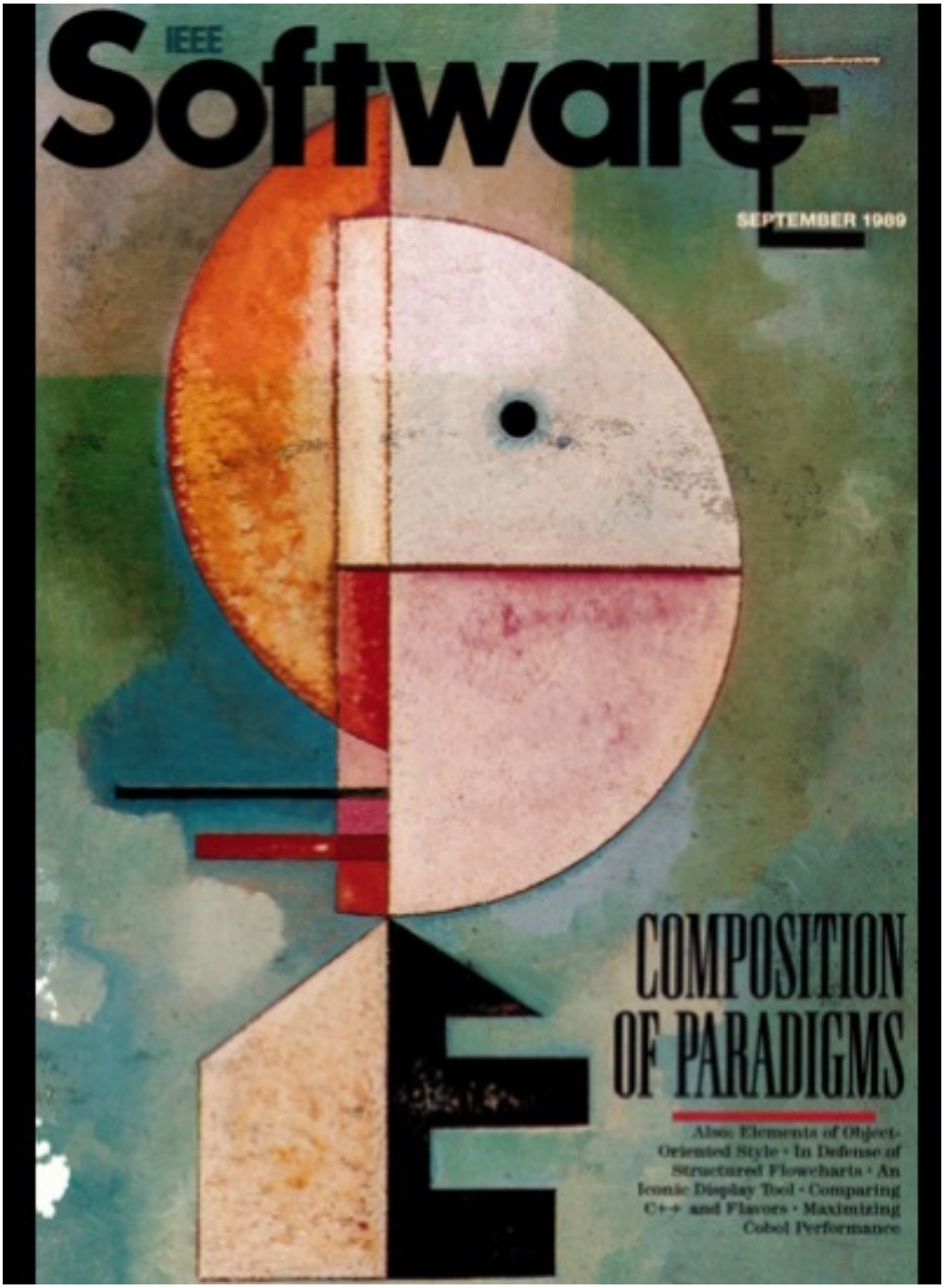
Shreekant Thakkar, ***Guest Editor’s Introduction: Parallel Programming—Harnessing the Hardware****, IEEE Software, July 1989.*[73]

[73] DOI: 10.1109/MS.1989.10037

"A strong common theme among the managers interviewed was the person's ability to **communicate** with both peers and managers … Most of the managers are looking for someone who will be **a good team player** - someone who can work for the good of the group and apply his skills and talents to **assist collective goals**. … I look for someone I can motivate and who **wants to be motivated**…"

Elliot Chikofsky, ***Looking For the Best Software Engineers***, *IEEE Software, July 1989.*[74]

[74] DOI: https://doi.ieeecomputersociety.org/10.1109/52.31661


IEEE
Software
SEPTEMBER 1989
COMPOSITION OF PARADIGMS
Abss: Elements of Object-Oriented Style • In Defense of Structured Flowcharts • An Iconic Display Tool • Comparing C++ and Flavors • Maximizing Cobol Performance

---

“**Multiparadigm programming** makes it possible to match the paradigm to the problem.”

Pamela Zave, ***A Compositional Approach to Multiparadigm Programming***, *IEEE Software*, *September 1989.*[75]

[75] DOI: 10.1109/52.35586

"The language-independent **Law of Demeter** … encodes the ideas of **encapsulation and modularity** in an easy-to-follow form for object-oriented programmers … The law was developed during the design and implementation of **the Demeter system**, which provides a high-level interface to class-based, object-oriented systems."

Ian M. Holland, Karl J. Lieberherr, ***Assuring Good Style for Object-Oriented Programs****, IEEE Software, September 1989.*[76]

[76] DOI: 10.1109/52.35588


IEEE
Software
NOVEMBER 1989
E S P R I T
Also: OS/2 Programming Tools • Software Policy • Annual Index

---

“The European Strategic Program for Research and Information Technology **ESPRIT** has had a profound effect on industrial technology in Europe.”

Annie Kuntzmann-Combelles, ***Guest Editor’s Introduction: ESPRIT—Key Results of the First Phase****, IEEE Software, November 1989.*[77]

[77] DOI: 10.1109/MS.1989.10064

1990


IEEE
Software
JANUARY
1990
MAINTENANCE &
REVERSE ENGINEERING &
DESIGN RECOVERY
Also: Distributed Debugging • Cost/Schedule Balance • Pruning Functions • Change Agents

---

"**Reverse engineering** is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction."

---

*Elliot J. Chikofsky, James H. Cross II, **Reverse Engineering and Design Recovery: A Taxonomy**, IEEE Software, January 1990.*[78]

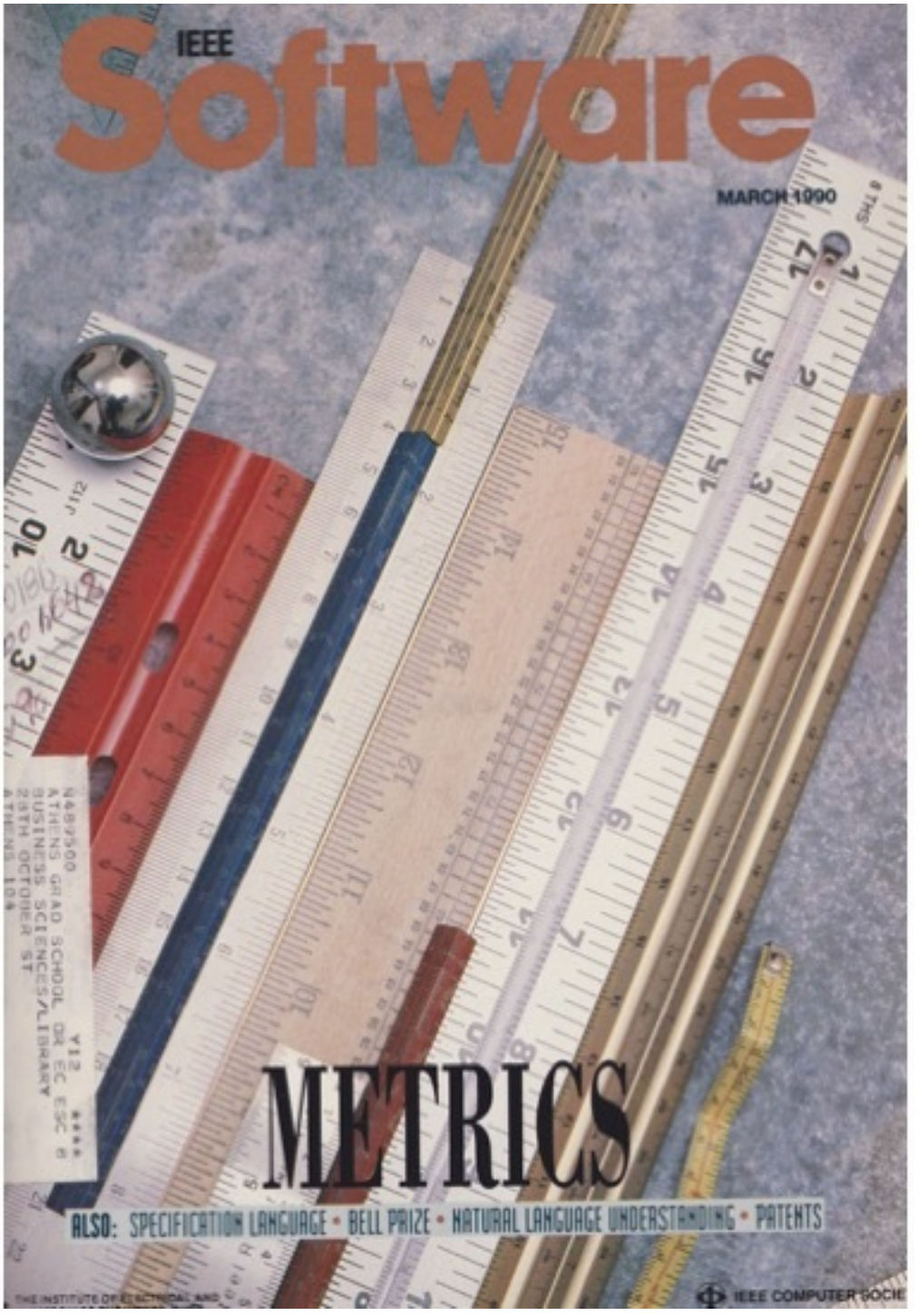
[78] DOI: 10.1109/52.43044

“**Restructuring** is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system’s external behavior (functionality and semantics). “

---

*Elliot J. Chikofsky, James H. Cross II, **Reverse Engineering and Design Recovery: A Taxonomy**, IEEE Software, January 1990.*[79]

[79] DOI: 10.1109/52.43044


IEEE
Software
MARCH 1990
METRICS
ALSO: SPECIFICATION LANGUAGE • BELL PRIZE • NATURAL LANGUAGE UNDERSTANDING • PATENTS
IEEE COMPUTER SOCIE

“You can’t control what you can’t measure. That fundamental reality underlies the importance of **software metrics**, despite the **controversy** that has surrounded them since **Maurice Halstead** put forth his idea of software science. Sketpics claim metrics are useless and expensive exercise in pointless data collection, while proponents argue they are valuable management and engineering tools.”

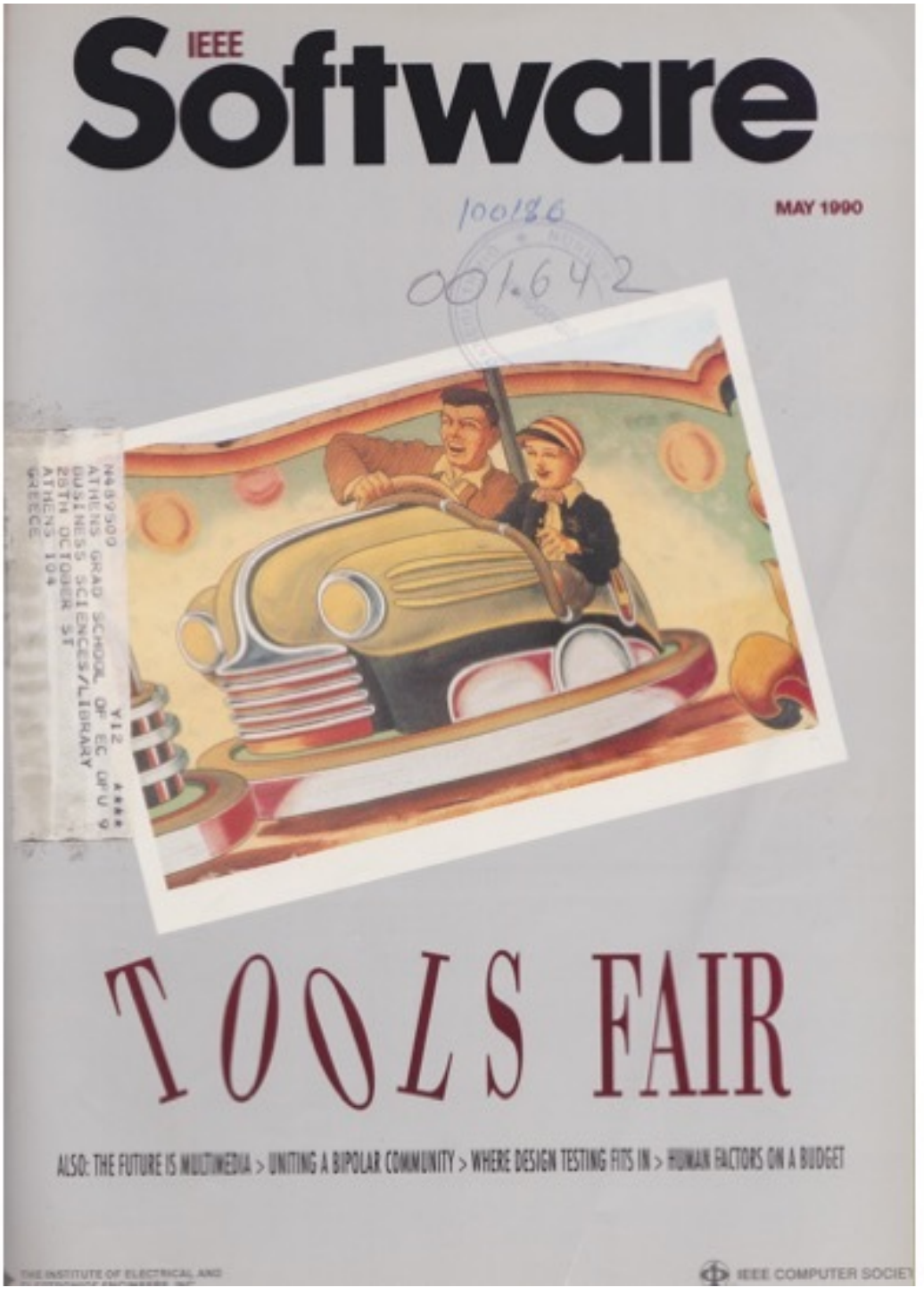
Peter B. Dyson, Harlan D. Mills, ***Guest Editors’ Introduction: Using Metrics to Quantify Development****, IEEE Software, March 1990.*[80]

[80] DOI: 10.1109/MS.1990.10016

“**Measurement** must be applied in individual experiments or case studies; … measurement can help continuously improve an organization’s state of the practice; … measurement requires automated support.”

H. Dieter Rombach, ***Design Measurement: Some Lessons Learned****, IEEE Software, March 1990.*[81]

[81] DOI: 10.1109/52.50770


IEEE
Software
MAY 1990
100186
001.642
N489300
ATHENS GRAD SCHOOL OF EC UFU 9
BUSINESS SCIENCES/LIBRARY
25TH OCTOBER ST
ATHENS 104
GREECE
TOOLS FAIR
ALSO: THE FUTURE IS MULTIMEDIA > UNITING A BIPOLAR COMMUNITY > WHERE DESIGN TESTING FITS IN > HUMAN FACTORS ON A BUDGET
IEEE COMPUTER SOCIETY

“**Practitioners** drift one way, **purists** the other. The purists must compromise.”

---

*Carl Chang, **Let’s Stop the Bipolar Drift**, IEEE Software, May 1990.*[82]

[82] DOI: 10.1109/MS.1990.10029

“**Tools** are divided into vertical and horizontal architectures. **Vertical tools** support specific activities in a single life-cycle phase, such as analysis, design, or testing. **Horizontal tools** support activities across the entire life cycle, such as project management and cost estimation.”

Paul W. Oman, Dennis B. Smith, ***Software Tools in Context****, IEEE Software, May 1990.*[83]

[83] DOI: 10.1109/52.55222

“**Performance tools** are a way of making systematic the work needed to carry out performance studies so that several studies can be carried out more easily and be compared in a consistent way.”

Kathleen D. Nichols, ***Performance Tools***, *IEEE Software*, *May 1990.*[84]

[84] DOI: 10.1109/52.55223

“Three approaches to **user-interface development**: **tool kits**, **user-interface management systems** (UIMS), and interactive **design tools**.”

---

*Ed Lee, **User-Interface Development Tools**, IEEE Software, May 1990.*[85]

[85] DOI: 10.1109/52.55225

“Some **testing tools** simulate the final execution environment as a way of expediting test execution, others automate the development of test plans, and still others collect performance data during execution.”

---

*Mike Lutz, **Testing Tools**, IEEE Software, May 1990.*[86]

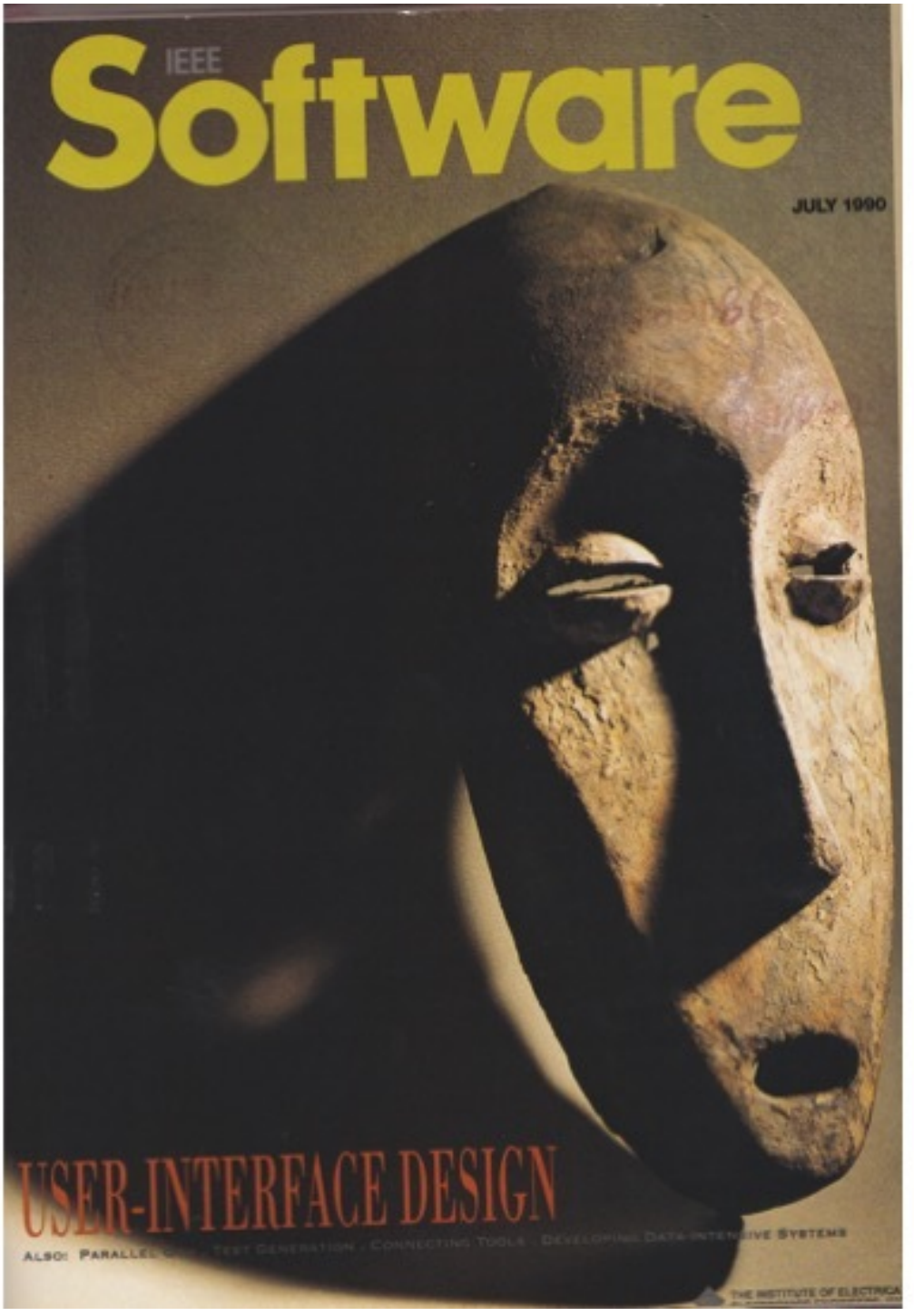
[86] DOI: 10.1109/52.55228

“**Code generators** … take a programmer’s inputs in the form of some abstraction, design, or direct interaction with the system and write out a source program that implements the details of the application…”

---

*Ted Lewis, **Code Generators**, IEEE Software, May 1990.*[87]

[87] DOI: 10.1109/52.55230


IEEE
Software
JULY 1990
USER-INTERFACE DESIGN
ALSO:
THE INSTITUTE OF ELECTRICA

“An environment for **creating user interfaces for embedded systems**, called the graphical specification system (GSS) … combines graphical and minimal low-level textual specification with a prototyping capability for rapid user-interface design and evaluation.”

---

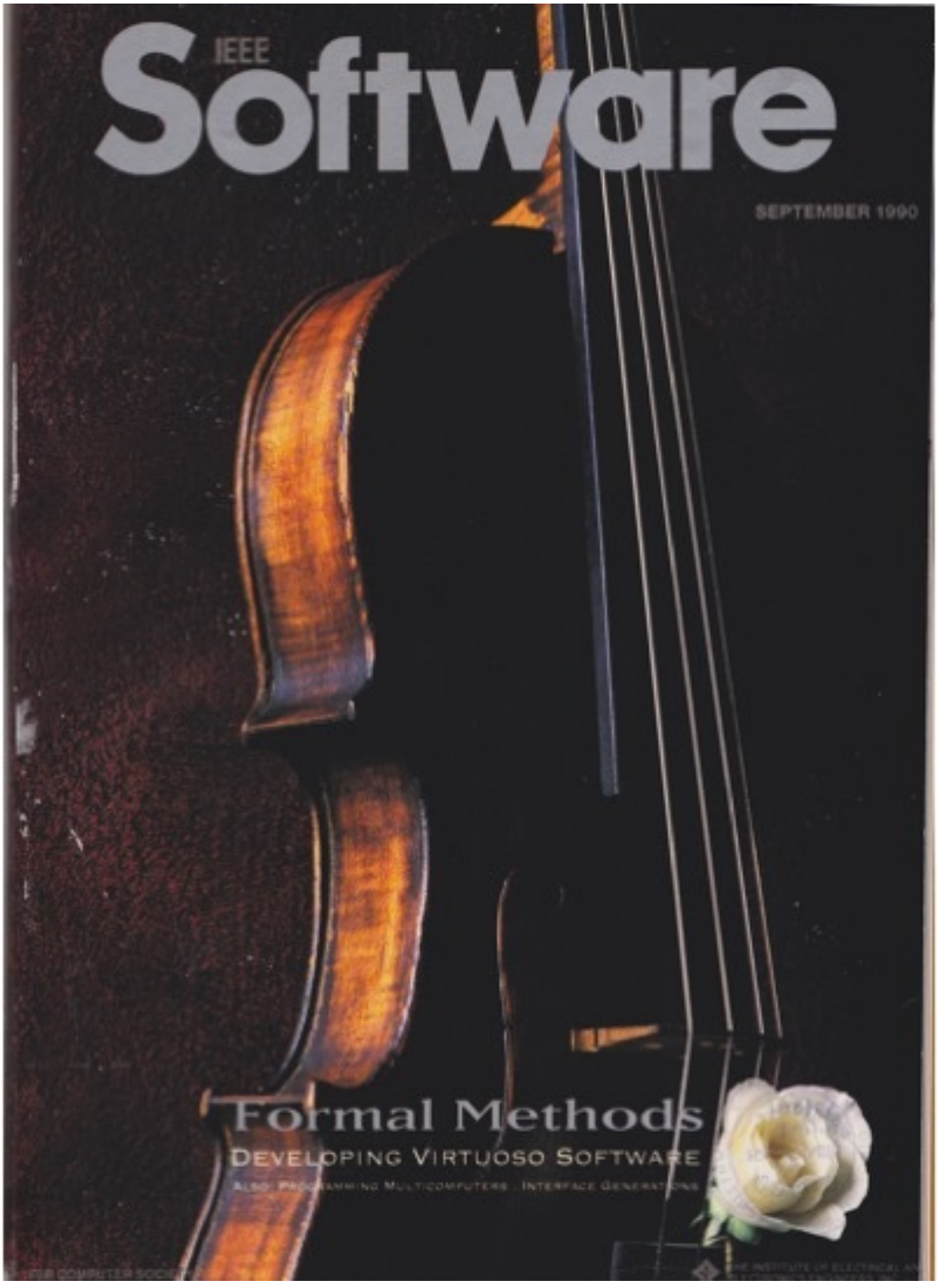
*Sallie Sheppard, Andrew Harbert, William Lively, **A Graphical Specification System for User-Interface Design**, IEEE Software, July 1990.*[88]

[88] DOI: 10.1109/52.56446

“The interpretive frame system (IFS), a tool for building application systems … separates high-level design and user-interface programming from domain-specific programming.”

Kiem-Phong Vo, ***IFS: A Tool to Build Application Systems****, IEEE Software, July 1990.*[89]

[89] DOI: 10.1109/52.56448


IEEE
Software
SEPTEMBER 1990
Formal Methods
DEVELOPING VIRTUOSO SOFTWARE

“Seven widely held conceptions about formal methods are challenged. These beliefs are variants of the following: formal methods can **guarantee** that software is perfect; they work by **proving** that the programs are correct; only highly critical systems **benefit** from their use; they involve **complex mathematics**; they **increase the cost** of development; they are **incomprehensible** to clients; and **nobody uses them** for real projects.”

---

*Anthony Hall, **Seven Myths of Formal Methods**, IEEE Software, September 1990.*[90]

[90] DOI: 10.1109/52.57887


IEEE
Software
NOVEMBER 1990
SOFTWARE
CHALLENGES
IEEE COMPUTER SOCIETY

"Five basic steps that the **software engineering profession** must take to become a true engineering discipline … are: understanding the **nature of expertise**, recognizing different ways to **get information**, encouraging **routine practice**, expecting professional **specializations**, and improving the **coupling between science and commercial practice**."

Mary Shaw, ***Prospects for an Engineering Discipline of Software****, IEEE Software, November 1990.*[91]

[91] DOI: 10.1109/52.60586

"**Cleanroom engineering** achieves intellectual control by applying rigorous, mathematics-based engineering practices, establishes an **errors-are-unacceptable attitude** and a **team responsibility for quality**, delegates development and testing responsibilities to separate teams, and certifies the software's mean time to failure through the application of statistical quality-control methods."

Harlan D. Mills, Richard H. Cobb, ***Engineering Software Under Statistical Quality Control****, IEEE Software, November 1990.*[92]

[92] DOI: 10.1109/52.60601

"**Schools** represent a challenge for the software industry. Although **diverting corporate resources to schools** may appear to be folly in the short run, in the long run it is a strategy for survival. Programmers and their organizations should devote their time and services to schools at the primary and secondary levels."

---

*Tom DeMarco, **Making a Difference in the Schools**, IEEE Software, November 1990.*[93]

[93] DOI: 10.1109/52.60592

1991


IEEE
Software
INDUSTRIAL DEVELOPMENT
Roadblocks to Integration
Reengineering Trial by Fire
Charting NSF Research
Shifting Sands at Condex

“Quantitative results based on a 1988 study of inspection of 2.5 million lines of high-level code at Bell-Northern Research … confirm that **code inspection** is still one of the most efficient ways to remove software defects.”

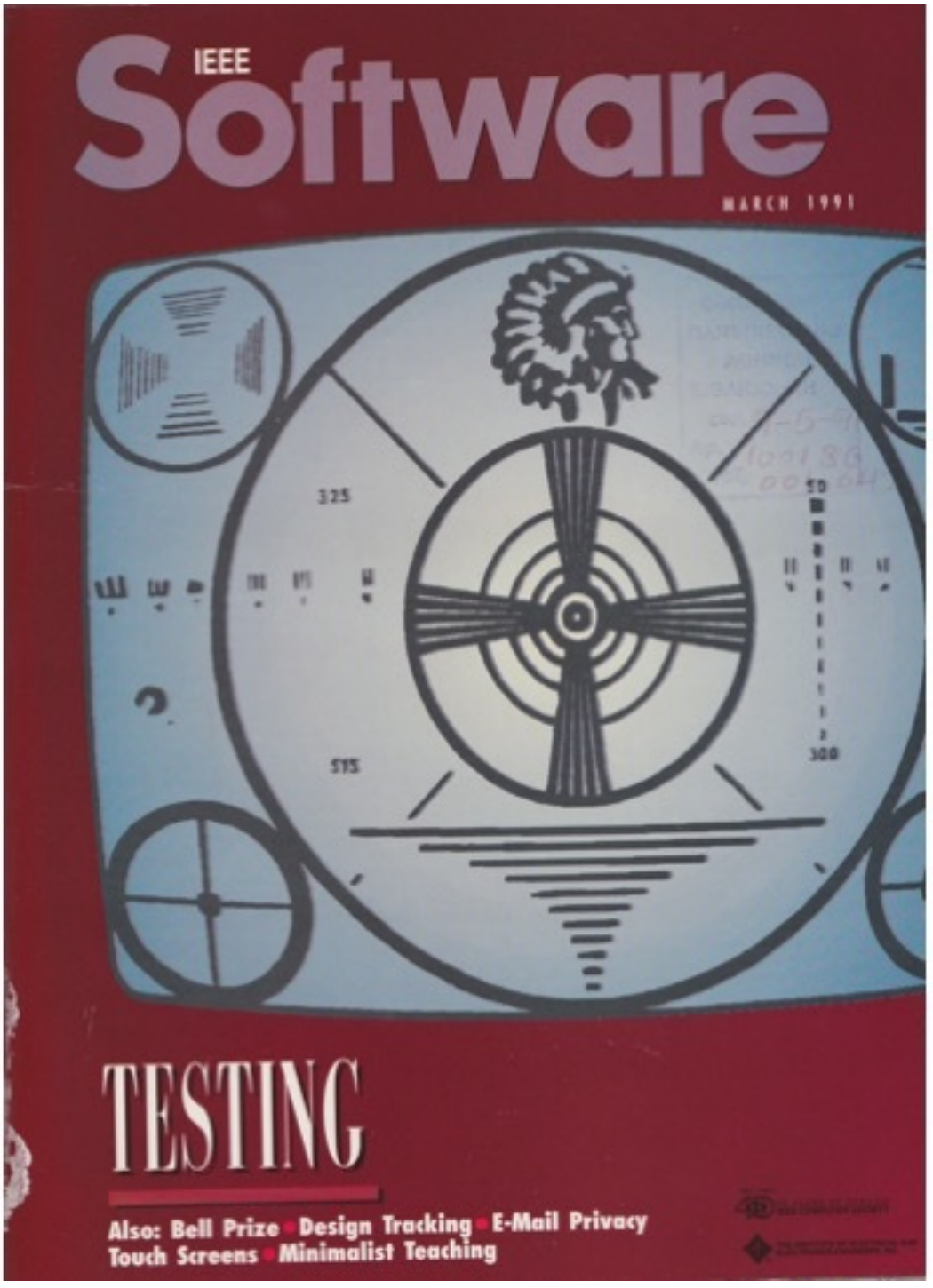
Glen W. Russell, ***Experience With Inspection in Ultralarge-Scale Development****, IEEE Software, January 1991.*[94]

[94] DOI: 10.1109/52.62929

"The emerging discipline of **software risk management** is … defined as an attempt to formalize the risk-oriented correlates of success into a readily applicable set of principles and practices. Its objectives are to identify, address, and eliminate risk items before they become either threats to successful software operation or major sources of software rework."

Barry W. Boehm, ***Software Risk Management: Principles and Practices****, IEEE Software, January 1991.*[95]

[95] DOI: 10.1109/52.62930


IEEE
Software
MARCH 1991
TESTING
Also: Bell Prize • Design Tracking • E-Mail Privacy
Touch Screens • Minimalist Teaching

“A model that patterns **software manufacturing** after **hardware manufacturing** … introduces a **testing** and **analysis station** between each development phase. At each station, the incoming product is tested and failure data are analyzed and compared with the quality criteria used. The decision is then made whether to proceed to the next phase or repair the software.”

---

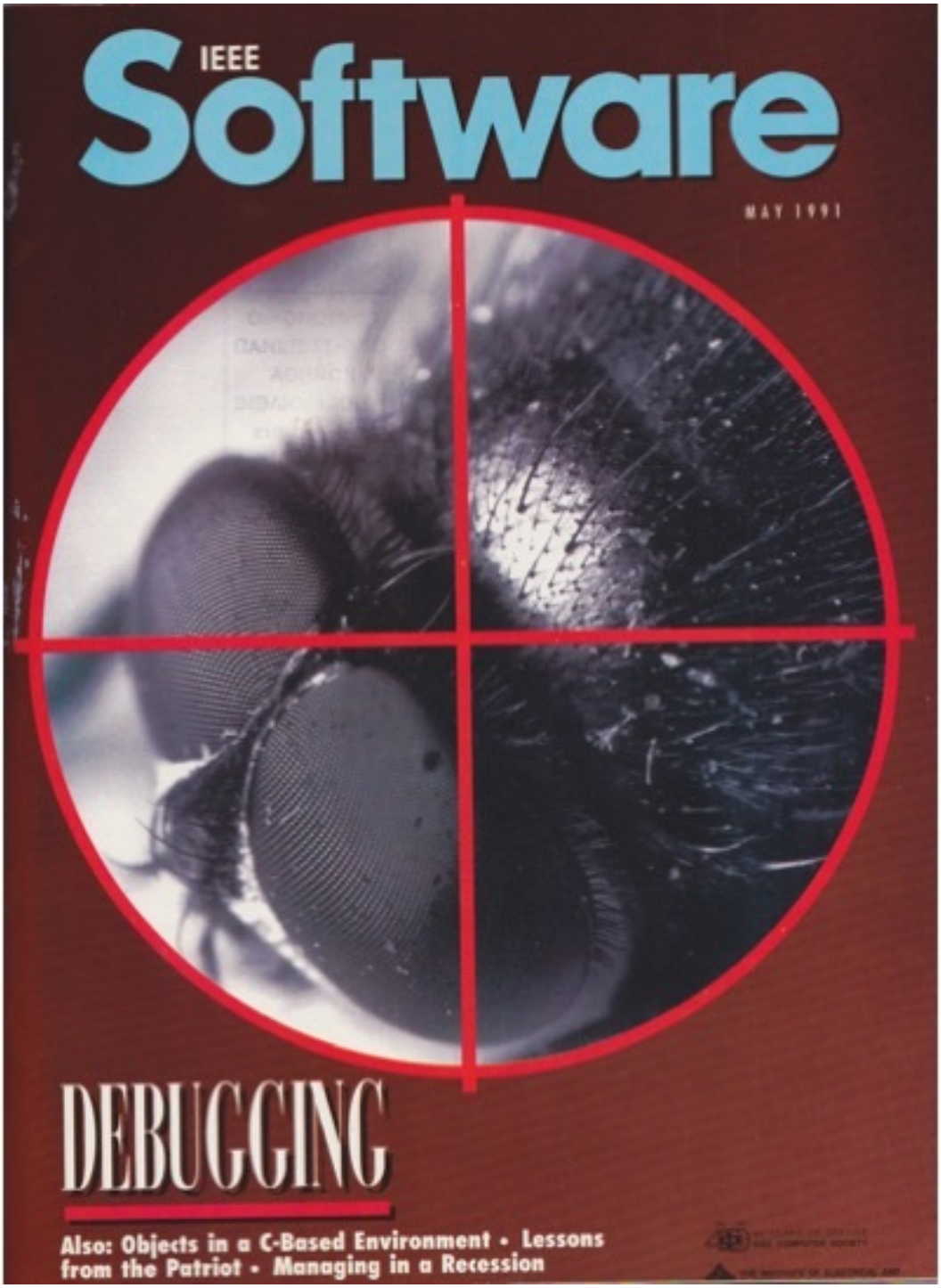
*Ytzhak Levendel, **Improving Quality With a Manufacturing Process**, IEEE Software, March 1991.*[96]

[96] DOI: 10.1109/52.73745

“The testing and quality assurance of the **Motif 1.0** graphical user-interface software are described. The testing goals, which fell into three general categories (code coverage, defect-density, and defect-arrival rate), and a deliverable formal test suite are examined. The three phases of the testing process-evaluation, test development, and regression testing-and the tools used in testing are discussed.”

Paul R. Ritter, Jason Su, ***Experience in Testing the Motif Interface****, IEEE Software, March 1991.*[97]

[97] DOI: 10.1109/52.73746


IEEE
Software
MAY 1991
DEBUGGING
Also: Objects in a C-Based Environment • Lessons from the Patriot • Managing in a Recession

---

“Our **natural limitations** in reasoning power, memory, and communication are both the **reasons we debug** and why the activity of debugging itself can be so difficult. The debuggers … are **visibility tools**, making hidden information available to programmers.”

Thomas G. Moher, Paul R. Wilson, ***Guest Editors’ Introduction: Offsetting Human Limits with Debugging Technology****, IEEE Software, May 1991.*[98]

[98] DOI: 10.1109/MS.1991.10025

“Programmers have no clear idea bow to systematically debug a program, which makes it difficult to share and evaluate methods … Debugging tools must support each stage in the debugging process: hypothesis verification, hypothesis-set modification, and hypothesis selection.”

Jingde Cheng, Keijiro Araki, Zengo Furukawa, ***A General Framework for Debugging****, IEEE Software, May 1991.*[99]

[99] DOI: 10.1109/52.88939


IEEE
Software
JULY 1991
PROCESS
ASSESSMENT
w to improve your score
w to improve the scorecard
ALSO:
• Design Diversity
• Safety Templates
• Academic Piracy

“In 1987 and 1990, the Software Engineering Institute conducted process assessments of the Software Engineering Division (SED) of Hughes Aircraft in Fullerton, CA.”

Watts S. Humphrey, Ronald R. Willis, Terry R. Snyder, ***Software Process Improvement at Hughes Aircraft****, IEEE Software, July 1991.*[100]

[100] DOI: 10.1109/52.300031

“The methods used by the Software Engineering Institute (SEI’s) **Software Capability Evaluation program (SCE)** … is so **seriously and fundamentally flawed** that it should be abandoned rather than modified or updated. “

Clement McGowan, Terry B. Bollinger, ***A Critical Look at Software Capability Evaluations****, IEEE Software, July 1991.*[101]

[101] DOI: 10.1109/52.300034

"**The article** by T. Bollinger and C. McGowan, entitled 'A critical look at software capability evaluation', see ibid., p.25-41 (1991), **contains a serious flaw**. … common misconceptions about the SEI … fall into six categories: SCE's purpose …; how the SCE method works in practice; the statistical methods used to determine levels; the ongoing process of refining the method; the maturity framework; and the coverage of technology issues."

Watts S. Humphrey, Bill Curtis, ***Comments on 'A Critical Look'****, IEEE Software, July 1991.*[102]

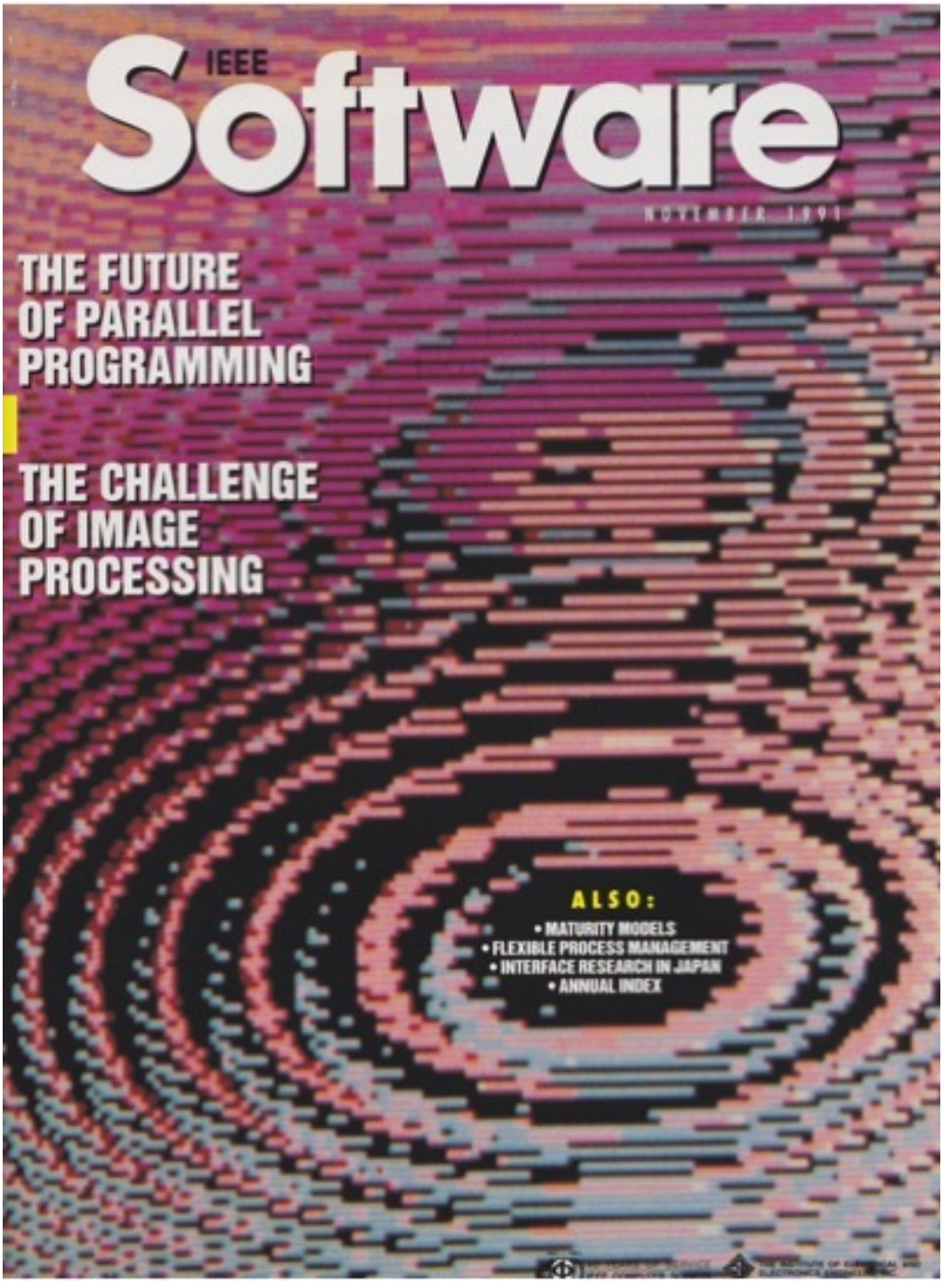
[102] DOI: 10.1109/52.300033


IEEE
Software
SEPTEMBER 1991
HIGH
ERFORMANCE
ALSO:
• WHAT'S WRONG WITH EXPERT SYSTEMS?
• A DAY IN THE LIFE OF A FIRE FIGHTER
• SIGNS OF CONVERGENCE IN MULTIMEDIA MARKET
• COLLABORATIVE INTERFACE TESTING

"In a nutshel **performance analysis** is the measuring, modeling, and tuning of software's time, space, efficiency, and accuracy."

Kathleen Nichols, Paul W. Oman, ***Guest Editors' Introduction: Navigating Complexity to Achieve High Performance****, IEEE Software, September 1991.*[103]

[103] DOI: 10.1109/MS.1991.10056


IEEE
Software
NOVEMBER 1991
THE FUTURE OF PARALLEL PROGRAMMING
THE CHALLENGE OF IMAGE PROCESSING
ALSO:
• MATURITY MODELS
• FLEXIBLE PROCESS MANAGEMENT
• INTERFACE RESEARCH IN JAPAN
• ANNUAL INDEX

“Process-management systems often focus on details at the expense of the big picture. The Cosmos model makes long-term objectives explicit, so managers can have both views. Cosmos … combines the best of existing models by incorporating three perspectives: activity, communication, and infrastructure. Cosmos is designed to manage a large software system from beginning to end.”

Raymond T. Yeh, David A. Naumann, John T. LeBaron, George E. Sumrall, William S. Gilmore, Roland T. Mittermeir, Reinhard A. Schlemmer, ***A Commonsense Management Model****, IEEE Software, November 1991.*[104]

[104] DOI: 10.1109/52.103574

“New computers are supplying the **massive power** that **vision applications** require. How to program this hardware efficiently and naturally is the challenge facing the **imaging community**.”

Virginio Cantoni, Stefano Levialdi, ***Guest Editors’ Introduction: Languages and Environments for Vision Applications****, IEEE Software, November 1991.*[105]

[105] DOI: 10.1109/MS.1991.10062

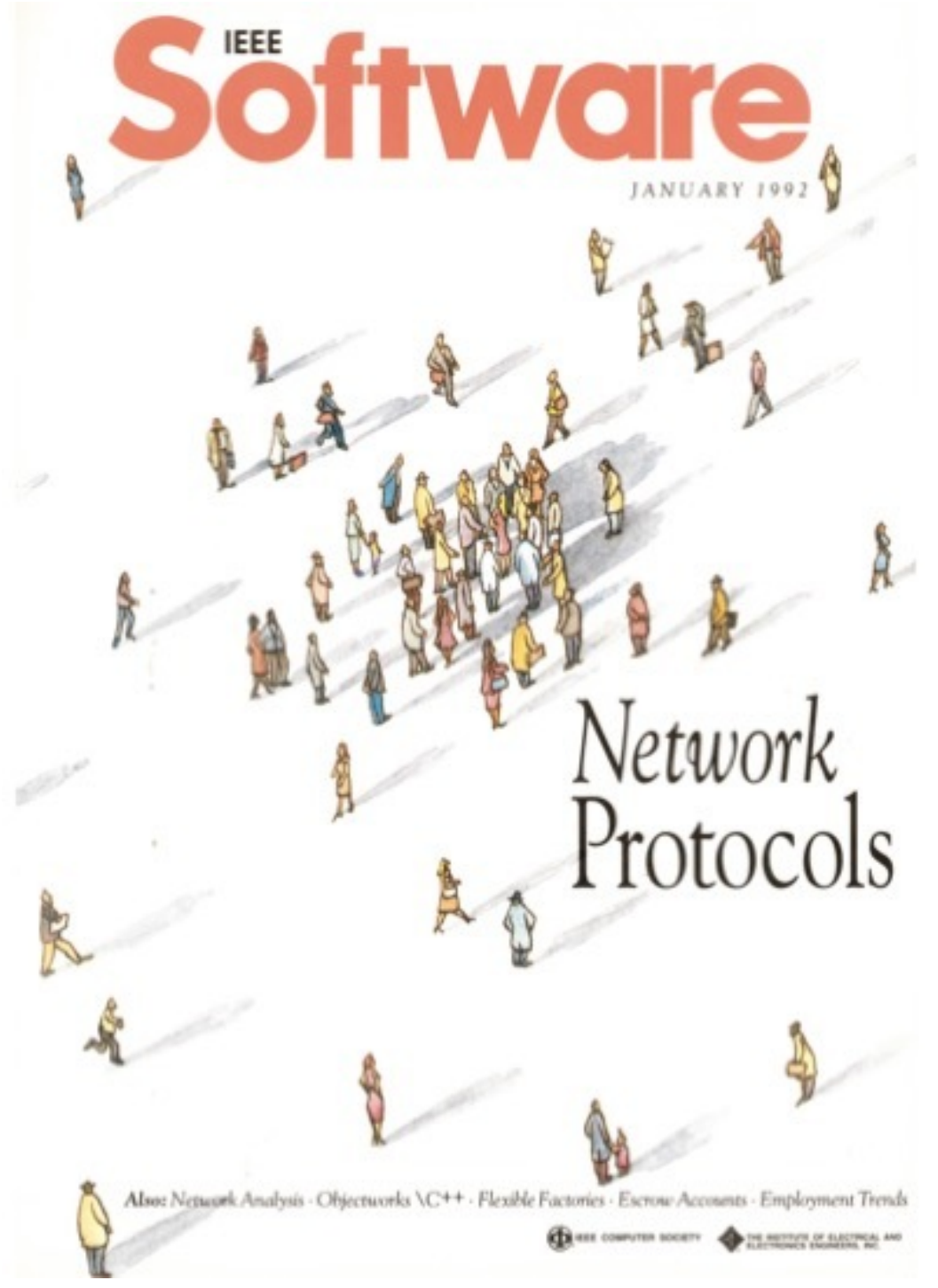
“With advances in **high-level parallel programming**, you can apply known techniques to computer-vision applications, rather than use special computer vision language.”

---

*Anthony P. Reeves, **Parallel Programming for Computer Vision**, IEEE Software, November 1991.*[106]

[106] DOI: 10.1109/52.103577

1992


IEEE
Software
JANUARY 1992
Network Protocols
Also: Network Analysis · Objectworks \C++ · Flexible Factories · Escrow Accounts · Employment Trends
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“Some well-established software fields combining to forge the new discipline of **protocol engineering**, which seeks to ease the development of today’s communication software.”

Ming T. Liu, Fuchun Joseph Lin, ***Guest Editors’ Introduction: The Rise of Protocol Engineering****, IEEE Software, January 1992.*[107]

[107] DOI: 10.1109/MS.1992.10000

“New **formal methods** now exist to design and validate even complex protocols. These methods are mature enough to be used by everyone. … A formal method is considered to be one that has the capability of rendering **correctness proofs**.”

Gerard J. Holzman, ***Protocol Design: Redefining the State of the Art***, *IEEE Software, January 1992.*[108]

[108] DOI: 10.1109/52.108773

"To succesfully validate a very large protocol, you need three ingredients: **formal modeling**, decomposition and abstraction, and **reachability analysis**."

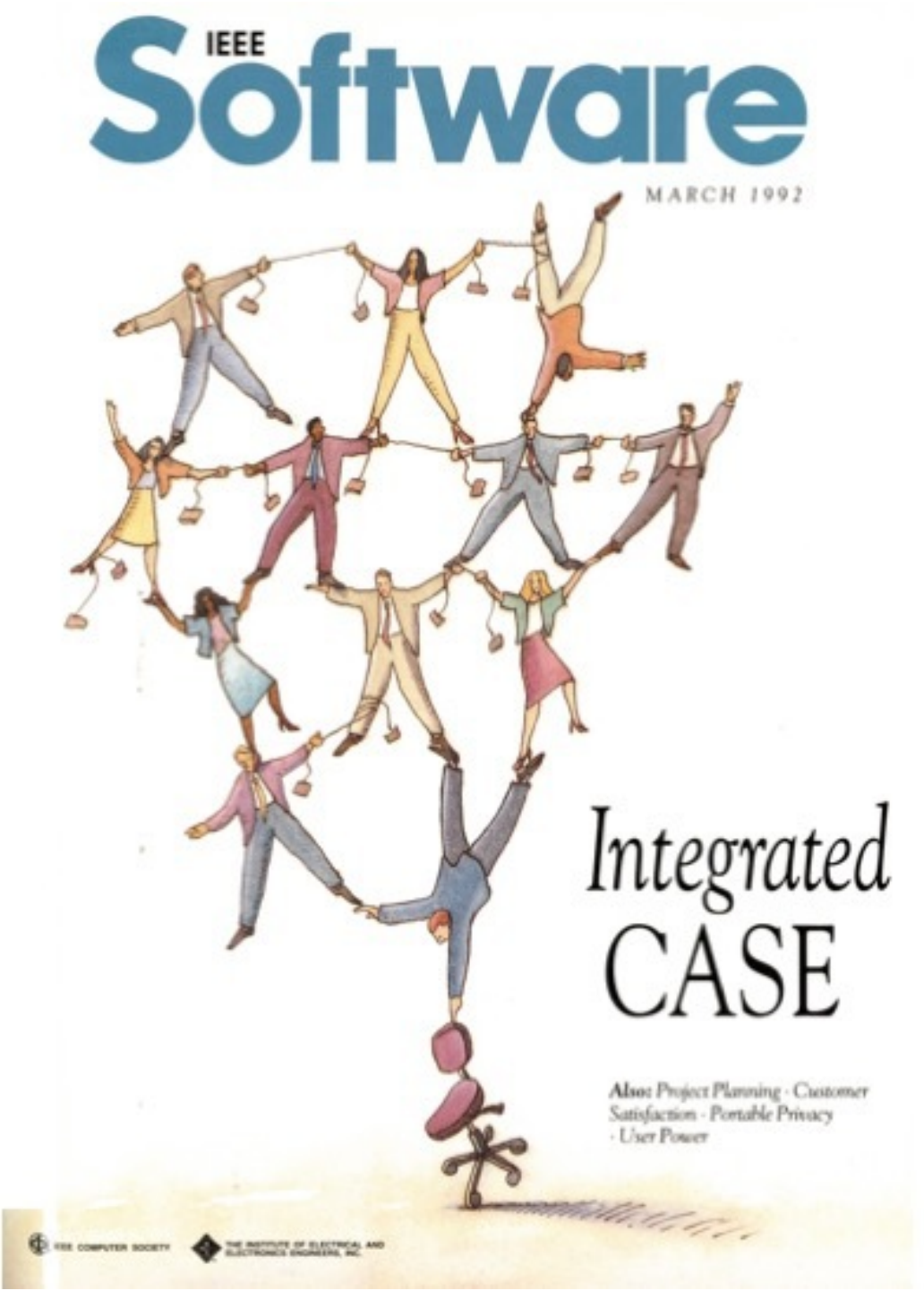
Ming T. Liu, Fuchun Joseph Lin, ***Protocol Validation for Large-Scale Applications****, IEEE Software, January 1992.*[109]

[109] DOI: 10.1109/52.108776

"A **knowledge-based design support system**, called KDSS … helps designers inexperienced in communication system design easily create advanced systems like **intelligent networks**, and large-scale distributed computing systems."

Norio Shiratori, Kenji Sugawara, Kaoru Takahashi, Tetsuo Kinoshita, ***Using Artificial Intelligence in Communication System Design****, IEEE Software, January 1992.*[110]

[110] DOI: 10.1109/52.108779


IEEE
Software
MARCH 1992
Integrated
CASE
Also: Project Planning · Customer Satisfaction · Portable Privacy · User Power
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“The focus of **CASE research and development** has shifted from making sure each tool works to making sure all tools can work together. Major vendors are striving to balance comprehensiveness and compatibility.”

Ronald J. Norman, Minder Chen, ***Guest Editors’ Introduction: Working Together to Integrate CASE****, IEEE Software, March 1992.*[111]

[111] DOI: 10.1109/MS.1992.10026

“The reference model permit **three forms of integration**: **data integration**, **control integration**, and **presentation integration**. … The organizational framework divides systems development and management into **three activity levels**: **IS infrastructure planning and design** is undertaken at the enterprise level, **systems project management and decisions** are made at the project level, and **software-development processes** are carried out at the individual and team level. “

Ronald J. Norman, Minder Chen, ***A Framework for Integrated CASE****, IEEE Software, March 1992.*[112]

[112] DOI: 10.1109/52.120597

“Current work on **integrated project support environments (IPSEs)** is based on an inappropriate view of integration, and … IPSE developers should create semantically rich infrastructures or produce well-integrated tool sets rather than open repositories or infrastructures.”

John A. McDermid, Alan W. Brown, ***Learning From IPSE’s Mistakes****, IEEE Software, March 1992.*[113]

[113] DOI: 10.1109/52.120598

“**Tool integration** is not a property of a single tool, but of its relationships with other elements in the environment, chiefly other tools, a platform, and a process. Tool integration is about the extent to which tools agree. The subject of these agreements may include **data format**, **user-interface conventions**, use of **common functions** …”

Brian A. Nejmeh, Ian Thomas, ***Definitions of Tool Integration for Environments****, IEEE Software, March 1992.*[114]

[114] DOI: 10.1109/52.120599

"The **software factory** concept … symbolizes a desired paradigm shift from labor-intensive software production to a more **capital-intensive style** in which substantial **investments** can be made at an acceptable **risk level** …"

Christer Fernström, Kjell-Håken Närfelt, Lennart Ohlsson,

***Software Factory Principles, Architecture, and Experiments**,*

IEEE Software, March 1992.[115]

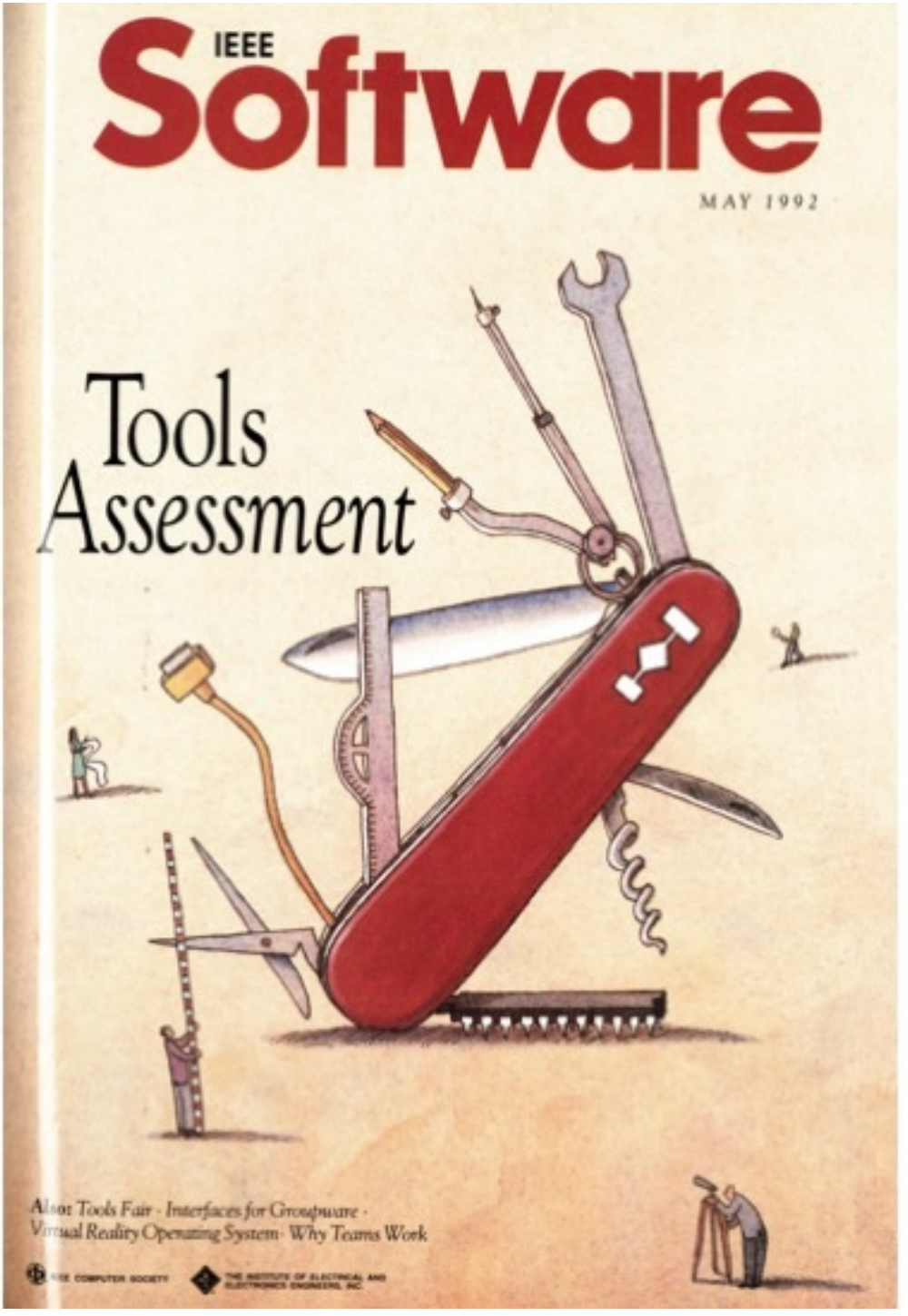
[115] DOI: 10.1109/52.120600

“HyperCASE integrates tools by combining a **hypertext-based user interface** with a common knowledge-based document repository.”

---

*Karl Reed, Jacob L. Cybulski, **A Hypertext Based Software-Engineering Environment**, IEEE Software, March 1992.*[116]

[116] DOI: 10.1109/52.120603


IEEE
Software
MAY 1992
Tools
Assessment
About Tools Fair · Interfaces for Groupware ·
Virtual Reality Operating System · Why Teams Work
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

"The industry has made great strides in making **more development tools** available. It's now time to find ways to consistently, objectively **evaluate** a tool's **utility and appropriateness**."

Elliot J. Chikofsky, David E. Martin, Hugh Chang, ***Assessing the State of Tools Assessment****, IEEE Software, May 1992.*[117]

[117] DOI: 10.1109/MS.1992.10039

“Organizations buy integrated CASE tools only to **leave them on the shelf** because they misinterpret the learning curve and its effect on productivity.”

Chris F. Kemerer, ***How the Learning Curve Affects CASE Tool Adoption***, *IEEE Software*, *May 1992.*[118]

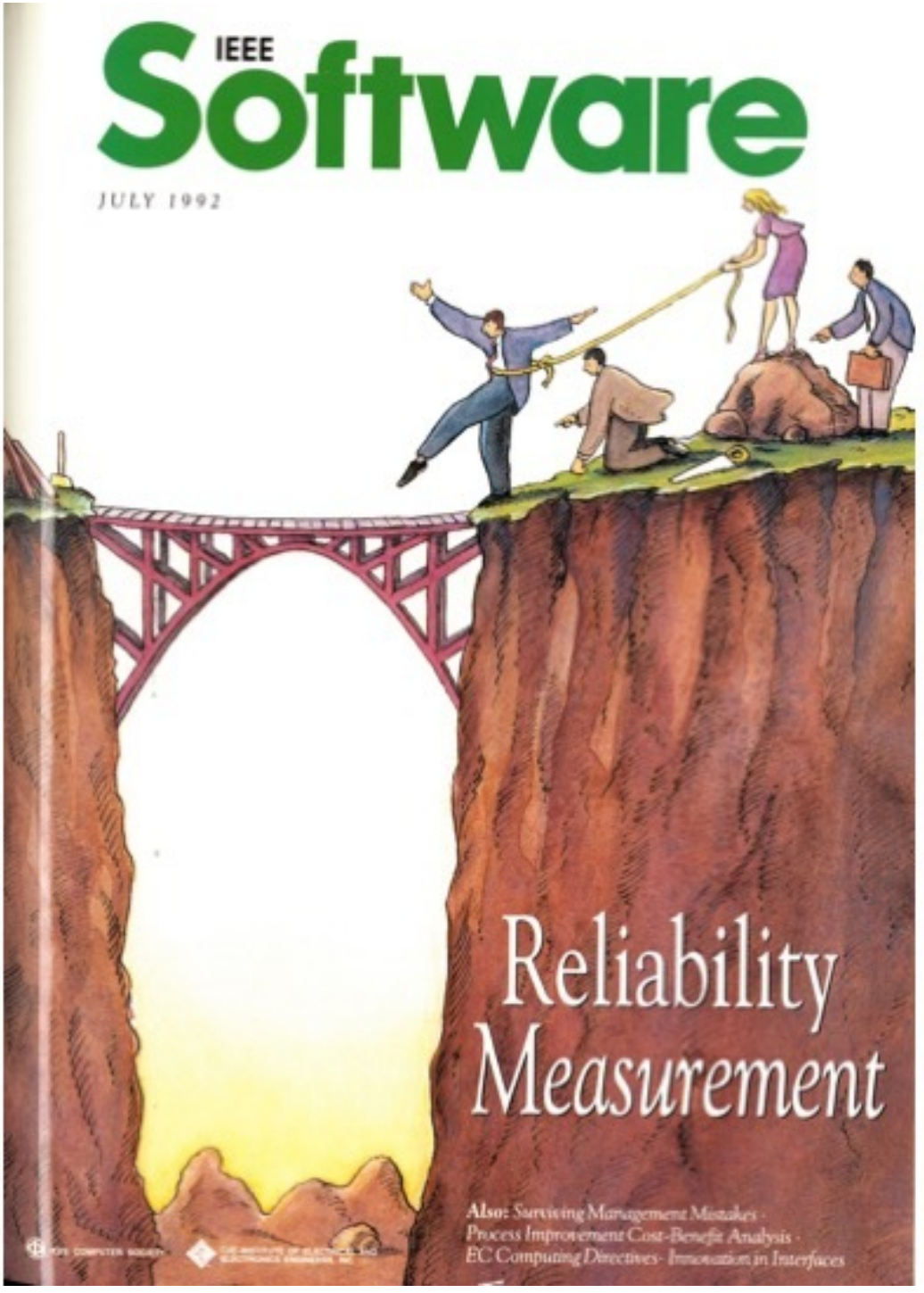
[118] DOI: 10.1109/52.136161

"Six categories of questions … determine how well a tool does what it was intended to do. The questionnaire comprises 140 questions divided into the categories: **ease of use**, **power**, **robustness**, **functionality**, **ease of insertion**, and **quality of support**."

---

*Vicky Mosley, **How to Assess Tools Efficiently and Quantitatively**, IEEE Software, May 1992.*[119]

[119] DOI: 10.1109/52.136163


IEEE
Software
JULY 1992
Reliability
Measurement
Also: Surviving Management Mistakes · Process Improvement Cost-Benefit Analysis · EC Computing Directives · Innovation in Interfaces

---

“Most developers either aren‘t familiar with **reliability models** or don‘t know bow to select and apply them. But the need for accurate predictions is acute, focusing attention on this comparatively young field. “

Pradip K. Srimani, Yashwant K. Malaiya, ***Guest Editors’ Introduction: Steps to Practical Reliability Measurement****, IEEE Software, July 1992.*[120]

[120] DOI: 10.1109/MS.1992.10044

“**Reliability** is a probability of **failure-free operation** for a specified time in a specified environment for a specified purpose.”

Krishna M. Kavi, Robert C. Tausworth, William W. Everett, Frederick T. Sheldon, Ralph Brettschneider, James T. Yu, ***Reliability Measurement: From Theory to Practice****, IEEE Software, July 1992.*[121]

[121] DOI: 10.1109/52.143095

“**Reliability** is the **statistical study of failures**, which occur because of some defect in the program. The failure is evident, but you don’t know what mistake is responsible or what you can do to make the failure disappear. **Reliability models** are supposed to tell you what **confidence** you can have in the **program’s correctness**. “

---

*Dick Hamlet, **Are We Testing for True Reliability?**, IEEE Software, July 1992.*[122]

[122] DOI: 10.1109/52.143097

"Three separate but related functions comprise an **integrated reliability program**: **prediction**, **control**, and **assessment**."

Ted W. Keller, Norman F. Schneidewind, ***Applying Reliability Models to the Space Shuttle****, IEEE Software, July 1992.*[123]

[123] DOI: 10.1109/52.143099

IEEE
Software
SEPTEMBER 1992

Real-Time Realities

Also: Operational Prototyping - Simplifying Process Metrics - Buyers, Builders, and Managers - Memory Management

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“In the future, **most consumer products** will **rely on embedded real-time computers**. To produce more robust applications, development techniques must keep pace with ever-changing requirements.”

Kwei-Jay Lin, Emmett J. Burke, ***Guest Editors’ Introduction: Coming to Grips with Real-Time Realities****, IEEE Software, September 1992.*[124]

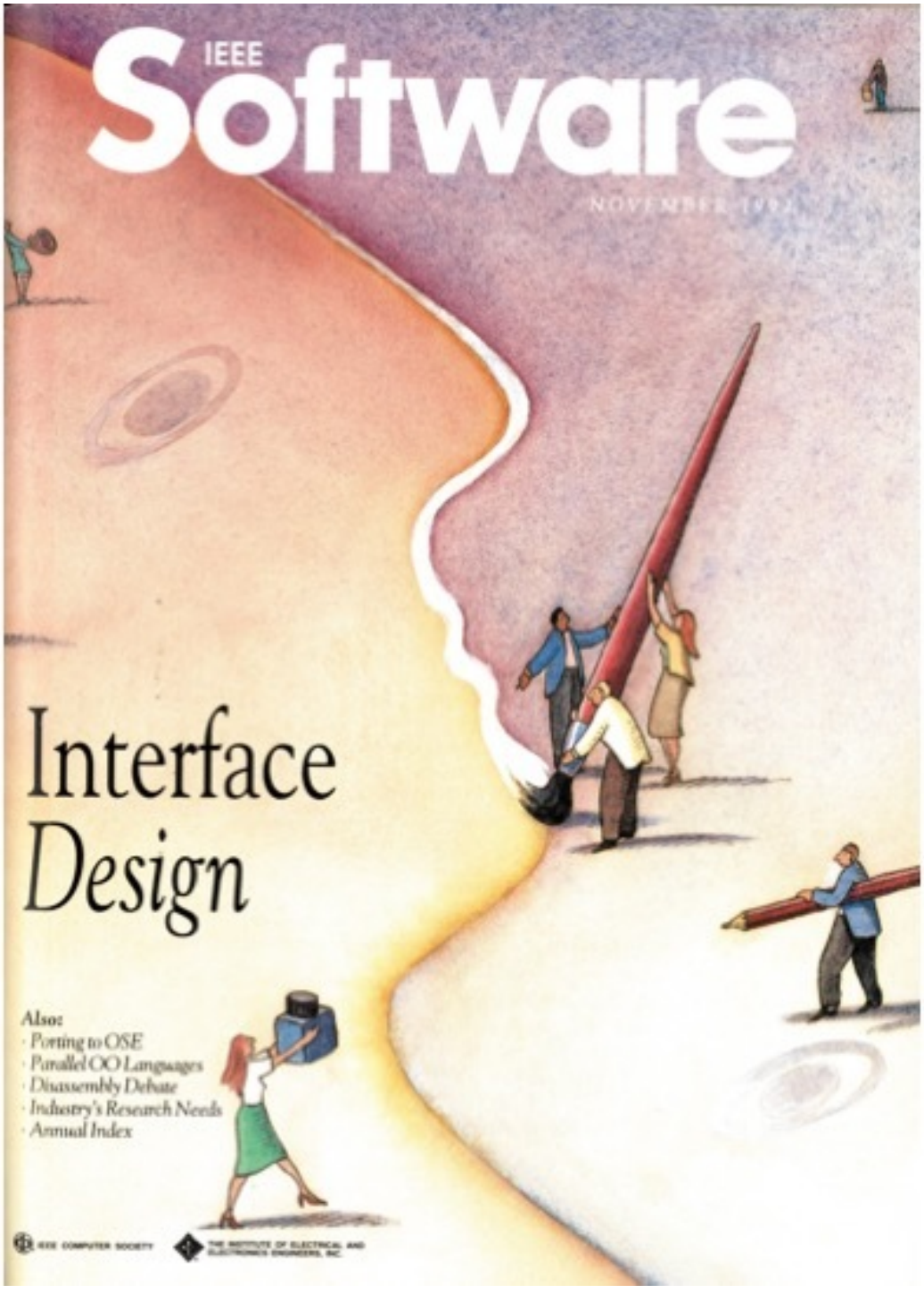
[124] DOI: 10.1109/MS.1992.10059

"The problem of **ensuring timing correctness** in **dynamic real-time systems** has three aspects: **resource requirements**, **resource availability**, and **guarantees**."

---

*Swaminathan Natarajan, Wei Zhao, **Issues in Building Dynamic Real-Time Systems**, IEEE Software, September 1992.*[125]

[125] DOI: 10.1109/52.156893


IEEE
Software
NOVEMBER 1992
Interface Design
Also:
· Porting to OSE
· Parallel OO Languages
· Disassembly Debate
· Industry's Research Needs
· Annual Index
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“Much of a HyperNews interface can be designed without writing any code at all. With **direct manipulation**, a user can design a graphical user interface simply by **creating**, **moving**, and **resizing** objects on the screen. Experimentation with different interface styles is possible …”

Jim Rudolf, Cathy Waite, ***Completing the Job Interface Design****, IEEE Software, November 1992.*[126]

[126] DOI: 10.1109/52.168854

"The Interaction Management Network (IMN) … captures the essential structure of any interface from **task-oriented specification** to **object-oriented implementation** is presented. IMN is essentially a **task-oriented specification scheme** based on a semantic network."

Raimund K. Ege, Christian Stary, ***Designing Maintainable, Reusable Interfaces****, IEEE Software, November 1992.*[127]

[127] DOI: 10.1109/52.168855

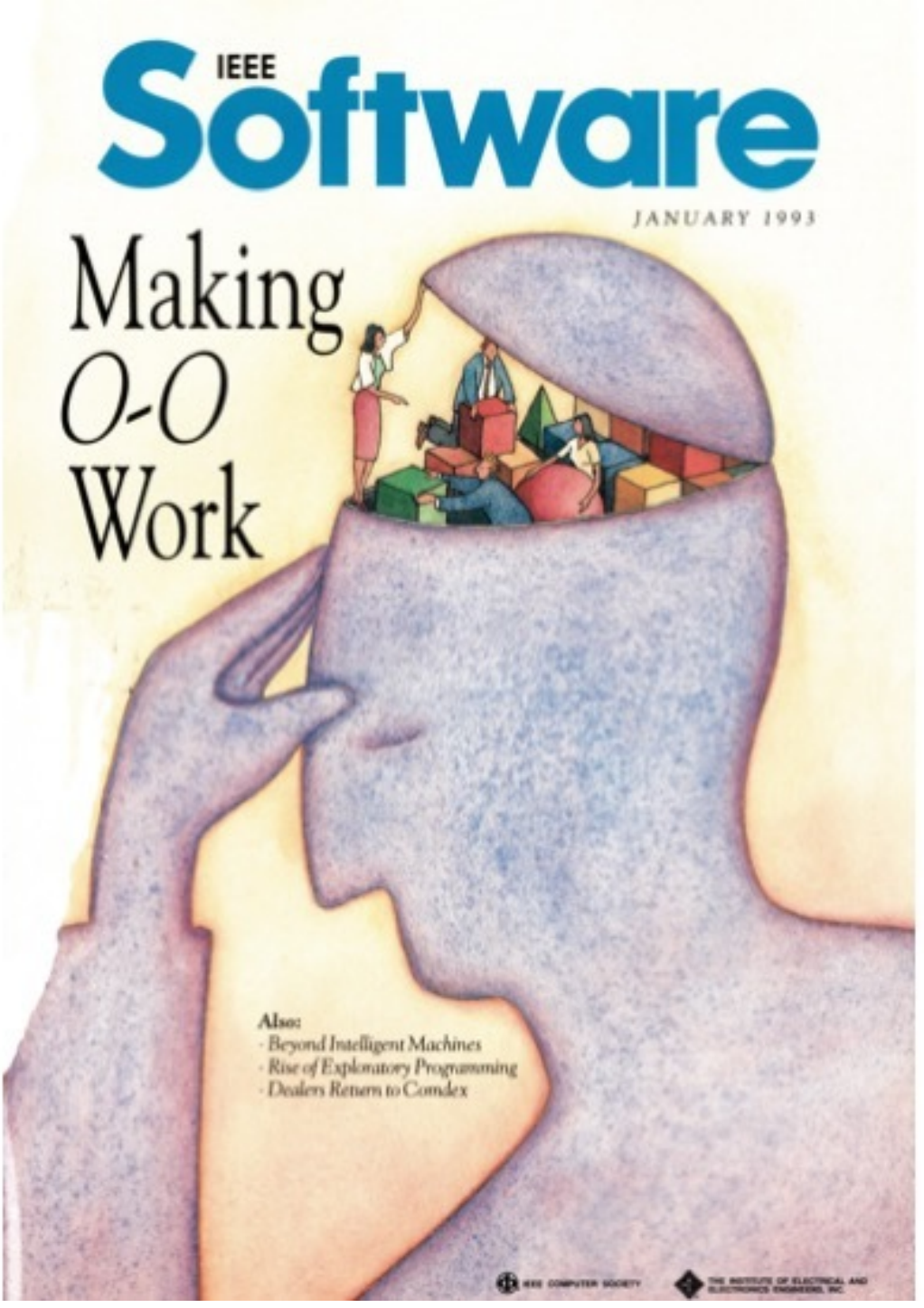
“Fourteen **concurrent object-oriented languages** are compared in terms of how they deal with **communication**, **synchronization**, **process management**, **inheritance**, and **implementation trade-offs**. … The languages discussed are **Actors**, **Abd/1**, **Abd/R**, **Argus**, **COOL**, **Concurrent Smalltalk**, **Eiffel**, **Emerald**, **ES-Kit C++**, **Hybrid**, **Nexus**, **Parmacs**, **POOL-T**, and **Presto**.”

---

*Krishna Kavi, Steve Hufnagel, Barbara B. Wyatt, **Parallelism in Object-Oriented Languages: A Survey**, IEEE Software, November 1992.*[128]

[128] DOI: 10.1109/52.168859

1993


IEEE
Software
JANUARY 1993
Making
O-O
Work
Also:
· Beyond Intelligent Machines
· Rise of Exploratory Programming
· Dealers Return to Comdex

“Will **object orientation** be the dominant paradigm in the near future? If it is to be widely used, it must overcome many legacies, especially the fact that most developers do not **think in terms of objects**.”

Annie Kuntzmann-Combelles, Wojtek Kozaczynski, ***What it Takes to Make OO Work****, IEEE Software, January 1993.*[129]

[129] DOI: 10.1109/MS.1993.10005

"The wide **acceptance of the object-oriented approach** is unprecedented in computer technoLogy. Judging from the the experience the last 25 year, developers who adopt this platform stand to reap generous rewards."

Ivar Jacobson, ***Is Object Technology Software's Industrial Platform?****, IEEE Software, January 1993.*[130]

[130] DOI: 10.1109/52.207218

“For better or worse, the **real language winner today is C++**.”

Ivar Jacobson, ***Is Object Technology Software’s Industrial Platform?****, IEEE Software, January 1993.*[131]

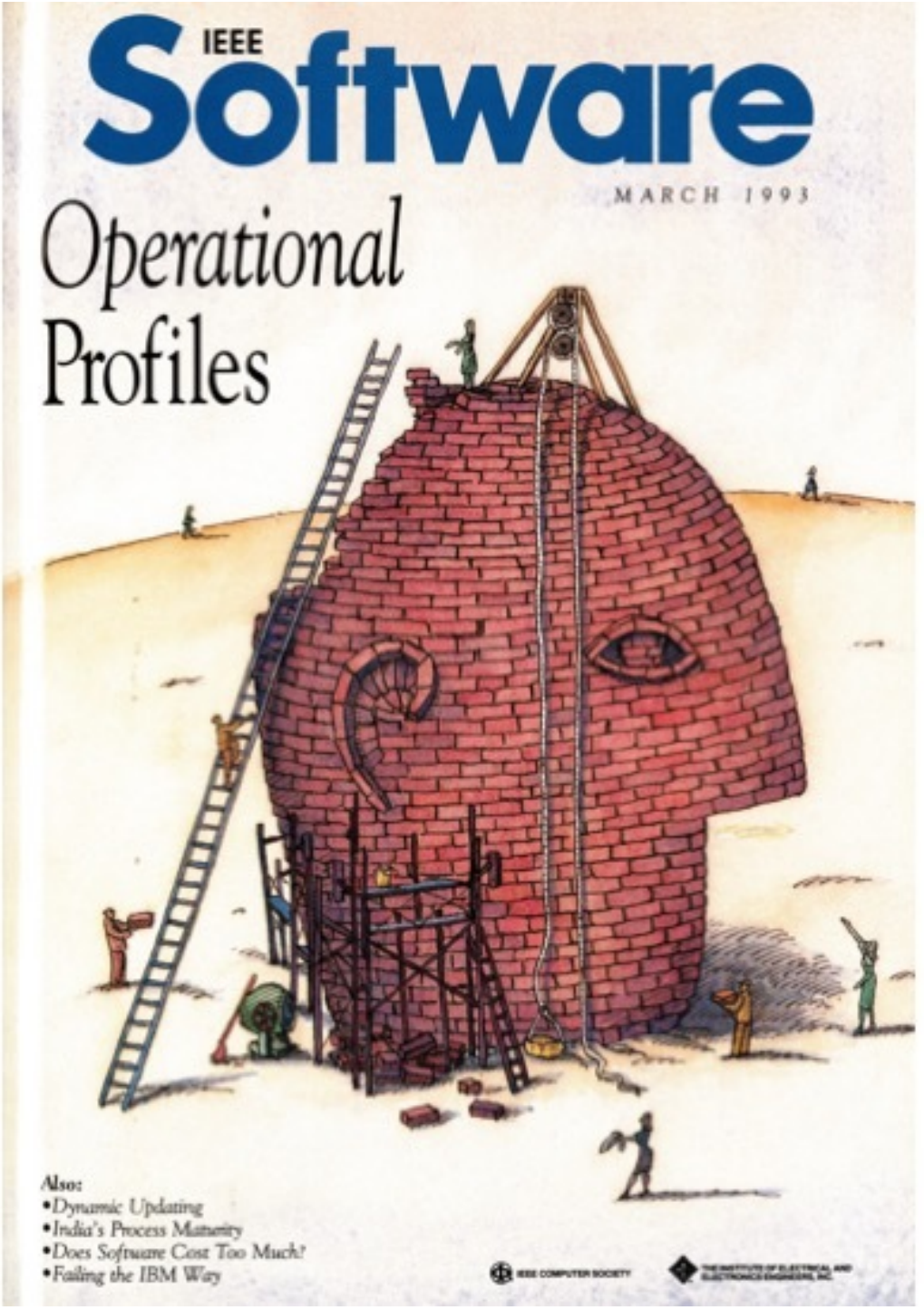
[131] DOI: 10.1109/52.207218

“**Object orientation** facilitates change but makes programs harder for maintainers to understand. … The appearance and organization of OO object-oriented code **may startle may programmers**.”

---

*Paul Matthews, Norman Wilde, Ross Huitt, **Maintaining Object-Oriented Software**, IEEE Software, January 1993.*[132]

[132] DOI: 10.1109/52.207232


IEEE
Software
MARCH 1993
Operational Profiles
Also:
•Dynamic Updating
•India's Process Maturity
•Does Software Cost Too Much?
•Failing the IBM Way
IEEE COMPUTER SOCIETY

“An **operational profile** describes how users employ a system … The operational profile is a **quantitative characterization** of **how a system will be used** that shows how to increase productivity and reliability and speed development by allocating development resources to function on the basis of use. Using an operational profile to **guide testing** ensures … the most-used operations will have received the most testing …”

John D. Musa, ***Operational Profiles in Software-Reliability Engineering***, *IEEE Software, March 1993.*[133]

[133] DOI: 10.1109/52.199724

“Determining the consequences of a stop-test decision … combines software reliability engineering and economic analysis … The approach … **quantify the economic consequences** associated with **terminating testing** …”

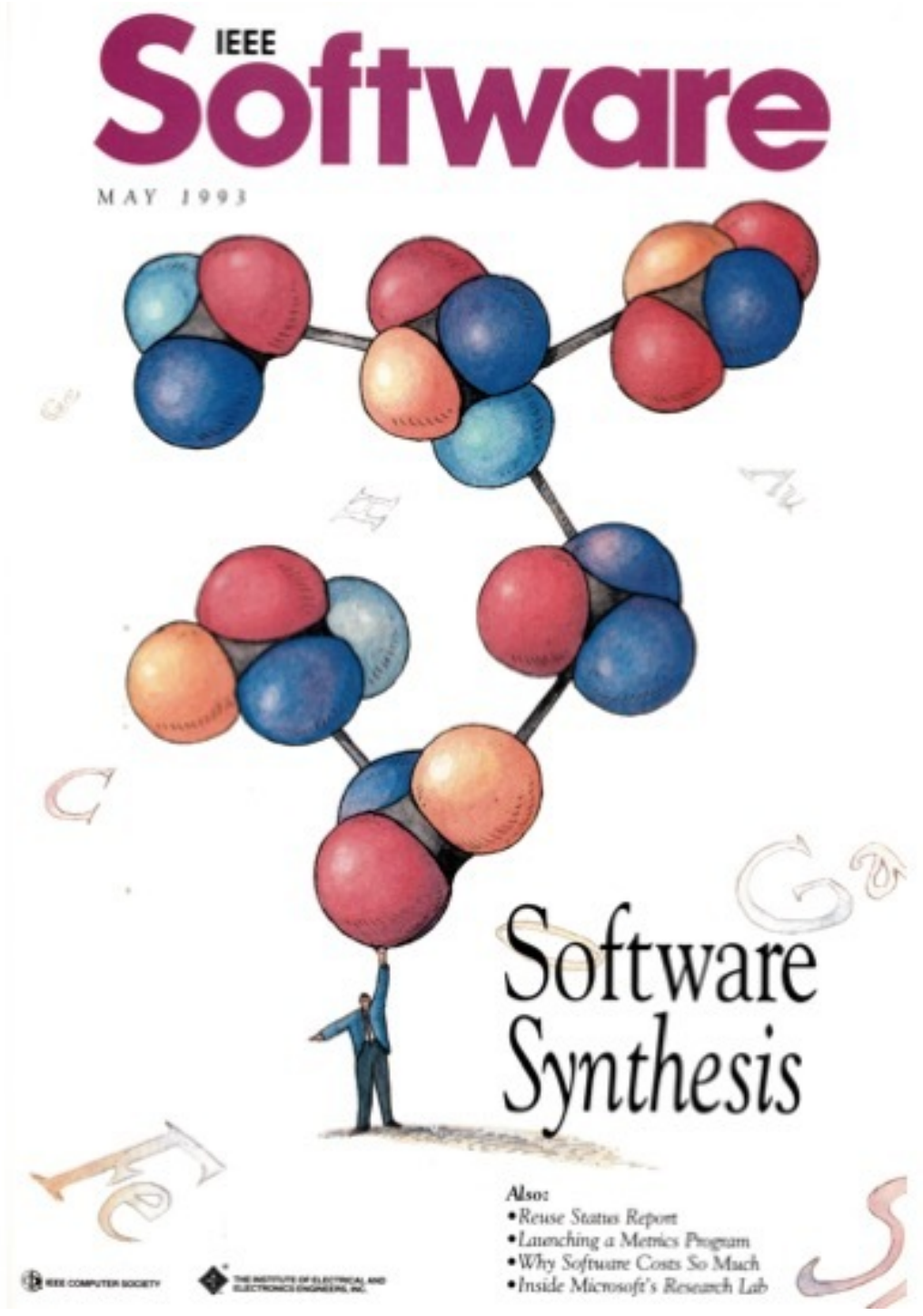
Willa Ehrlich, Jar Wu, Bala Prasanna, John Stampfel, ***Determining the Cost of a Stop-Test Decision****, IEEE Software, March 1993.*[134]

[134] DOI: 10.1109/52.199726

“**The Shlaer-Mellor object-oriented analysis method** provides a structured means of identifying objects within a system by analyzing abstract data types and uses these objects as a basis for building three formal models: **information**, **state**, and **process**.”

Mark A. Roberts, Jerry R. Klatt, Mohamed E. Fayad, Louis J. Hawn, ***Using the Shlaer-Mellor Object-Oriented Analysis Method****, IEEE Software, March 1993.*[135]

[135] DOI: 10.1109/52.199729


IEEE
Software
MAY 1993
Software Synthesis
Also:
•Reuse Status Report
•Launching a Metrics Program
•Why Software Costs So Much
•Inside Microsoft's Research Lab
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“One idea that holds great promise in increasing software productivity is the **automatic synthesis of software** from higher level specifications and reusable components.”

---

*Tom Cain, Elaine Kant, Dorothy Setliff, **Practical Software Synthesis**, IEEE Software, May 1993.*[136]

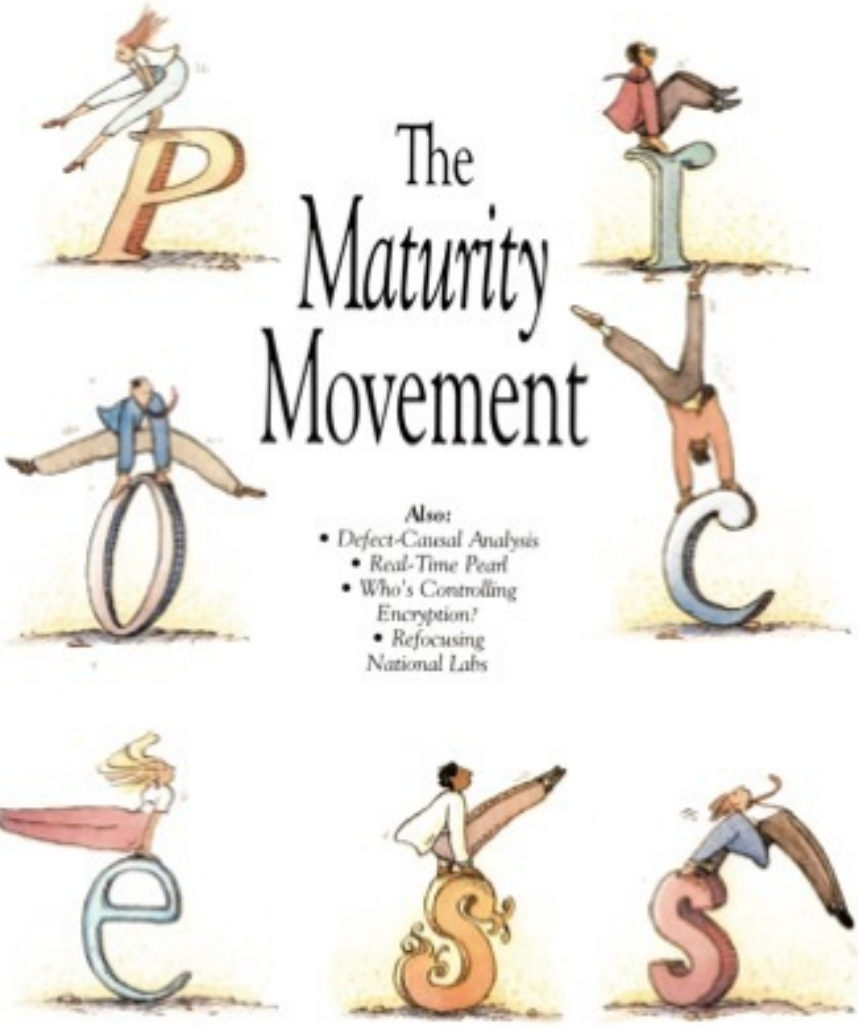
[136] DOI: 10.1109/MS.1993.10027

“A **model-based**, automatic software synthesis environment … automatically generates a macro-dataflow computation from declarative models. Central to the approach is the Multigraph Architecture, which provides the framework for model-based synthesis in real-time, parallel-computing environments.”

Ted Bapty, Ben Abbott, Gabor Karsai, Janos Sztipanovits, Csaba Biegl, ***Model-Based Software Synthesis****, IEEE Software, May 1993.*[137]

[137] DOI: 10.1109/52.210602

IEEE
Software
JULY 1993



The Maturity Movement

Also:
- *Defect-Causal Analysis*
- *Real-Time Pearl*
- *Who's Controlling Encryption?*
- *Refocusing National Labs*

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“Competitive pressure is pushing and pulling the field to a new level of industrialization. Developers know they **must stop improvising** and instill some discipline in to their process”

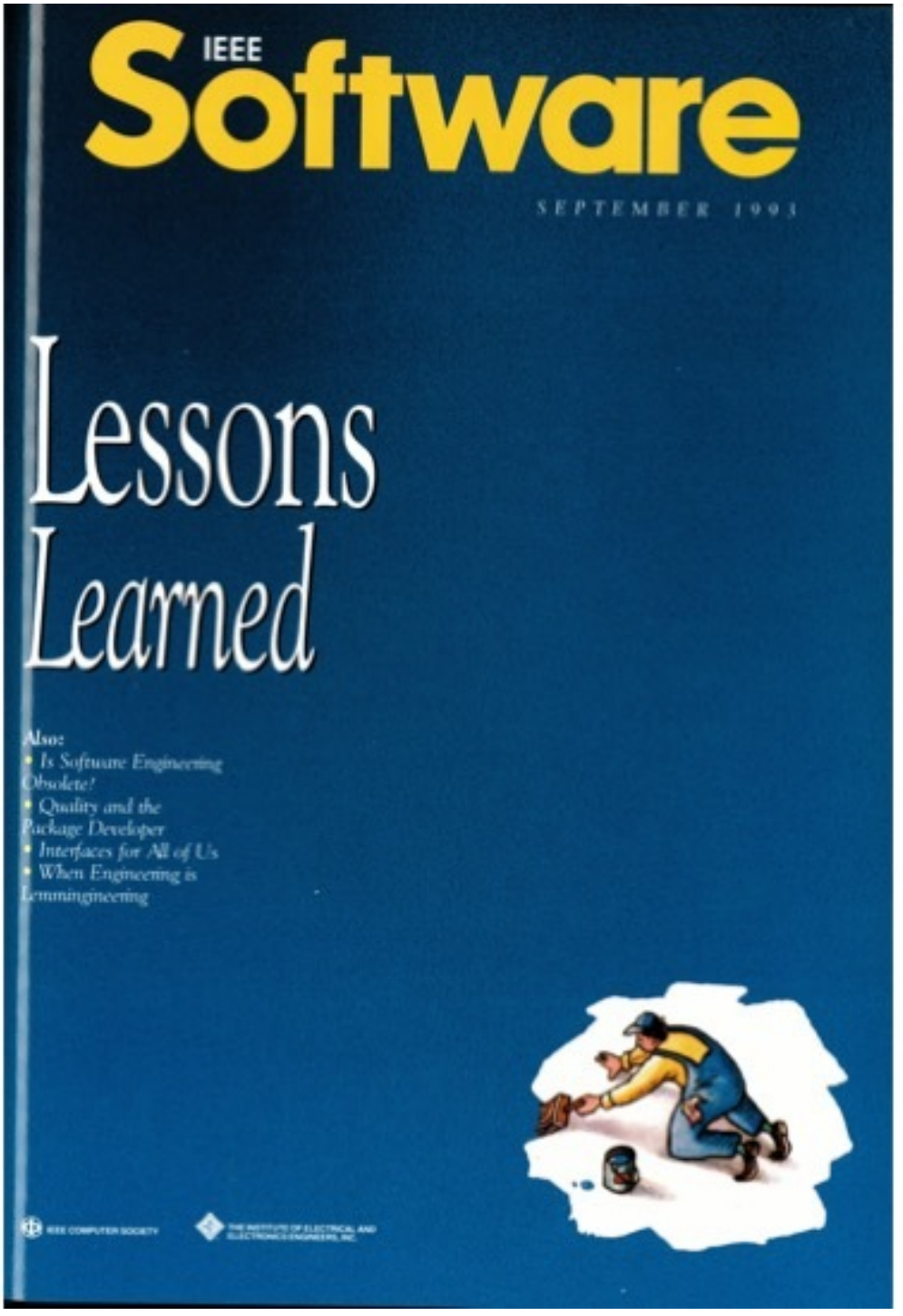
Robert Lai, ***The Move to Mature Processes***, *IEEE Software, July 1993.*[138]

[138] DOI: 10.1109/MS.1993.10038

“The **CMM capability maturity model** was designed to help developers select **process-improvement strategies** by determining their current process maturity and identifying the issues most critical to improving their software quality and process. … The current version of the CMM is the result of the feedback from that workshop and ongoing feedback from the software community.”

Mark C. Paulk, Charles V. Weber, Mary Beth Chrissis, Bill Curtis, ***Capability Maturity Model, Version 1.1****, IEEE Software, July 1993.*[139]

[139] DOI: 10.1109/52.219617


IEEE
Software
SEPTEMBER 1993
Lessons Learned
Also:
• Is Software Engineering Obsolete?
• Quality and the Package Developer
• Interfaces for All of Us
• When Engineering is Lemmingneering
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“**Following good software-engineering practices** is almost **a moral issue**; we need a belief system.”

Pei Hsia, ***Learning to Put Lessons Into Practice***, *IEEE Software, September 1993.*[140]

[140] DOI: 10.1109/MS.1993.10046

“We are so **used to the notion** that **quality must take a back seat to productivity** that we continue to put up with practices that we know will produce software of lesser quality.”

Pei Hsia, ***Learning to Put Lessons Into Practice***, *IEEE Software*, *September 1993.*[141]

[141] DOI: 10.1109/MS.1993.10046

“Most software engineering research has been following a **research-then-transer** methodology, with mixed results … In the **industry-as-laboratory** approach … researchers identify problems through **close involvement with industrial projects** and create and evaluate solutions in an almost **indivisible research activity**. This approach emphasizes what people actually do or can do in practice, rather than what is possible in principle.”

---

*Colin Potts, **Software-Engineering Research Revisited**, IEEE Software, September 1993.*[142]

[142] DOI: 10.1109/52.232392

"**Software engineering** most resembles a **dynamically overloaded queue or rush-hour traffic jam**."

Neil C. Olsen, ***The Software Rush Hour***, *IEEE Software*, *September 1993.*[143]

[143] DOI: 10.1109/52.232394

“Five problems … can contribute to a software project’s failure … : **inadequate system engineering** during the proposal and during front-end development; **inadequate tracing**, tracking, and management of system and software requirements; **improper sizing** of the target hardware; selection of design, production, and integration and test **methodologies that are inappropriate** to a large software development; and **failure to provide a metrics program** that would let managers track the progress of software production and test.”

---

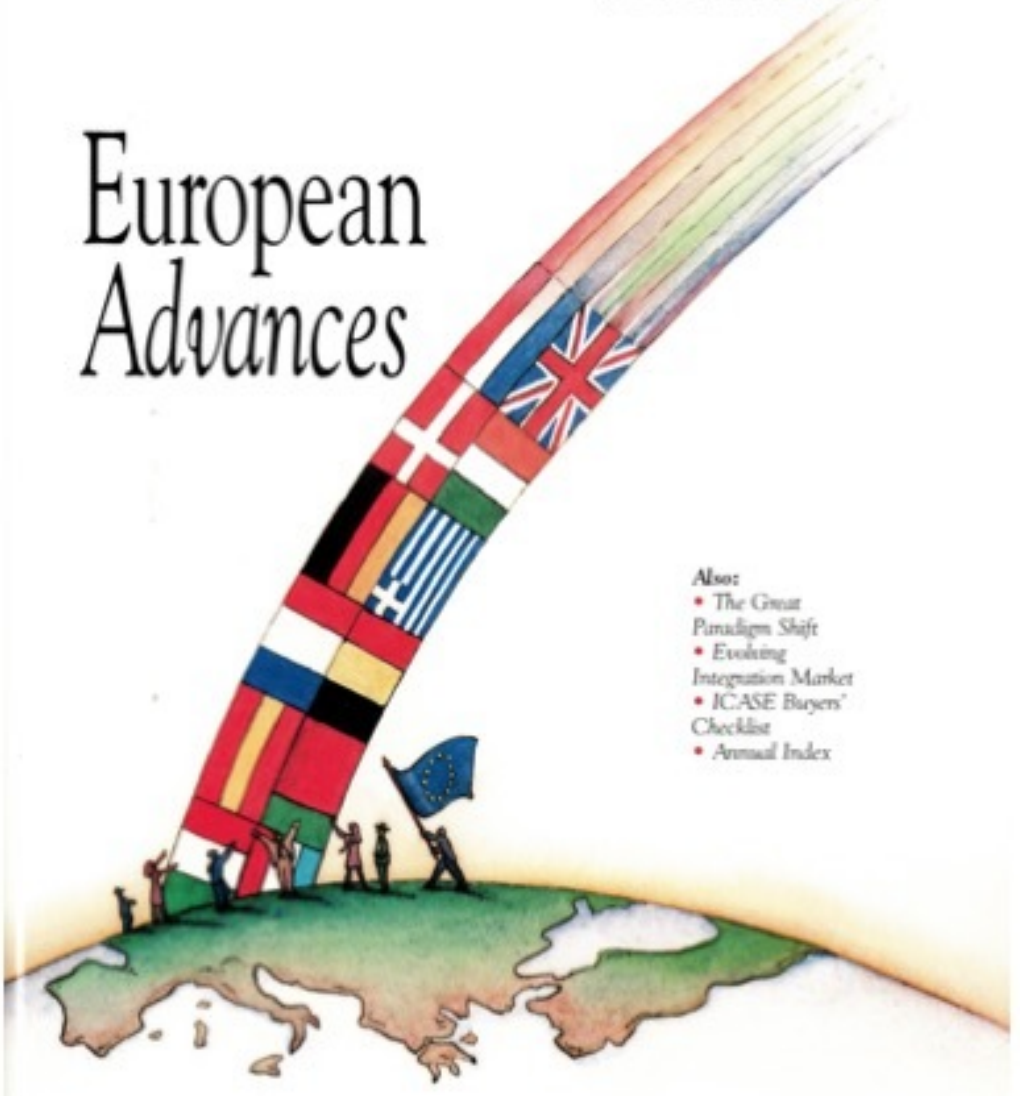
*David R. Lindstrom, **Five Ways to Destroy a Development Project**, IEEE Software, September 1993.*[144]

[144] DOI: 10.1109/52.232400

IEEE Software

NOVEMBER 1993

European Advances



Also:
- The Great Paradigm Shift
- Evolving Integration Market
- ICASE Buyers' Checklist
- Annual Index

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“**Europe** has **spent billions** in public and private funds on developing its **information-technology industry**. This effort has certainly made the EC a stronger global competitor; but the question remains: **Is it strong enough**?”

Luqi null, Annie Kuntzmann-Combelles, ***Guest Editors’ Introduction: Advancing Europe’s Fortunes****, IEEE Software, November 1993.*[145]

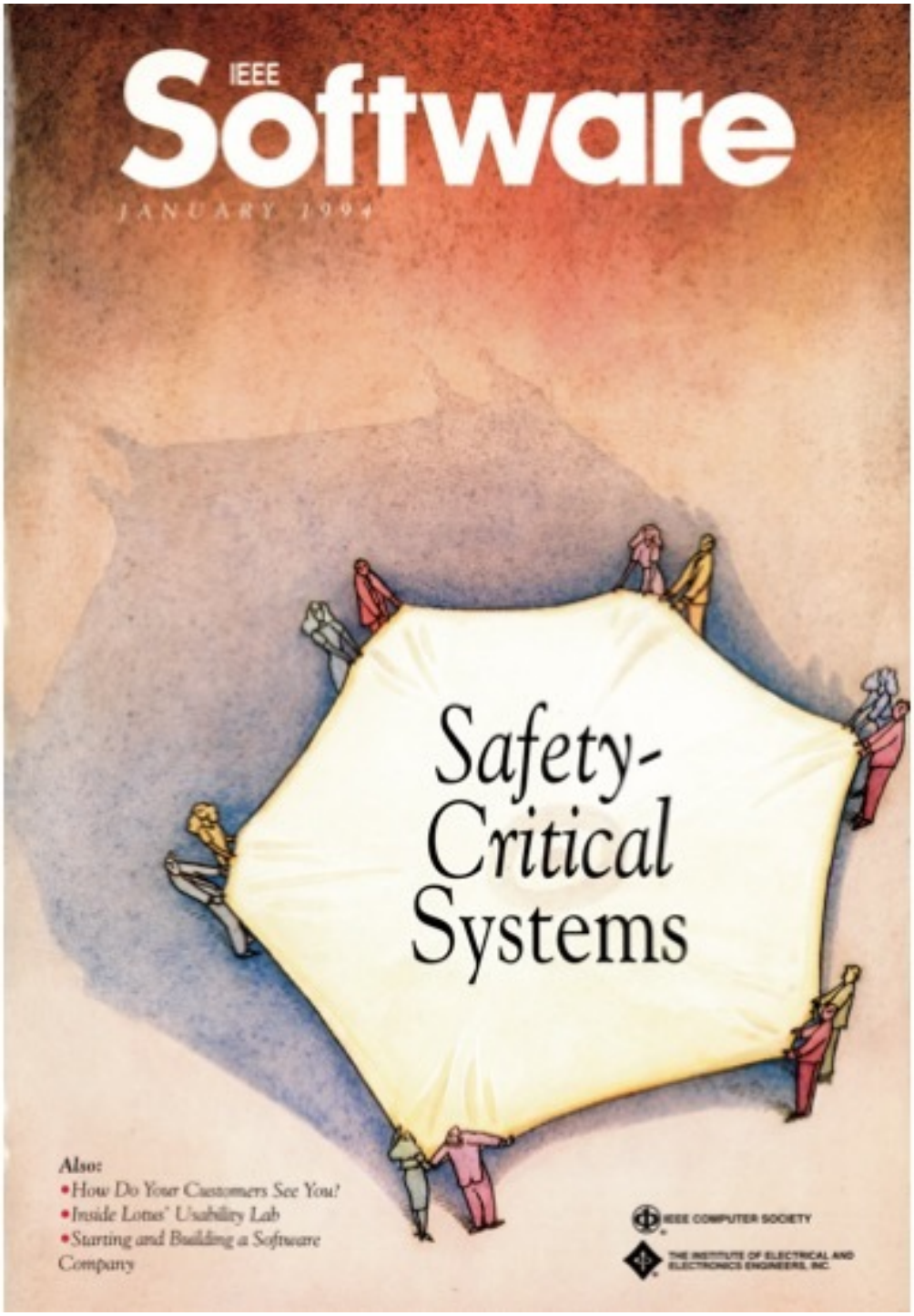
[145] DOI: 10.1109/MS.1993.10067

"A big problem in technology transfer is that **no one believes in the product**. By using a system of **hypotheses and experiments**, technology providers can begin to offer **real benefits instead of hype**."

Heinz Saria, Christophe Debou, Norbert Fuchs, ***Selling Believable Technology****, IEEE Software, November 1993.*[146]

[146] DOI: 10.1109/52.241961

1994


IEEE
Software
JANUARY 1994
Safety-Critical Systems
Also:
•How Do Your Customers See You?
•Inside Lotus' Usability Lab
•Starting and Building a Software Company
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“**Safety-critical software** must perform as desired and **should never fail**. The need for dependability stems from the fact that the **consequences of failure are extremely high**, usually a **threat to human life**. To write such systems, most now agree that we must adopt rigorous techniques, rooted in mathematics.”

Bev Littlewood, John Knight, ***Guest Editors’ Introduction: Critical Task of Writing Dependable Software****, IEEE Software, January 1994.*[147]

[147] DOI: 10.1109/52.251196

“Although there are indisputable benefits to society from the **introduction of computers into everyday life**, some applications are inherently **risky**.”

---

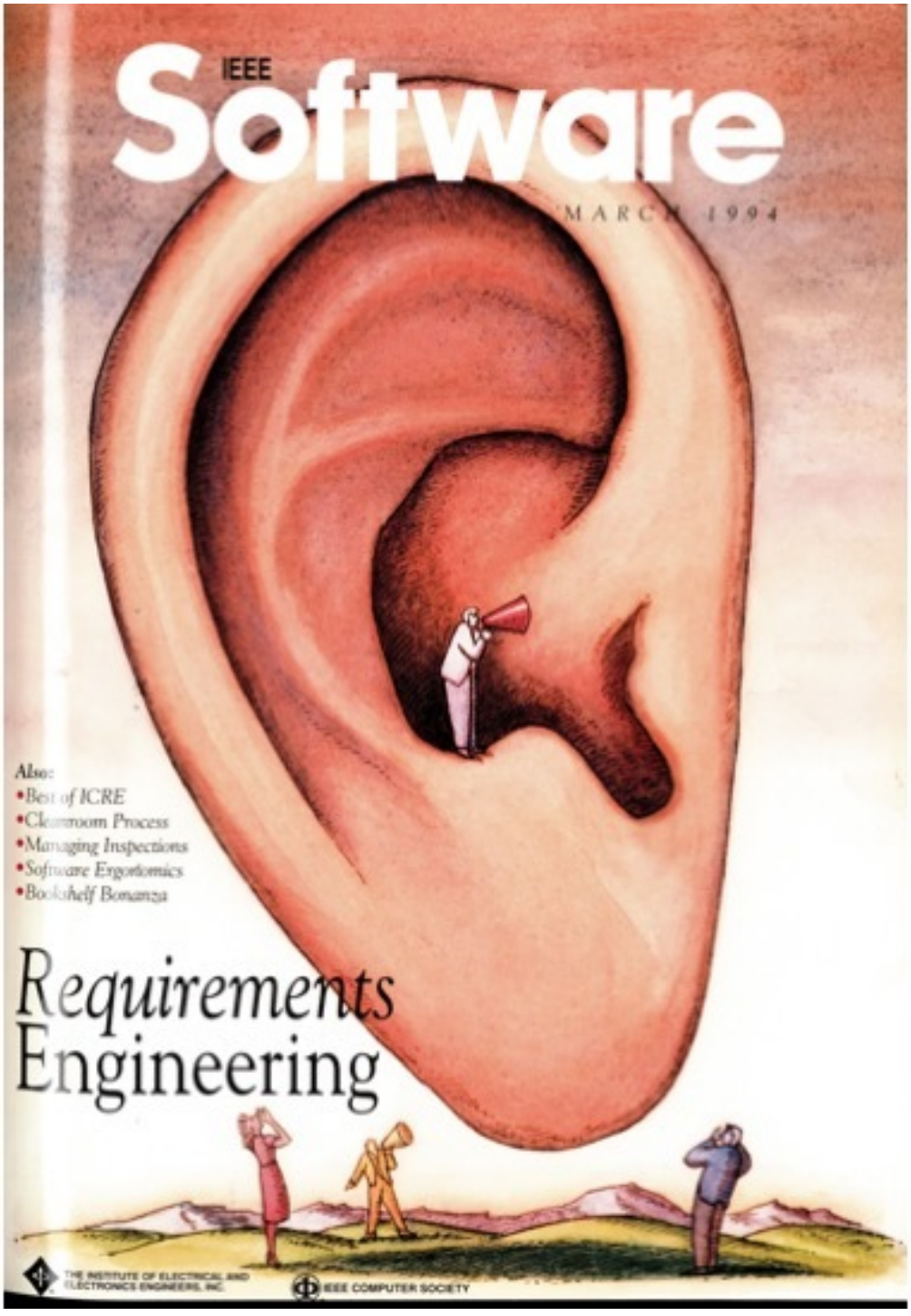
*Dan Craigen, Ted Ralston, Susan Gerhart, **Experience with Formal Methods in Critical Systems**, IEEE Software, January 1994.*[148]

[148] DOI: 10.1109/52.251198

“Darlington is a **four-reactor nuclear plant** east of Toronto. … Each reactor has two independent shutdown systems: SDS1 drops neutron-absorbing rods into the core, while SDS2 injects liquid poison into the moderator. Both are safety-critical and require high levels of confidence. In 1982, Ontario Hydro, with the concurrence of the Atomic Energy Control Board of Canada (AECB), had decided to **fully implement the shutdown systems’ decision-making logic on computers**. This was to be the first Canadian instance of such a system, so there were questions about what procedures to follow, both in developing and licensing the system.”

Dan Craigen, Susan Gerhart, Ted Ralston, ***Case Study: Darlington Nuclear Generating Station****, IEEE Software, January 1994.*[149]

[149] DOI: 10.1109/52.251201


IEEE
Software
MARCH 1994
Also:
•Best of ICRE
•Cleanroom Process
•Managing Inspections
•Software Ergonomics
•Bookshelf Bonanza
Requirements Engineering
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.
IEEE COMPUTER SOCIETY

“Developers have plenty of reasons to avoid investing in requirements engineering: It is **next to impossible to capture user needs** completely, and needs are constantly evolving. The gap between software research and practice is no more evident than in the field of requirements engineering. Requirement engineering has a fairly **narrow goal - determine a need and define the external behavior of a solution** - but the range of research into requirements is enormous.”

Pei Hsia, Alan M. Davis, ***Guest Editors’ Introduction: Giving Voice to Requirements Engineering****, IEEE Software, March 1994.*[150]

[150] DOI: 10.1109/52.268949

“The author examines two widely held beliefs: requirements describe a **system’s ‘what, not its ‘how’**. Requirements must be **represented as abstractions**.”

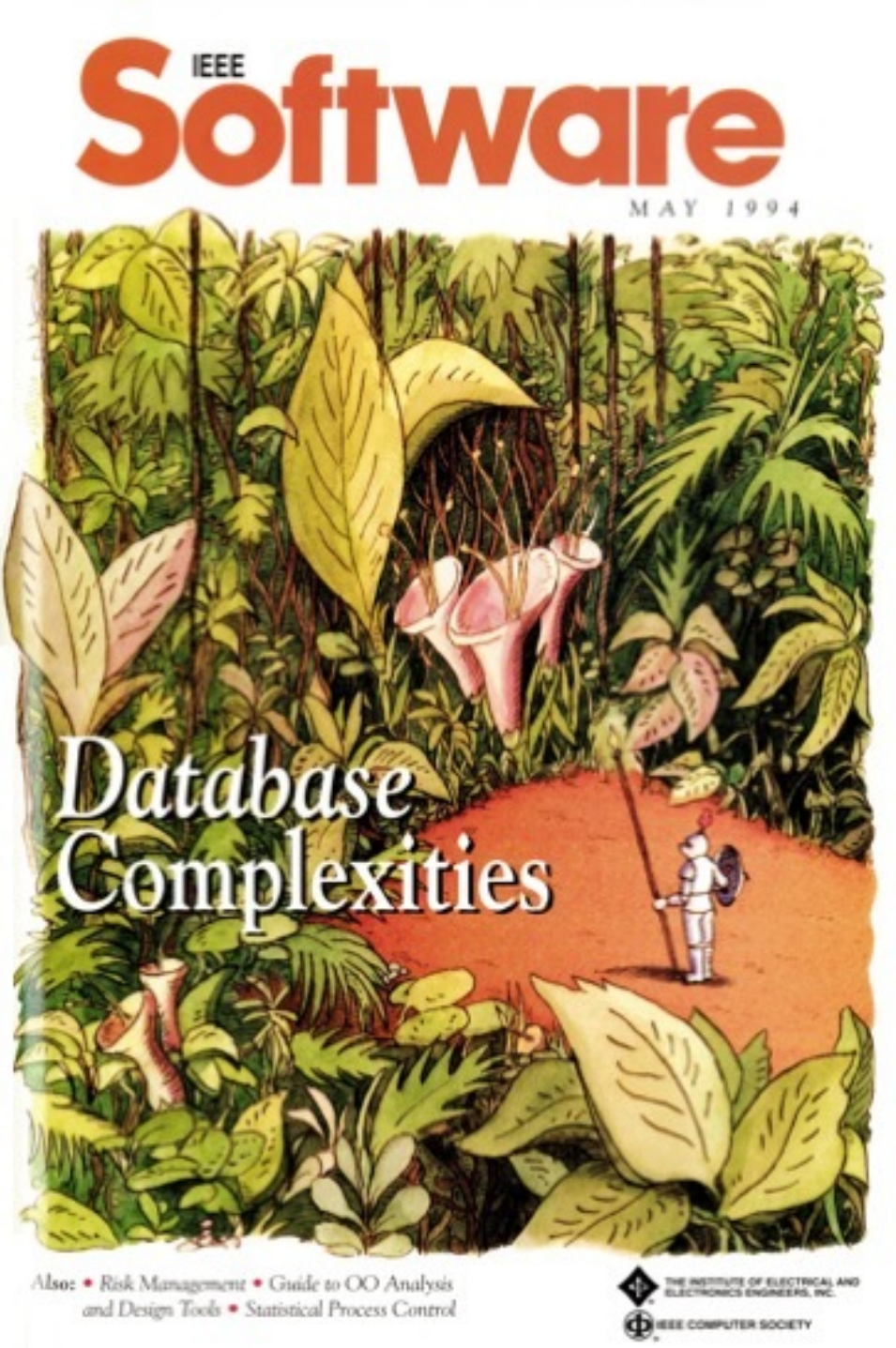
Jawed Siddiqi, ***Challenging Universal Truths of Requirements Engineering****, IEEE Software, March 1994.*[151]

[151] DOI: 10.1109/52.268951

“**Scenarios** offer promise as a way to tame requirements analysis, but progress has been impeded by the lack of a systematic way to analyze, generate, and validate them.”

Jayarajan Samuel, Pei Hsia, David Kung, Jerry Gao, Cris Chen, Yasafumi Toyoshima, ***Formal Approach to Scenario Analysis****, IEEE Software, March 1994.*[152]

[152] DOI: 10.1109/52.268953


IEEE
Software
MAY 1994
Database Complexities
Also: • Risk Management • Guide to OO Analysis and Design Tools • Statistical Process Control
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.
IEEE COMPUTER SOCIETY

“**Database technology** is exploding, as the hierarchical and relational models give way to **object-oriented, distributed heterogeneous**, and other kinds of specialized models. Designers, programmers, and users need new tools.”

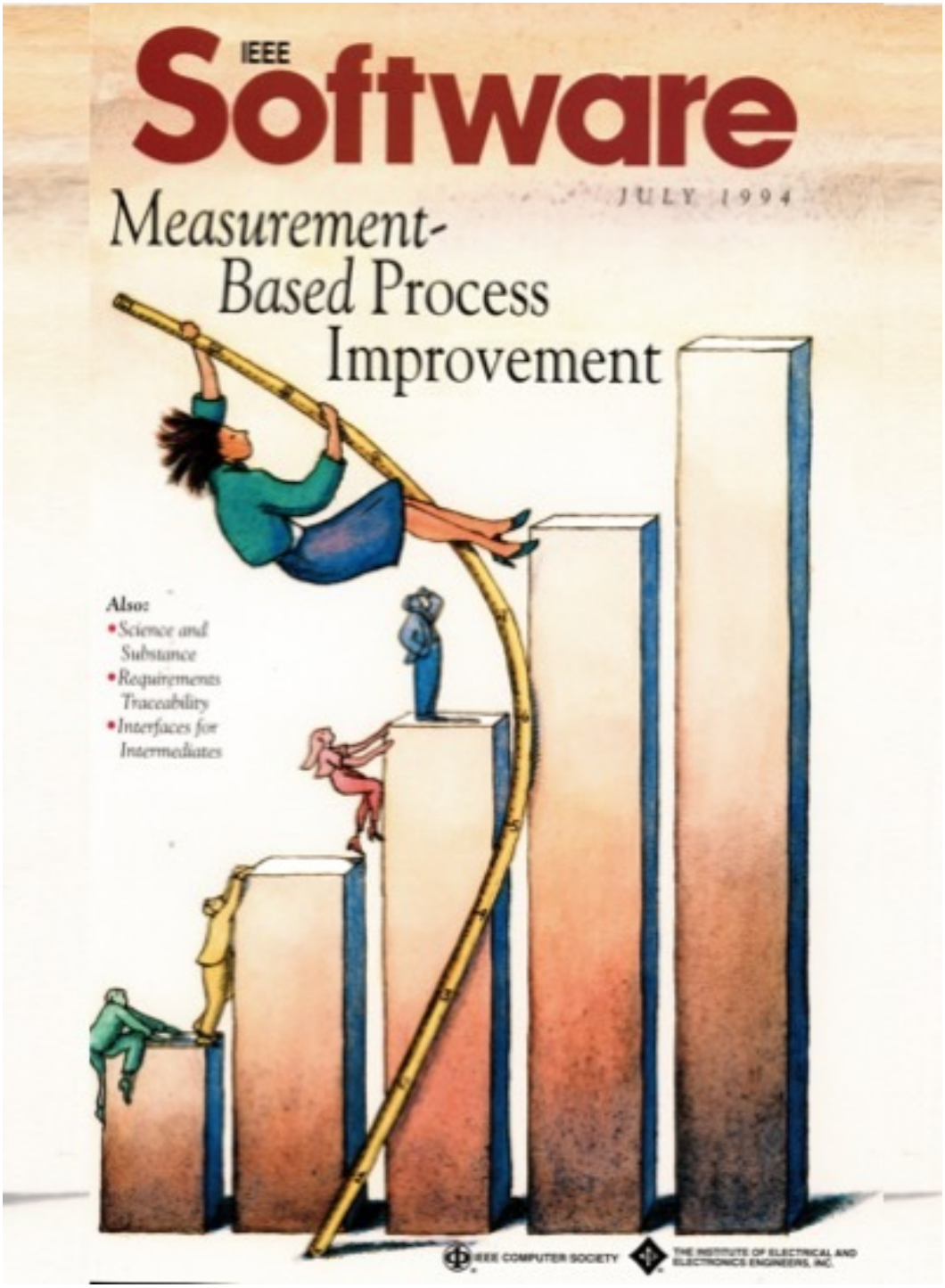
Clement Yu, Weiyi Meng, ***Confronting Database Complexities****, IEEE Software, May 1994.*[153]

[153] DOI: 10.1109/52.281712

“Retrieval speed and precision ultimately determine the success of any database system. … Much work remains to help users retrieve information with ease and efficiency from a heterogeneous environment in which **relational**, **object-oriented**, **textual**, and **pictorial databases** coexist.”

Weiyi Meng, Clement Yu, ***Progress in Database Search Strategies****, IEEE Software, May 1994.*[154]

[154] DOI: 10.1109/52.281713


IEEE
Software
JULY 1994
Measurement-Based Process Improvement
Also:
• Science and Substance
• Requirements Traceability
• Interfaces for Intermediates
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“Used together, the two relatively young concepts of **measurement and process improvement are more than the sum of their parts**. Careful measurement helps you draw an objective process model. Thoughtful application of improvement techniques improves your ability to measure quality. Leveraging one with the other can take your organization to new heights.”

Shari Lawrence Pfleeger, Hans Dieter Rombach, ***Measurement Based Process Improvement****, IEEE Software, July 1994.*[155]

[155] DOI: 10.1109/52.300077

“There are two approaches to process improvement. The **top-down approach** compares an organization’s process with some generally accepted standard process. … The **bottom-up approach** assumes that process change must be driven by an organization’s goals, characteristics, product attributes, and experiences.”

Frank McGarry, Martyn Thomas, ***Top-Down vs. Bottom-Up Process Improvement****, IEEE Software, July 1994.*[156]

[156] DOI: 10.1109/52.300121

“In their efforts to determine how technology affects the software development process, **researchers often overlook organizational and social issues**.”

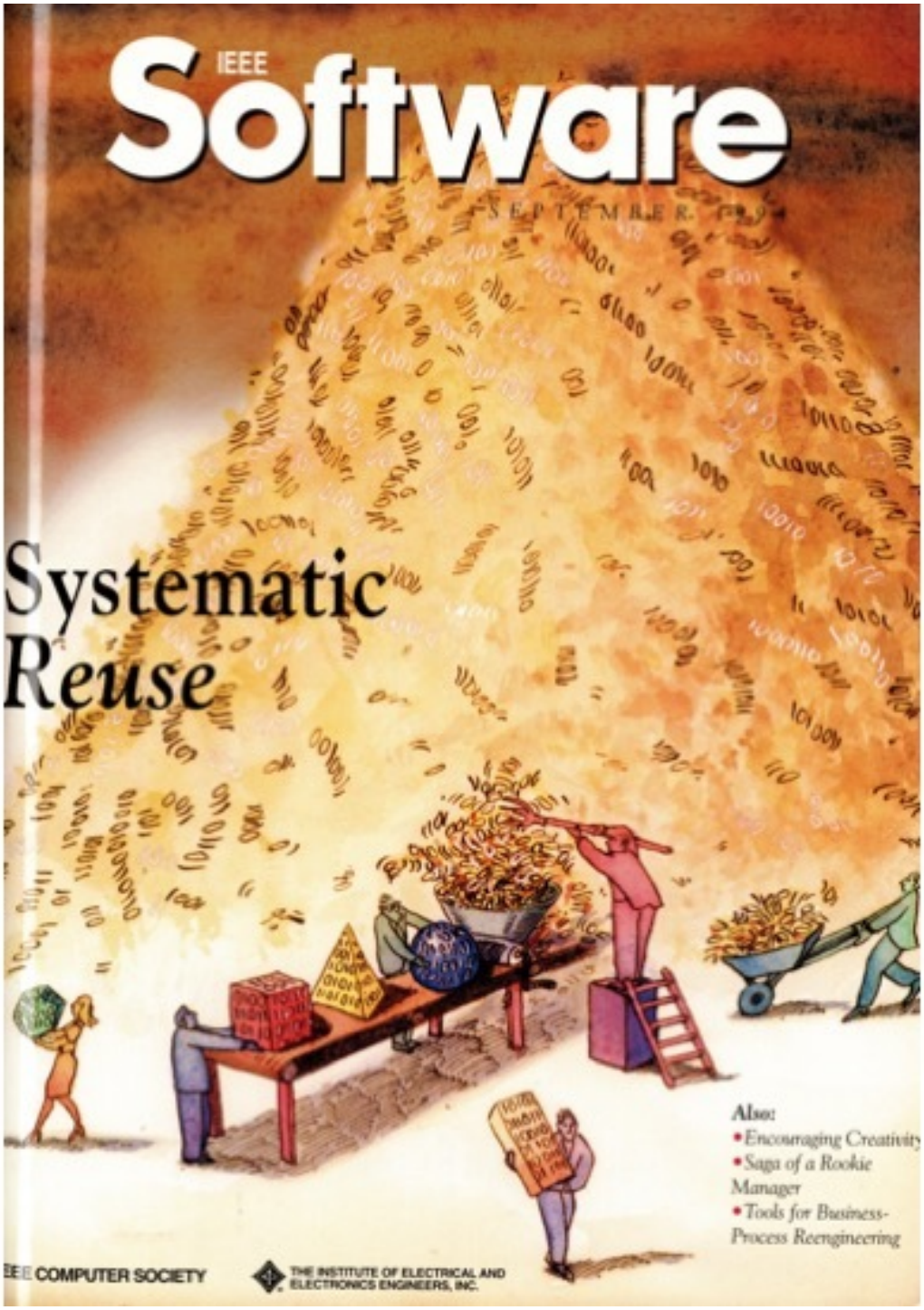
Nancy A. Staudenmayer, Dewayne E. Perry, Lawrence G. Votta, ***People, Organizations, and Process Improvement****, IEEE Software, July 1994.*[157]

[157] DOI: 10.1109/52.300082

“For 25 years, software researchers have proposed improving software development and maintenance with new practices whose **effectiveness is rarely, if ever, backed up by hard evidence**. We suggest several ways to address the problem, and we challenge the community to **invest in being more scientific**.”

Robert L. Glass, Shari Lawrence Pfleeger, Norman Fenton, ***Science and Substance: A Challenge to Software Engineers****, IEEE Software, July 1994.*[158]

[158] DOI: 10.1109/52.300094


IEEE
Software
SEPTEMBER
Systematic
Reuse
Also:
• Encouraging Creativity
• Saga of a Rookie Manager
• Tools for Business-Process Reengineering
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

"**Systematic software reuse** is a paradigm shift in software engineering from building single systems to **building families of related systems**. The goal of software reuse research is to discover systematic procedures for engineering new systems from existing assets. Implementing systematic reuse is risky. Not doing it is also risky."

William B. Frakes, Sadahiro Isoda, ***Success Factors of Systematic Reuse****, IEEE Software, September 1994.*[159]

[159] DOI: 10.1109/52.311045

"Reuse is **not just a technical issue**. Hewlett-Packard studied why people sometimes **resist reuse** and which organizational models appear to encourage reuse more than others. … successful reuse programs must be **integrated within the culture** of a company's existing organizational structure. One crucial organizational factor is the **relationship between producers and consumers** of reuse components and services."

---

*Danielle Fafchamps, **Organizational Factors and Reuse**, IEEE Software, September 1994.*[160]

[160] DOI: 10.1109/52.311049

IEEE
Software
NOVEMBER 1994

Global Vision

Also:
- *Dynamic Queries Interface*
- *Surviving the Coming Shakeout*
- *Fifteen Principles of Software Engineering*

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“The worldwide software industry is poised for change well into the next century. How well each developer; researcher, or country fares may depend on how clear its **visions of the future** are.”

---

*Jawed Siddiqi, Mikio Aoyama, William W. Everett, **Software Beyond 2001: A Global Vision**, IEEE Software, November 1994.*[161]

[161] DOI: 10.1109/52.329394

“Many **developing countries** are now **entering the commercial software domain**, and this trend should accelerate in the twenty-first century.”

Capers Jones, ***Globalization of Software Supply and Demand***, *IEEE Software, November 1994.*[162]

[162] DOI: 10.1109/52.329397

“Who wants to **try new techniques** when there are **madman prowling for someone to blame or ax**?”

Tom DeMarco, Sheila Brady, ***Management-Aided Software Engineering****, IEEE Software, November 1994.*[163]

[163] DOI: 10.1109/52.329398

“**Iincreasing connectivity** and **consumer demands** will power an unprecedented **growth in software’s volume and complexity** … the flexibility and robustness of **an object-oriented approach** can best meet these future challenges.”

Grady Booch, ***Coming of Age in an Object-Oriented World***, *IEEE Software*, *November 1994.*[164]

[164] DOI: 10.1109/52.329399

"The twentieth century was a **time of ignorance**, but also the **dawn of golden age of practice**."

---

*Robert L. Glass, **The Software-Research Crisis**, IEEE Software, November 1994.*[165]

[165] DOI: 10.1109/52.329400

“The large aspiration to place the whole of **software development** alongside the **established branches** as one more branch of engineering is **misconceived**. … Our aspiration should be to **develop specialized branches of software** engineering, each meriting its own place alongside the specialized established branches.”

---

*Michael Jackson, **Problems, Methods and Specialization**, IEEE Software, November 1994.*[166]

[166] DOI: 10.1109/52.329402

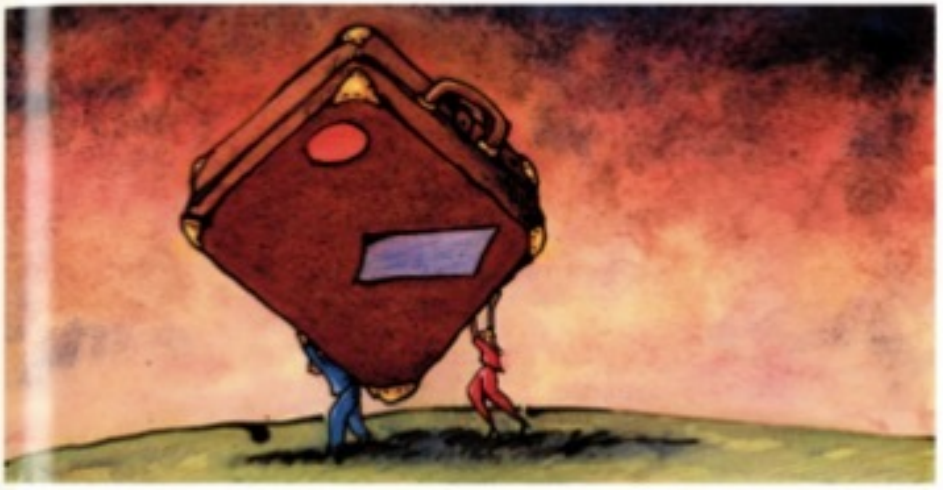
"For data in which there is a **known relationship among variables**, the dynamic queries interface is useful for training and **education by exploration**. For situations in which there are understood correlations, but their complexity makes it difficult for nonexperts to follow, dynamic queries can allow a wider range of people to explore the interactions (among health and demographic variables, a table of elements, and economic or market data, for example)."

Ben Schneiderman, ***Dynamic Queries for Visual Information Seeking****, IEEE Software, November 1994.*[167]

[167] DOI: 10.1109/52.329404

1995

IEEE
Software
JANUARY 1995



Legacy Systems

also:

lessons from rapid prototyping

Machiavelli on software development

how ISO and CMM overlap

enterprise integration and people

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

"**Legacy software** was written years ago using outdated techniques, yet it continues to do useful work. Migrating and updating this **baggage from our past** has technical and nontechnical challenges, ranging from justifying the expense to dealing with offshore contractors to using program-understanding and visualization techniques."

Keith Bennett, ***Legacy Systems: Coping with Success****, IEEE Software, January 1995.*[168]

[168] DOI: 10.1109/52.363157

“As the manager of a small software-reengineering company, I am continually confronted with the task of **justifying reengineering**. … they **technical issues may be irrelevant** if you are not able to **make a business case** for solving them.”

Harry M. Sneed, ***Planning the Reengineering of Legacy Systems***, *IEEE Software, January 1995.*[169]

[169] DOI: 10.1109/52.363168

“Opinions on **rapid prototyping** as a practical development tool vary widely, with conventional wisdom seeing it more as a research topic than a workable method. The authors counter this notion with results from 39 case studies, most of which have used this approach successfully.”

James M. Bieman, V. Scott Gordon, ***Rapid Prototyping: Lessons Learned****, IEEE Software, January 1995.*[170]

[170] DOI: 10.1109/52.363162

"**Usability engineering** isn't just for the multimillion dollar companies with massive internal test labs. Jakob Nielsen, a distinguished engineer at SunSoft, relates how he and another designer (yes, a two-person project) employed low-cost, easily accessible techniques to perform several useful studies. The techniques, detailed in his recent book Usability Engineering (AP Professional, 1994), are virtually free of complex statistical methods, **relying instead on simple observation and interpretation**."

Bill Curtis, Jakob Nielsen, ***Applying Discount Usability Engineering****, IEEE Software, January 1995.*[171]

[171] DOI: 10.1109/52.363161

IEEE
Software
MARCH 1995

Development Tools

also:

calling all heroes

helping users help themselves

debut of ISO's Ada 95

taligent's new paradigm

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“**Tools** are not the driving force of software technology they used to be. But this does not mean they have become less important; on the contrary, a full set of supporting tools for each development phase is now a basic requirement. It does mean **other things are now as important as tools**…”

Pertti Lounama, ***The Future Belongs to the Specialized Tool****, IEEE Software, March 1995.*[172]

[172] DOI: 10.1109/MS.1995.10010

“Computer technology is **exploding**. Friendly interfaces, a full range of media, and unlimited connectivity are pushing technology ever deeper into the fabric of society. How can developers incorporate all this new technology into their applications? Today’s development environments are taking advantage of emerging integration standards to offer specialized tools.”

David Sharon, Rodney Bell, ***Tools to Engineer New Technologies into Applications****, IEEE Software, March 1995.*[173]

[173] DOI: 10.1109/52.368255

“Point: Alan Chmura — Proper use of **CASE tools can and does significantly improve developer productivity** and yield high-quality systems. … Counterpoint: Henry David Crockett — As they are currently applied, **CASE tools seldom improve productivity or quality**. … Many IS managers assume that CASE is a software solution to all development problems.”

Alan Chmura, Henry David Crockett, ***Point Counterpoint: What’s the Proper Role for CASE Tools?****, IEEE Software, March 1995.*[174]

[174] DOI: 10.1109/52.368258

“In many applications users must **browse large images**. Building on user **familiarity with one-dimensional scroll bars**, many designers simply use **two one-dimensional scroll bars** when the application requires independent control over the horizontal and vertical directions, **as in panning a map**. … in many cases this solution is inadequate.”

Ben Shneiderman, Catherine Plaisant, David Carr,

***Image-Browser Taxonomy and Guidelines for Designers**,*

IEEE Software, March 1995.[175]

[175] DOI: 10.1109/52.368260

“**Sound** can potentially **reveal patterns and anomalies in data** that are difficult to perceive visually. Moreover, **psychological studies** show that some types of data are more quickly assimilated when presented with sound; **an audio alarm** is the classic example.”

Daniel A. Reed, Tara M. Madhyastha, ***Data Sonification: Do You See What I Hear?****, IEEE Software, March 1995.*[176]

[176] DOI: 10.1109/52.368264

IEEE Software
MAY 1995

Reliable Systems

also:

home page design

good enough software

object communication

experimental science and software

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“Not a day goes by that the **general public** does not come into **contact with a real-time system**. As their numbers and importance grow, so do the **implications** for software **developers**.”

William W. Everett, Shinichi Honiden, ***Guest Editors’ Introduction: Reliability and Safety of Real-Time Systems****, IEEE Software, May 1995.*[177]

[177] DOI: 10.1109/52.382177

"**Software verification** is often the last defense against disasters caused by faulty software development. When lives and fortunes depend on software, **software quality** and its verification demand increased attention. … How do you **assess** that critical automated systems are acceptably **safe and reliable**?"

Keith W. Miller, Jeffrey M. Voas, ***Software Testability: The New Verification****, IEEE Software, May 1995.*[178]

[178] DOI: 10.1109/52.382180

“**The best prototype** for designing your new user interface is **your old user interface**. **The second-best prototype** is a **competing product**. Your competitors have invested significant resources in designing and implementing what they believe to be good user interfaces. You should take advantage of those investments. “

Jakob Nielsen, ***A Home-Page Overhaul Using Other Web Sites***, *IEEE Software, May 1995.*[179]

[179] DOI: 10.1109/52.382190

IEEE
Software
JULY 1995

That's Debatable!

also:
product liability
rejecting the lie
visible quality

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“There is an **enormous disconnection** between what **academicians and consultants** think will revolutionize development and **what actually works in the trenches**. We have much to learn about what really works, what works a little, and what doesn’t work at all. **Debating issues** over time can help us **accumulate knowledge** and **reach a consensus** on the most promising solutions.”

Stephen J. Andriole, ***Debatable Development: What Should We Believe?****, IEEE Software, July 1995.*[180]

[180] DOI: 10.1109/MS.1995.10034

"**Systems analysis** is the study of a system for the purpose of **understanding and documenting** its **essential characteristics**. Analysis is **neither design nor implementation**. Analysis focuses on real-world problems, whereas design and implementation focus on computerized solutions."

Robert B. Jackson, David W. Embley, Scott N. Woodfield, ***OO Systems Analysis: Is It or Isn't It?****, IEEE Software, July 1995.*[181]

[181] DOI: 10.1109/52.391825

"Many **nonformalists** seem to believe that formal methods are **merely an academic exercise** – a form of **mental masturbation** that has no relation to real-world problems. … We address and dispel seven new myths about formal methods …"

Jonathan P. Bowen, Michael G. Hinchey, ***Seven More Myths of Formal Methods****, IEEE Software, July 1995.*[182]

[182] DOI: 10.1109/52.391826

“We redesigned the Sun home page after observing users … One change we made shows the **benefits of this type of testing**: We discovered early on that users didn’t recognize the “What’s New at Sun” button as a link, and … we redesigned the button immediately … use of the ‘What’s New at Sun’ button **increased by 416 percent** … **small interface changes** can lead to **dramatic changes in user behavior**! “

Jakob Nielsen, ***Using Paper Prototypes In Home-page Design****, IEEE Software, July 1995.*[183]

[183] DOI: 10.1109/52.391840

IEEE Software
SEPTEMBER 1995

Rapid Application Development

also:

Mythical Man-Month revisited

interactive information retrieval

capitalizing on multiple markets

AI-based game algorithm

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“In our haste to **speed development**, we should consider if we are sprinting in the right direction. Is faster time-to-market equally important to every developer? Does a **faster cycle time** necessarily guarantee success? And how does the answer change from one year to the next?”

David N. Card, ***Guest Editor’s Introduction: The RAD Fad–Is Timing Really Everything?****, IEEE Software, September 1995.*[184]

[184] DOI: 10.1109/MS.1995.10045

“To be successful with RAD (**rapid application development**), we can no longer look just at the software product in isolation. A RAD development process demands that we expand our view to **encompass users** and their work environments-and if we do it right, everyone benefits. But given all the risks, would you stake **mission-critical projects on RAD**? If it is not mission-critical, then why bother at all?”

Erran Carmel, John P. Reilly, ***Point-Counterpoint: Does RAD Live Up to the Hype?****, IEEE Software, September 1995.*[185]

[185] DOI: 10.1109/52.406752

“Earl Wheeler told me …: ‘The key thrust … was delegating power down. It was likemagic! **Improved quality**, **productivity**, **morale**. We have **small teams**, with **no central control**. The **teams own the process**, but they have to haveone. They have many different processes. They **own the schedule**, but they **feel the pressure** of the market. This pressure causes them to reach for tools on their own’.”

Frederick P. Brooks Jr., ***The Mythical Man-Month: After 20 Years***, *IEEE Software, September 1995.*[186]

[186] DOI: 10.1109/MS.1995.10041

"The **Mythical Man-Month** is **only incidentally about software** but **primarily about how people in teams make things**.There is surely some truth in this; in the preface to the 1975 edition I said that managing a software project is more like other management than most programmers initially believe. I still believe that to be true. Human history is a drama in which the stories stay the same, the scripts of those stories change slowly with evolving cultures, and the stage settings change all the time. So it is that we see our twentieth-century selves mirrored in Shakespeare, Homer,and the Bible."

Frederick P. Brooks Jr., ***The Mythical Man-Month: After 20 Years****, IEEE Software, September 1995.*[187]

[187] DOI: 10.1109/MS.1995.10041

“At **Hitachi Software**, we organize software projects in a way that retains high-quality software and improves scheduling. We do this by **forcing ‘necessary’ conflicts** among independent groups within the larger software-development team. We believe that by creating **a competitive atmosphere** between **the design and quality-assurance departments**, engineers on both teams are motivated to be quality sensitive.”

Akira K. Onoma, Tsuneo Yamaura, ***Practical Steps Toward Quality Development****, IEEE Software, September 1995.*[188]

[188] DOI: 10.1109/52.406760

“What are the three most important things to remember when you buy a house? Most real estate agents have a simple answer: **location**, **location**, **location**. And what are the three most important things to remember when attempting software-process improvement? My answer: **people**, **people**, **people**.”

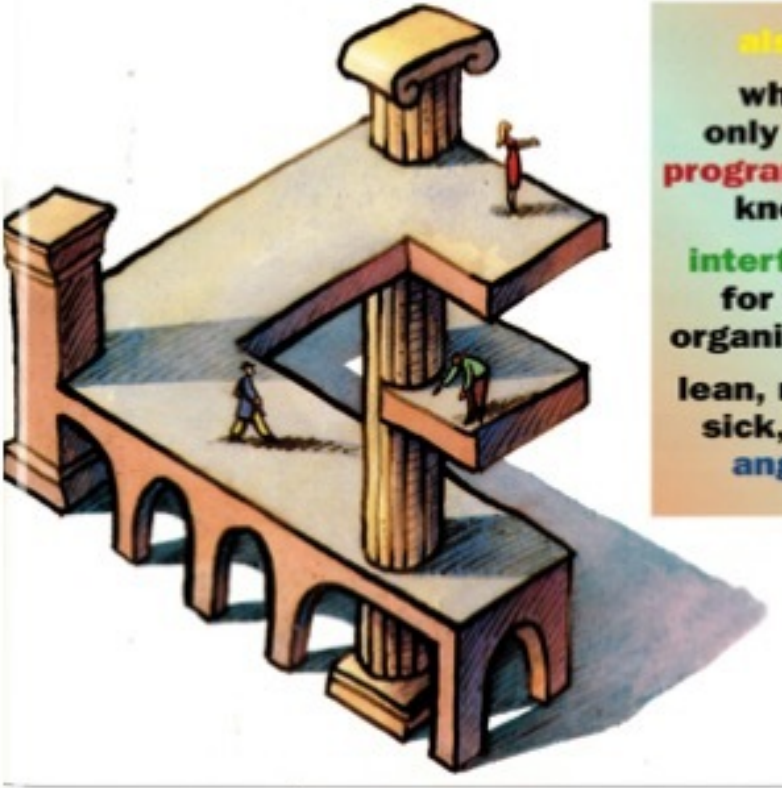
---

*Watts S. Humphrey, **Making Process Improvement Personal**, IEEE Software, September 1995.*[189]

[189] DOI: 10.1109/52.406762

IEEE
Software
NOVEMBER 1995

Architecture



also:

what only real programmers know

interfaces for the organization

lean, mean, sick, and angry

IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“There is undoubtedly a large measure of **art involved in software design**. But **artistic expression** in the absence of rules results in **chaotic design**. To produce open systems, we must agree on some **well-defined rules** to govern interaction among systems and subsystems.”

Maarten Boasson, ***The Artistry of Software Architecture****, IEEE Software, November 1995.*[190]

[190] DOI: 10.1109/MS.1995.10051

“Most of the **designs** appeal to **multiple styles**, but they generally fall into four main groups: **object-oriented** architectures, including information hiding; **state-based** architectures; **feedback-control** architectures; and architectures that emphasize the system’s **real-time** properties.”

---

*Mary Shaw, **Comparing Architectural Design Styles**, IEEE Software, November 1995.*[191]

[191] DOI: 10.1109/52.469758

"**The 4+1 View Model** describes software architecture using **five concurrent views** … : The **logical view** describes the design's object model, the **process view** describes the design's concurrency and synchronization aspects; the **physical view** describes the mapping of the software onto the hardware and shows the system's distributed aspects, and the **development view** describes the software's static organization in the development environment. Software designers can organize the description of their **architectural decisions** around these four views and then illustrate them with a few selected use cases, or **scenarios**, which constitute **a fifth view**. "

Philippe Kruchten, ***The 4+1 View Model of Architecture****, IEEE Software, November 1995.*[192]

[192] DOI: 10.1109/52.469759

“I get uncontrollable giggles when people tell me their organizations are ‘**lean and mean**.’ They say it in the most ponderous tones. They wrinkle their brows earnestly and look me right in the eye. ‘We’re **lean and mean** here,’ they say. They say this even though they **themselves are overweight and rather sweet**. And that’s only the first of the contradictions.”

---

*Tom DeMarco, **What ‘Lean and Mean’ Really Means**, IEEE Software, November 1995.*[193]

[193] DOI: 10.1109/52.469767

1996

IEEE
Software
JANUARY 1996

Quality

also:

getting to level 6

beyond the black box

who cares about code?

data visualization

IEEE COMPUTER SOCIETY 50 YEARS OF SERVICE 1946-1996

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

"The **consequences of Moore's Law** for IEEE Software are significant. In short, in the coming decades, we will see technological and market niches that constantly form, shift, merge, split, and disappear. **Computers** will **permeate most of our buildings** (homes and offices), **cars**, **wallets**, **watches**, maps, and credit cards. Existing devices (faxes, phones, PCs, TVs, GPSs, pagers, PDAs, and the like) will merge into forms that are specialized to particular uses (the so-called 'convergence revolution')."

Ted J. Biggerstaff, ***Moore's Law: Change or Die****, IEEE Software, January 1996.*[194]

[194] DOI: 10.1109/MS.1996.476277

“If you are a software developer, manager, or maintainer, quality is often on your mind. But **what do you really mean by software quality**? Is your definition adequate? Is the software you produce better or worse than you would like it to be?”

Shari Lawrence Pfleeger, Barbara Kitchenham, ***Software Quality: The Elusive Target****, IEEE Software, January 1996.*[195]

[195] DOI: 10.1109/52.476281

“Discussions of **software quality** typically **focus on** the development **process** or the characteristics of the software **product**. The third level of quality - **the outcome** of software development - is usually neglected, although this is perhaps of greatest interest to business management. **Outcome** depends on how the product is used and determines **the business value** obtained from the development project.”

Pamela Simmons, ***Quality Outcomes: Determining Business Value****, IEEE Software, January 1996.*[196]

[196] DOI: 10.1109/52.476283

“Today the dominant modus operandi for software development is **heavily process-oriented** … the **emphasis on process** … comes at the expense of … using adequate **product quality models**. The fundamental axiom of software product quality is: … tangible **internal characteristics** … determine its external quality attributes. … A **product quality model** … must … identify the tangible (measurable and/or assessable) **internal** product **characteristics** that have the most significant effect on external quality attributes. “

R. Geoff Dromey, ***Cornering the Chimera***, *IEEE Software*, *January 1996.*[197]

[197] DOI: 10.1109/52.476284

"In 1991, **Philips' CEO** named a **Software Process Improvement task force** to focus on the increasing importance of software … In addition to improving its processes, the organization improved its requirements-and-design engineering architecture and its **inspections**, and it **introduced metrics**."

Hans Aerts, Jan Rooijmans, Michiel Van Genuchten, ***Software Quality in Consumer Electronics Products****, IEEE Software, January 1996.*[198]

[198] DOI: 10.1109/52.476286

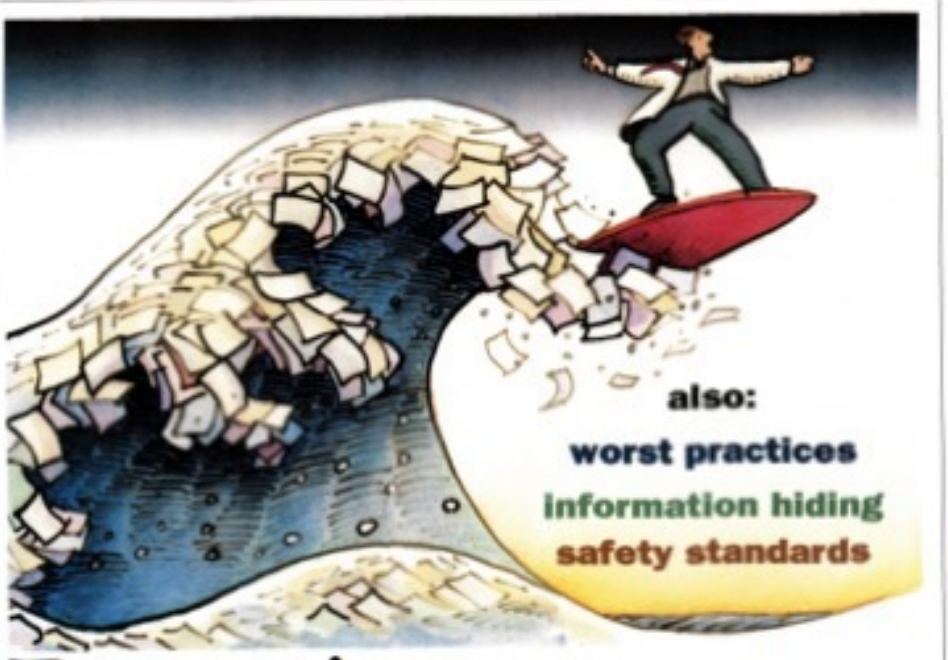
"The **need to be beautiful** and produce more elaborate and **polished designs** changes our traditionally restrained and functionality-oriented design focus."

---

*Jakob Nielsen, **The Importance of Being Beautiful**, IEEE Software, January 1996.*[199]

[199] DOI: 10.1109/52.476290

IEEE
Software
MARCH 1996



Requirements Engineering

IEEE COMPUTER SOCIETY
50 YEARS OF SERVICE 1946-1996
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“Developments in **requirements engineering**, as in system development, have come in waves. The next wave of requirements techniques and tools should account for the problem and development **context**, accommodate **incompleteness**, and recognize and exploit the **non-absolute nature of user needs**.”

---

*Jawed Siddiqi, M. Chandra Shekaran, **Requirements Engineering: The Emerging Wisdom**, IEEE Software, March 1996.*[200]

[200] DOI: 10.1109/MS.1996.506458

"To find the **right balance** of **quality-attribute requirements**, you must **identify the conflicts** among desired quality attributes and **work out a balance** of attribute satisfaction."

Hoh In, Barry Boehm, ***Identifying Quality-Requirement Conflicts****, IEEE Software, March 1996.*[201]

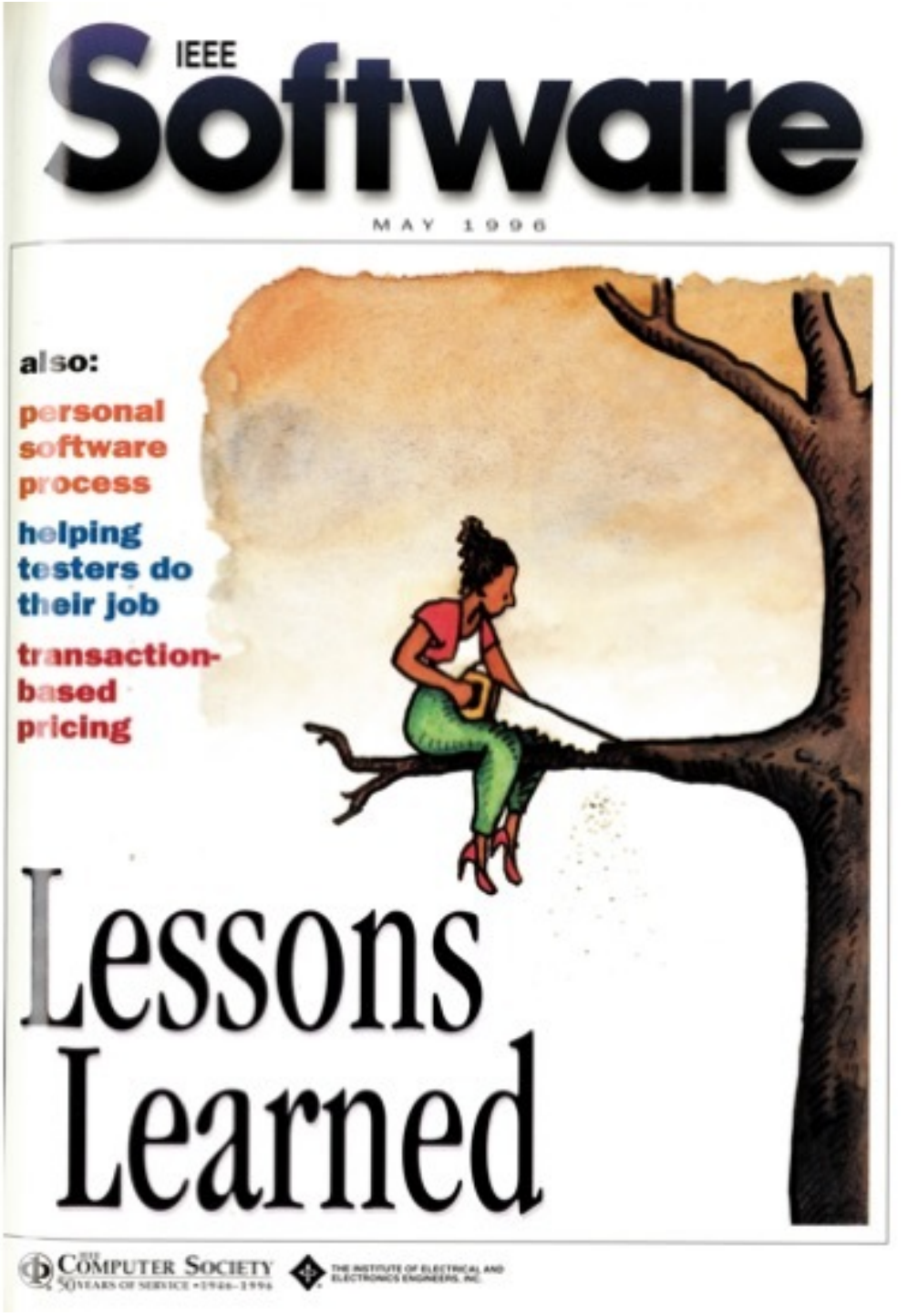
[201] DOI: 10.1109/52.506460

“A **lack of solid historical data** makes project managers, executives, and clients **blind to the realities** of software development.”

---

*Capers Jones, **Our Worst Current Development Practices**, IEEE Software, March 1996.*[202]

[202] DOI: 10.1109/52.506467


IEEE
Software
MAY 1996
also:
personal software process
helping testers do their job
transaction-based pricing
Lessons Learned
IEEE COMPUTER SOCIETY
50 YEARS OF SERVICE • 1946–1996
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“Software engineering is a young profession. We still have much to learn from each other. Past experience suggests that an **incremental delivery model** would **reduce risk** and **improve project management**.”

Pei Hsia, ***Making Software Development Visible***, *IEEE Software*, *May 1996.*[203]

[203] DOI: 10.1109/MS.1996.493016

“If you’re **building a relatively complex system** involving multiple computers and multiple users, and if the system entails significant innovation - such as new technology or expanded scale - something will **inevitably go wrong**. Realizing this might encourage you to use both design and user-interface prototypes?,’and the spiral model of development so that you can look ahead and assess risks as you go.”

Karen Mackey, ***Why Bad Things Happen to Good Projects***, *IEEE Software, May 1996.*[204]

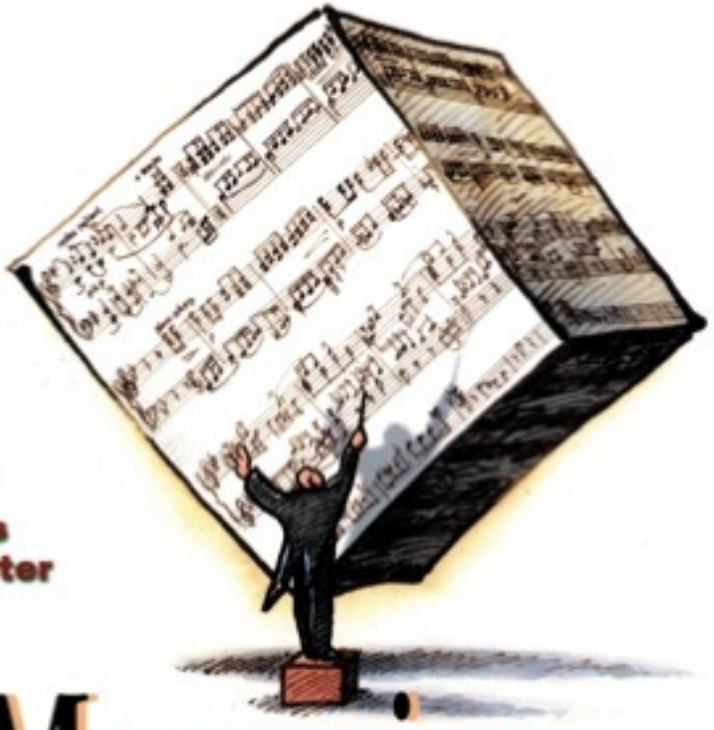
[204] DOI: 10.1109/52.493017

"**Product-line development** seeks to achieve reuse across a domain, or family, of systems. Product-line development separates the software-development process into two separate life cycles: **domain engineering**, which aims to create reusable assets, and **application engineering**, which fields systems using those assets. … We learned that product-line development demands **careful strategic planning**, a **mature** development **process**, and the ability to overcome **organizational resistance**."

Lynn D. Stuckey Jr., David C. Gross, Randall R. Macala, ***Managing Domain-Specific, Product-Line Development****, IEEE Software, May 1996.*[205]

[205] DOI: 10.1109/52.493021

IEEE
Software
JULY 1996



also:
beyond blaming
9 best practices
questions of character

Managing Megaprojects

IEEE COMPUTER SOCIETY
50 YEARS OF SERVICE • 1946-1996
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

“**Thirty years ago**, as the software industry was first gathering steam, most software projects were **run by people** with **no software experience**. Today this is **no longer true**. … Yet few would **argue** that the quality of software project management **has improved** in the same period.”

Ann Miller, Tom DeMarco, ***Managing Large Software Projects****, IEEE Software, July 1996.*[206]

[206] DOI: 10.1109/MS.1996.526827

"Organizations have invested in **dozens of technological innovations** such as **fourth-generation** languages, **CASE** products, **object-oriented** analysis and programming, and software **reuse**. Yet productivity tools simply **aren't delivering increased productivity** even when a project is managed 'by the book.' … the software development environment is **a complex social system** that causes such practices to have unintended consequences."

---

*Tarek K. Abdel-Hamid, **The Slippery Path to Productivity Improvement**, IEEE Software, July 1996.*[207]

[207] DOI: 10.1109/52.526831

"The most important lesson we have learned is to **follow up with suppliers** so that they **know** the **importance we place** on **quality** and process improvement."

Jim Nielsen, Ann Miller, ***Selecting Software Subcontractors****, IEEE Software, July 1996.*[208]

[208] DOI: 10.1109/52.526837

“Only through **experimentation** can **true learning**, and hence progress, take place. However, it should be remembered that one **definition of insanity** is when a person, failing at a task, **tries the same thing** over and over again, **expecting a different result**.”

Robert N. Charette, ***Large-Scale Project Management Is Risk Management****, IEEE Software, July 1996.*[209]

[209] DOI: 10.1109/52.526838

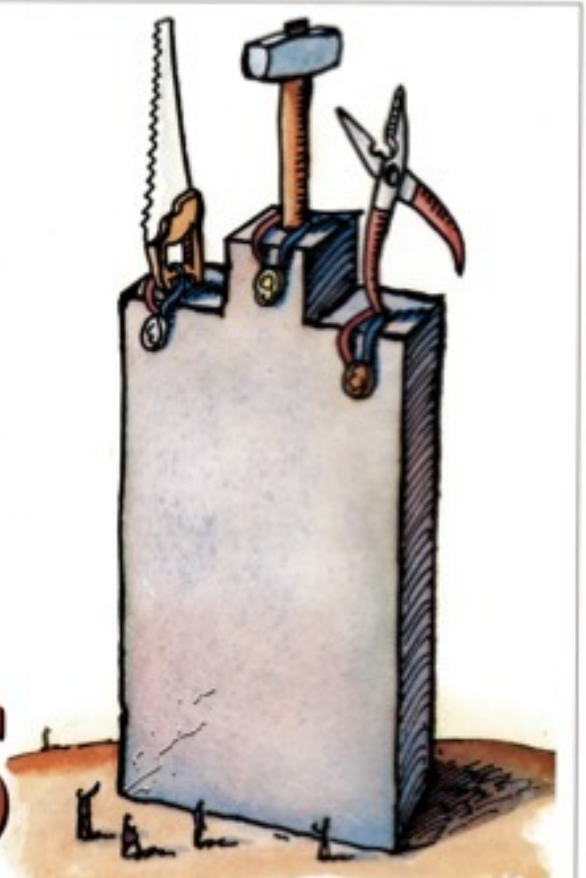
IEEE Software
SEPTEMBER 1996

also:

tools
fair

software
process
impediments

seductive
interfaces



Tools Assessment

IEEE COMPUTER SOCIETY
50 YEARS OF SERVICE • 1946-1996

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

"**Software tools** and new **software methodologies** can together play a key role in achieving a higher level of **software quality and productivity**. Software quality can be greatly improved by selecting a **correct development tool** … Selecting an **inappropriate tool**, on the other hand, can actually hinder software development."

Ez Nahouraii, Krishna Kavi, ***Guest Editors' Introduction: Software Tools Assessment****, IEEE Software, September 1996.*[210]

[210] DOI: 10.1109/MS.1996.536455

"Because we don't know **how to analyze a tool's impact** on specific projects, we generally adopt them based on an **intuitive understanding** of their expected impact. In many cases, the actual **results** of this practice are **disappointing**. The problem is aggravated because tool adoption often brings considerable costs."

John Henshaw, Ingrid Janssen, Nazim H. Madhavji, Tilmann Bruckhaus, ***The Impact of Tools on Software Productivity****, IEEE Software, September 1996.*[211]

[211] DOI: 10.1109/52.536456

“The organization attempts to understand and balance **competing concerns** regarding the **new technology**. These concerns include **acquisition costs**, the technology’s effect on quality and **time to market**, and the **training** and **support** services it will require. “

Alan W. Brown, Kurt C. Wallnau, ***A Framework for Evaluating Software Technology****, IEEE Software, September 1996.*[212]

[212] DOI: 10.1109/52.536457

“**Migrating legacy systems** and developing new systems for **client/server environments** has dominated the software development **tool market in the ’90s**.”

Alan Chmura, David Sharon, ***Tools Fair: Untangling the Web with Web and Client/Server Development Tools****, IEEE Software, September 1996.*[213]

[213] DOI: 10.1109/52.536460

IEEE
Software
NOVEMBER 1996

Toward a Discipline

also:

usability metrics

crossing cultures

software scenarios

IEEE COMPUTER SOCIETY 50 YEARS OF SERVICE • 1946–1996

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

"**Quantitative data** makes sense when you are trying to decide between **two or more alternatives**."

Jakob Nielsen, ***Usability Metrics: Tracking Interface Improvements***, *IEEE Software*, *November 1996.*[214]

[214] DOI: 10.1109/MS.1996.10031

“Despite rapid changes in computing and software development, some **fundamental ideas have remained constant**. … Eight such concepts together constitute a viable **foundation** for a **software engineering discipline**: **abstraction**, analysis and design **methods and notations**, user interface **prototyping**, **modularity** and architecture, software **life cycle** and **process**, **reuse**, **metrics**, and **automated support**.”

---

*Anthony I. Wasserman, **Toward a Discipline of Software Engineering**, IEEE Software, November 1996.*[215]

[215] DOI: 10.1109/52.542291

“Working in and **designing for other cultures** can lead to **communication breakdowns**.”

Kumiyo Nakakoji, ***Beyond Language Translation: Crossing the Cultural Divide***, *IEEE Software, November 1996.*[216]

[216] DOI: 10.1109/52.542293

"It's difficult to determine event order in **distributed systems** because of the **problem of observability**."

Colin Fidge, ***Fundamentals of Distributed System Observation****, IEEE Software, November 1996.*[217]

[217] DOI: 10.1109/52.542297

1997

IEEE Software
JANUARY/FEBRUARY 1997

William Davidow
Interview, p. 38

Object Methods, Patterns, and Architectures

Also:

Measuring Intangibles

Friendlier DIV

Implementing CM

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.
IEEE COMPUTER SOCIETY

“**Security demands a rigor** like that of other computing areas that require high quality … In security, the **universe** is **by definition hostile**: a malicious agent actively seeks to cause a failure.”

Charles P. Pfleeger, ***The Fundamentals of Information Security***, *IEEE Software*, *January 1997.*[218]

[218] DOI: 10.1109/52.566419

“**Objects**, **patterns**, and **architectures** have much in common. Each holds the **promise** of solving chronic software development problems: high **development costs**, even higher **maintenance costs**, low levels of **reuse**, unbelievable—and unrealized—**schedules**, and so on. “

Stephen J. Mellor, Ralph Johnson, ***Why Explore Object Methods, Patterns, and Architectures?****, IEEE Software, January 1997.*[219]

[219] DOI: 10.1109/MS.1997.566424

“**Patterns** have given us a **vocabulary** to talk about **structures larger than modules, procedures, or objects**—structures that outstrip the vocabularies of the proven object design methods that have served us for the past decade.”

---

*James O. Coplien, **Idioms and Patterns as Architectural Literature**, IEEE Software, January 1997.*[220]

[220] DOI: 10.1109/52.566426

“In the software development context, **a pattern** is an **important and recurring** system construct and a **pattern language** is a system of patterns **organized in a structure** that guides the patterns’ application.”

Ward Cunningham, Norman L. Kerth, ***Using Patterns to Improve Our Architectural Vision****, IEEE Software, January 1997.*[221]

[221] DOI: 10.1109/52.566428

IEEE Software
MARCH/APRIL 1997

Tim Lister
Interview, p. 134

Measurement

Also:

Processing Beethoven

Alert on Contract Law

Cleanroom Fails Test

IEEE COMPUTER SOCIETY

“The goals of consistent design can be clarified by a look at early **user interface design for automobiles**. Early automobile designers offered their own distinct designs for a profusion of controls. Some designs, such as a brake that was too far from the gas pedal, were dangerous. There was also a consistency issue. If your brake was to the left of the gas pedal and your neighbor’s car had the reverse design, it might be risky to trade cars. Achieving good design and appropriate consistency in automobiles took half a century. Let’s hope we can make the transition faster for Web-search user interfaces.”

---

*Ben Shneiderman, **A Framework for Search Interfaces**, IEEE Software, March 1997.*[222]

[222] DOI: 10.1109/52.582969

"As in most other sciences, we are moving along **a measurement continuum**; just as temperature measurement began as an **index finger in the water** (and a scale of not hot enough, hot enough, and too hot) and **grew to sophisticated scales**, tools, and techniques, so too is **software measurement maturing** and leading to a more sophisticated understanding of better ways to produce better products."

---

*Shari Lawrence Pfleeger, **Assessing Measurement**, IEEE Software, March 1997.*[223]

[223] DOI: 10.1109/52.582970

"The most **successful measurement programs** are ones in which **researchers**, **practitioner**, and **customer** work **hand in hand** to meet goals and solve problems. But **such collaboration is rare**."

---

*Ross Jeffery, Bill Curtis, Barbara Kitchenham, Shari Lawrence Pfleeger, **Status Report on Software Measurement**, IEEE Software, March 1997.*[224]

[224] DOI: 10.1109/52.582973

“The more **integral** software **measurement** is to the company’s underlying **business strategy**, the more likely it is to succeed.”

---

*Raymond J. Offen, Ross Jeffery, **Establishing Software Measurement Programs**, IEEE Software, March 1997.*[225]

[225] DOI: 10.1109/52.582974

IEEE Software
MAY/JUNE 1997

Capers Jones
Interview, p. 114

Managing Risk

Also:
Wise Words
Industry Snapshots
Y2K Looms

IEEE COMPUTER SOCIETY

“Our culture has evolved such that **owning up to risks** is often **confused with defeatism**. Thus, a manager faced with a nearly impossible schedule may deliberately **ignore risks** to project a confident, **‘can-do’ attitude**.”

Tom DeMarco, Barry W. Boehm, ***Software Risk Management****, IEEE Software, May 1997.*[226]

[226] DOI: 10.1109/MS.1997.589225

“**Risk management in maintenance** differs in major ways from risk management in development. Risk opportunities are **more frequent**, risks come from **more diverse** sources, and projects have **less freedom to act** on them.”

Kevin Macg. Adams, Robert N. Charette, Mary B. White, ***Managing Risk in Software Maintenance****, IEEE Software, May 1997.*[227]

[227] DOI: 10.1109/52.589232

“Incorporating **hard data** into **risk estimates** can help make them **more accurate**.”

Kari Känsälä, ***Integrating Risk Assessment with Cost Estimation***, *IEEE Software*, *May 1997.*[228]

[228] DOI: 10.1109/52.589236

“There are approximately **670 working days** between now and **January 1, 2000**. Is that enough time to fix the date fields in your programs?”

---

*John Charles, Capers Jones, **Interview with Capers Jones: Slow Response to Year 2000 Problem**, IEEE Software, May 1997.*[229]

[229] DOI: 10.1109/MS.1997.10009

IEEE Software
JULY/AUGUST 1997

Ken Whitaker
Interview, p. 102

Creating Effective Interfaces

Also:

- Why Numbers Lie
- Domain Name Chaos
- Programmers as Writers

IEEE COMPUTER SOCIETY

“There’s something desperately wrong with the **quality of the quantities** we’ve been using.”

---

*Robert L. Glass, **Telling Good Numbers from Bad Ones**, IEEE Software, July 1997.*[230]

[230] DOI: 10.1109/MS.1997.595876

“**Users** should be **involved in the design** process to improve the mapping of their goals to the design. Now that prototypes are being introduced … users should be **tested** using them **in realistic ways** and situations. The results of such testing should then be **fed back** into the process, along with other lessons from the prototyping activity, to **modify design**. “

Andrew Sears, Arnold M. Lund, ***Creating Effective User Interfaces****, IEEE Software, July 1997.*[231]

[231] DOI: 10.1109/MS.1997.595887

"The true choice is not between **discount** and **deluxe usability engineering**. If that were the choice, I would agree that the **deluxe** approach would bring better results. The true choice, however, is between **doing something** and **doing nothing**. Perfection is not an option. My choice is to do something!"

---

*Jakob Nielsen, **Something Is Better than Nothing**, IEEE Software, July 1997.*[232]

[232] DOI: 10.1109/MS.1997.595892

“**A checklist** for **choosing a tool** that fits your needs: **usability**, **functionality**, **flexibility**, **portability**, **support**, and **cost** are all part of the picture.”

Laura A. Valaer, Robert G. Babb II, ***Choosing a User Interface Development Tool****, IEEE Software, July 1997.*[233]

[233] DOI: 10.1109/52.595896

“**Discussions with users** must be **carefully planned** … Identifying ourselves and gaining their confidence was of utmost importance … **Learning** the **work culture** and adapting to it go a long way toward **winning support**. “

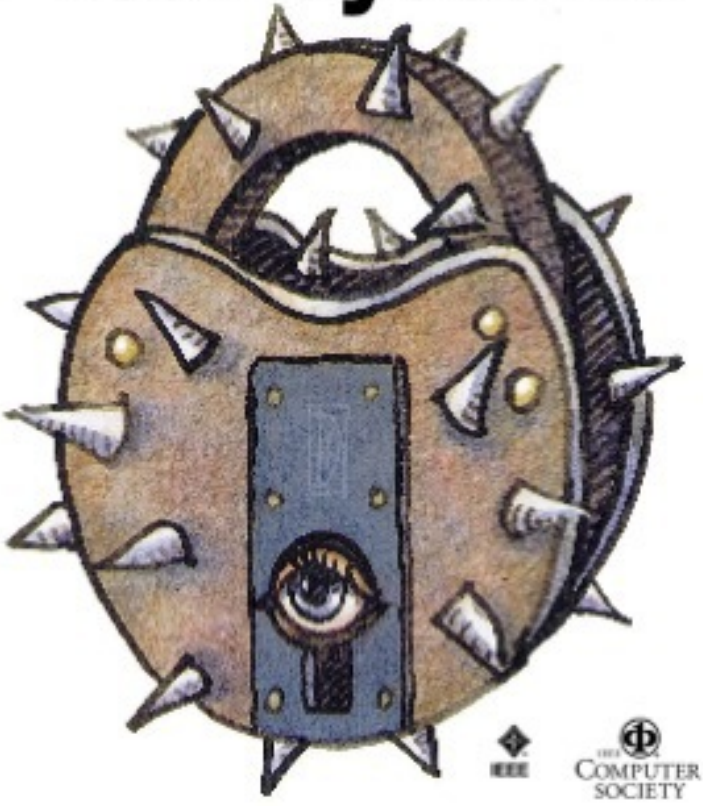
Ben Shneiderman, Catherine Plaisant, Anne Rose, Ajit J. Vanniamparampil, ***Low-Effort, High-Payoff User Interface Reengineering****, IEEE Software, July 1997.*[234]

[234] DOI: 10.1109/52.595958

IEEE Software
SEPTEMBER/OCTOBER 1997

Dorothy Denning
Interview, p. 108

How Safe Is Your System?



Also:

Does CMM Pay? Motorola's Story

Legalities Of Linking

The Mousetrap Paradigm

IEEE COMPUTER SOCIETY

"Too often we **fail to see a new problem** clearly because we color our perception of it with recollections of similar problems we've solved in the past. We also **inject fragments of those past solutions** into our **thinking**, further obscuring the current problem."

Carlo Pescio, ***When Past Solutions Cause Future Problems****, IEEE Software, September 1997.*[235]

[235] DOI: 10.1109/52.605925

“All this **personal connectivity** has a **price**: at times, people deal electronically with people they **do not know**, **cannot name**, and certainly have **no basis to trust**.”

Charles P. Pfleeger, Deborah M. Cooper, ***Security and Privacy: Promising Advances****, IEEE Software, September 1997.*[236]

[236] DOI: 10.1109/52.605928

“Computer use leaves **trails of activity** that can reveal **signatures** of **misuse** as well as of **legitimate activity**. Depending on the audit method used, one can record a user’s **keystrokes**, the system **resources used**, or the **system calls** made by some collection of processes.”

Steven A. Hofmeyr, Andrew P. Kosoresow, ***Intrusion Detection via System Call Traces****, IEEE Software, September 1997.*[237]

[237] DOI: 10.1109/52.605929

IEEE Software
NOVEMBER/DECEMBER 1997

Bill Cheswick
Interview, p. 110

Training for Tomorrow

Also:
The CS-IS Divide
Internet2
Software on a Diet
Craft vs. Science

http://computer.org

IEEE COMPUTER SOCIETY

“Before software development can become a true engineering discipline, its **practitioners** must be **well schooled** in computer **science**, discrete **mathematics**, and a subject too rarely addressed in most university courses today: engineering **economy**.”

---

*Steve Tockey, **A Missing Link in Software Engineering**, IEEE Software, November 1997.*[238]

[238] DOI: 10.1109/52.636594

“Most software engineering **graduates** begin their careers lacking an **appreciation of real-world** conditions.”

---

*Ray Dawson, Ron Newsham, **Introducing Software Engineers to the Real World**, IEEE Software, November 1997.*[239]

[239] DOI: 10.1109/52.636640

“When it comes to software engineering education, there is **a gap** between what **industry needs** and what **universities offer**.”

Nancy R. Mead, Neal Coulter, Kathy Beckman, Soheil Khajenoori, ***Collaborations: Closing the Industry-Academia Gap****, IEEE Software, November 1997.*[240]

[240] DOI: 10.1109/52.636668

“**Complexity** is not completely essential; it is feasible to **reduce** and to **manage complexity**. “

Christof Ebert, ***The Road to Maturity: Navigating Between Craft and Science***, *IEEE Software*, *November 1997.*[241]

[241] DOI: 10.1109/52.636674

1998

IEEE
January/February 1998

SOFTWARE

Building the Community of Leading Software Practitioners

Java co-creator
Patrick Naughton
Interview: p/14

A Tale of Two Futures

Also:

Beyond Dilbert • Handling Your Data • Fear of Trying

IEEE COMPUTER SOCIETY
http://computer.org

“Our canonical example was **the thermostat on the wall**. … There’s **nothing about** the **Web**, or desktop, or Windows vs. Sun that had anything to do with the **initial design of Java**.”

---

*Patrick Naughton, **Basic to Java: Assembling a Career**, IEEE Software, January 1998.*[242]

[242] DOI: 10.1109/MS.1998.646824

“We ended up **merging NeWS and X**, which **was just torture**. I was constantly agitating for mercy; I wanted them to just shoot us. “

Patrick Naughton, ***Basic to Java: Assembling a Career***, *IEEE Software, January 1998.*[243]

[243] DOI: 10.1109/MS.1998.646824

“The likely future of the software industry is this: it will be either **the best of times** or the **worst of times-or both**. … **Year 2000 crisis** may plunge us all into several years of decidedly **unpleasant times**.”

---

*Ed Yourdon, **A Tale of Two Futures**, IEEE Software, January 1998.*[244]

[244] DOI: 10.1109/52.646826

"I like to tell people that my **head** is **in** software's **academic world**, but that my **heart** is **in** its **practice**."

---

*Robert L. Glass, **In Praise of Practice**, IEEE Software, January 1998.*[245]

[245] DOI: 10.1109/MS.1998.10007

“For software developers at **established organizations**, caught in the cross-fire of the **language wars**, the **operating system wars**, the **platform wars**, the **middleware wars**, and the **browser wars**, laboring to push yet another release out the door under compressed schedules, weighed down by the **millstone of legacy code**, and torn asunder by the ravages of ill-conceived and ever-changing requirements, then the **future looks grim** - to put it mildly. “

---

*Grady Booch, **Leaving Kansas**, IEEE Software, January 1998.*[246]

[246] DOI: 10.1109/MS.1998.646876

“**Modesty prevents** me from using the **adjectives** most commonly applied by developers toiling under such circumstances.”

Grady Booch, ***Leaving Kansas***, *IEEE Software*, *January 1998.*[247]

[247] DOI: 10.1109/MS.1998.646876

“We might be expected to become **serious software engineers**. We won’t, of course. “

---

*Michael Jackson, **Will There Ever Be Software Engineering?**, IEEE Software, January 1998.*[248]

[248] DOI: 10.1109/MS.1998.646877

“The **software development** and **maintenance processes**, which I prefer to unify and call **software evolution**, **are key** to managing computerization. In my view it is key to our survival in this computer age.”

---

*M.m. Lehman, **Software’s Future: Managing Evolution**, IEEE Software, January 1998.*[249]

[249] DOI: 10.1109/MS.1998.646878

“History does suggest that our **early expectations** about the **problem-solving potential** of **unbridled technology** should be relatively **low**.”

Steve Andriole, ***Software: The Good, the Bad, and the Real***, *IEEE Software, January 1998.*[250]

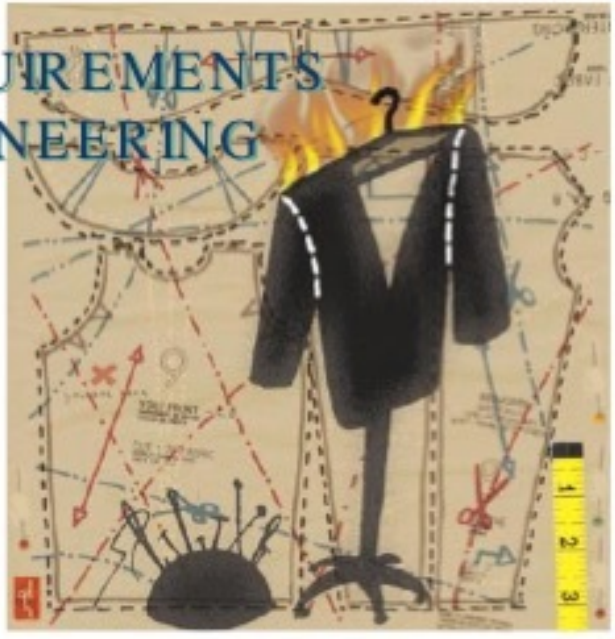
[250] DOI: 10.1109/MS.1998.646879

March/April 1998

IEEE SOFTWARE

Building the Community of Leading Software Practitioners

REQUIREMENTS ENGINEERING



Also:

Who Is the Real Patrick Naughton?

Inventing the Future with Alan Kay

Black Death Parallels

Online Fantasies

Design Studio Diaries

IEEE COMPUTER SOCIETY
http://computer.org

“You could … say that the **main business** of **everyone on earth** is to **help everyone else** — including ourselves — get **enlightened** because the technology is getting more and more dangerous. “

Alan Kay, ***Alan Kay: Inventing the Future***, *IEEE Software*, *March 1998.*[251]

[251] DOI: 10.1109/MS.1998.10013

“Most things printed in the **first 100 years of the printing** press and most of the stuff printed now is **crap**. But because of luck and the bell curve of life, you can still get decently educated by reading just a few thousand books.”

Alan Kay, ***Alan Kay: Inventing the Future***, *IEEE Software, March 1998.*[252]

[252] DOI: 10.1109/MS.1998.10013

"A lot of thinking is like **case law**: 'Has somebody **rich and famous** done this? If so, maybe I should **pay attention** to it.'"

---

*Alan Kay, **Alan Kay: Inventing the Future**, IEEE Software, March 1998.*[253]

[253] DOI: 10.1109/MS.1998.10013

“The first microprocessors were **incredibly slow**. You had to **understand Moore’s law**, what was going to happen. And you had to **have imagination**.”

Alan Kay, ***Alan Kay: Inventing the Future***, *IEEE Software*, *March 1998.*[254]

[254] DOI: 10.1109/MS.1998.10013

“Slowly, we are bridging the gap between **requirements engineering** research and practice. The gap is still large, but we have a few more **practice-validated methods** and tools in our pockets, and the bridge building continues.”

Daniel M. Berry, Brian Lawrence, ***Guest Editors’ Introduction: Requirements Engineering****, IEEE Software, March 1998.*[255]

[255] DOI: 10.1109/MS.1998.663780

"Something is **seriously wrong** with **reuse**. If there is a motherpie-and-applehood topic in software engineering, **reuse** is it. Everyone believes in it; everyone thinks we should be doing more of it. So do I. … 'Why hasn't that potential already been achieved?' … I think reuse hasn't succeeded to the extent we would like because **there aren't that many software components** that **can be reused**."

Robert L. Glass, ***Reuse: What's Wrong with This Picture?***, *IEEE Software, March 1998.*[256]

[256] DOI: 10.1109/52.663785

“Some software designers have recently turned to **building architecture for inspiration** in their efforts to improve professional practice. An attempt to apply **the studio method** of architectural training to **software design education** … reveals much about education and practice in both professions. Studio courses **provoke creative reflection** on how to improve current training practices and could provide a new way to develop software design expertise.”

Sarah Kuhn, ***The Software Design Studio: An Exploration***, *IEEE Software*, *March 1998.*[257]

[257] DOI: 10.1109/52.663788


May/June 1998
IEEE
SOFTWARE
Building the Community of Leading Software Practitioners
Webward Ho!
Also
Selling OO:
Who's Buying, Who's Not
Y2K B42L8
How to Build
More Usable APIs
IEEE COMPUTER SOCIETY
http://computer.org

“**Web technology** can offer a relatively painless way to **extend the life of legacy systems**, which are, by definition, both **fragile and valuable**. The Web can give aging applications a **modern graphical interface**, deliver them to employees’ desktops regardless of platform, and grant access to databases distributed across the enterprise.”

Ellis Horowitz, ***Guest Editor’s Introduction: Migrating Software to the World Wide Web****, IEEE Software, May 1998.*[258]

[258] DOI: 10.1109/MS.1998.676714

"**Geographic information systems** store a wealth of information for diverse applications, and users must often access GISs from various vendors and residing on varying platforms. … a **spatial query mechanism** for GISs that is distributed and open, giving users access to many different GISs across the Internet."

Serena Coetzee, Judith Bishop, ***A New Way to Query GISs on the Web****, IEEE Software, May 1998.*[259]

[259] DOI: 10.1109/52.676719

“Object-orientation (OO) does not match the way we normally think. … however, that this is **true of programming in any language** using any paradigm.”

---

*Richard Wiener, **Watch Your Language!**, IEEE Software, May 1998.*[260]

[260] DOI: 10.1109/52.676738

“When the authors asked users to **test an API early** in the development life cycle, the **users’ questions** about how the API works and in what contexts turned out to be **extremely helpful**. **Iterative API design** and testing, along with **feedback from real users**, contribute to cleaner design and a more helpful reference manual.”

Clay I. Spinuzzi, Samuel G. McLellan, Alvin W. Roesler, Joseph T. Tempest, ***Building More Usable APIs****, IEEE Software, May 1998.*[261]

[261] DOI: 10.1109/52.676963

“**McDonald’s** produces **hamburgers** that perfectly **match user requirements** and **satisfy expectations** for affordability and timely arrival. **But quality dining**? No one, not even McDonald’s, claims that.”

Robert L. Glass, ***Defining Quality Intuitively***, *IEEE Software*, *May 1998.*[262]

[262] DOI: 10.1109/52.676973

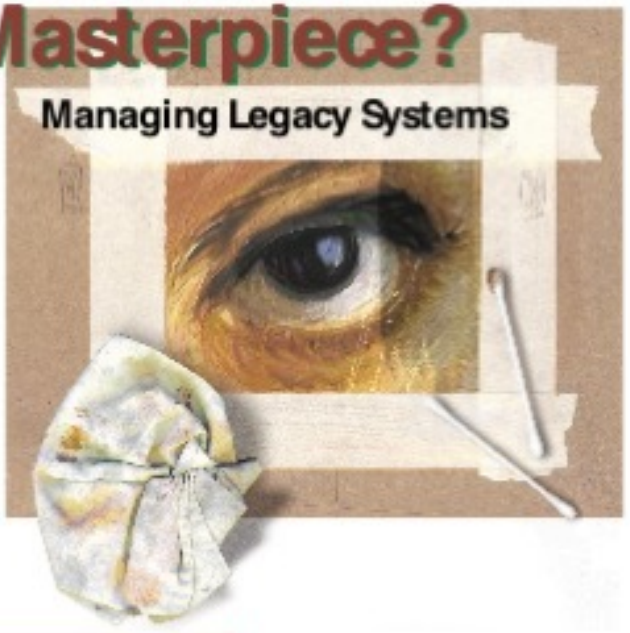
July/August 1998

IEEE SOFTWARE

Building the Community of Leading Software Practitioners

Menace or Masterpiece?

Managing Legacy Systems



Also:

- AI's Farewell Predictions
- Netscape Opens Up
- Managing OO Projects
- Requirements: Value vs. Cost

IEEE COMPUTER SOCIETY

http://computer.org

“**Legacy software systems** represent a **significant investment** but often become difficult to maintain as they age. Not only does **technology evolve** beyond them, but **business needs change** and may require adding or modifying functions. “

Christof Ebert, Norman F. Schneidewind, ***Guest Editors' Introduction: Preserve or Redesign Legacy Systems?****, IEEE Software, July 1998.*[263]

[263] DOI: 10.1109/MS.1998.687937

"**Rebuilding a legacy system** has some parallels to the **restoration of a work of art** … The restoration involved far **more than updating the code**: the development team also had to **understand the existing** architecture, **add new** functionality, and develop a long-term hardware migration plan."

Jim White, Spencer Rugaber, ***Restoring a Legacy: Lessons Learned****, IEEE Software, July 1998.*[264]

[264] DOI: 10.1109/52.687941

“Drawing on extensive data from the **NASA Space Shuttle**’s guidance software, Schneidewind … provides **several formulas** for **tracking maintenance stability** defined as increasing functionality with decreasing failures …”

Norman F. Schneidewind, ***How To Evaluate Legacy System Maintenance****, IEEE Software, July 1998.*[265]

[265] DOI: 10.1109/52.687942

“If this decade’s mantra is ‘**show me the money**,’ those of us in software will **find it in maintenance**. Life-cycle data shows that maintenance is where we spend the biggest chunk of practitioner **time and money**.”

Robert L. Glass, ***Maintenance: Less Is Not More***, *IEEE Software, July 1998.*[266]

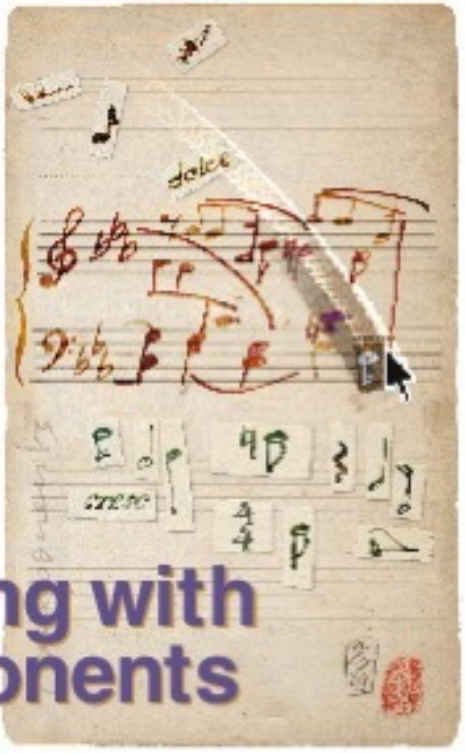
[266] DOI: 10.1109/52.687948

September/October 1998

IEEE

SOFTWARE

Building the Community of Leading Software Practitioners



Working with Components

IEEE COMPUTER SOCIETY

http://computer.org

“The **rate of change** in the software world is such that shortly after the ink dries on this issue, we can only hope to have captured a snapshot of a rapidly maturing domain.”

Grady Booch, Wojtek Kozaczynski, ***Guest Editors’ Introduction: Component-Based Software Engineering****, IEEE Software, September 1998.*[267]

[267] DOI: 10.1109/MS.1998.714621

“A **component-based system** requires an **infrastructure for communication and collaboration**. … examines the **Microsoft MTS** and **OMG Corba** infrastructure technologies. … comparing how Microsoft’s MTS and **Sun’s Enterprise JavaBeans** handle transactions and component state.”

Alan W. Brown, Kurt C. Wallnau, ***The Current State of CBSE****, IEEE Software, September 1998.*[268]

[268] DOI: 10.1109/52.714622

“The need to closely examine a problematic aspect of component reuse: the necessity and potential **expense** of **validating components** in their new environments.”

Elaine J. Weyuker, ***Testing Component-Based Software: A Cautionary Tale***, *IEEE Software*, *September 1998.*[269]

[269] DOI: 10.1109/52.714817

“Ellen Ullman, describing her first **Linux** installation, wrote ‘I exaggerate only a little if I say that **Linux** is a **reassertion of our dignity** as humans working with mere machine; a **return**, quite literally, **to the source**.’”

James Sanders, ***Linux, Open Source, and Software’s Future****, IEEE Software, September 1998.*[270]

[270] DOI: 10.1109/52.714831

November/December 1998

IEEE SOFTWARE

Building the Community of Leading Software Practitioners

Setting the Standard

Also

IEEE COMPUTER SOCIETY
http://computer.org

"This issue is dedicated to **Alan Davis**, our dear **colleague**, **thinker**, **doer**, **leader**, sometimes **dreamer**, and a congenial friend always."

Maarten Boasson, Carl K. Chang, Tomoo Matsubara, ***Guest Editors' Introduction: Setting the Standard****, IEEE Software, November 1998.*[271]

[271] DOI: 10.1109/MS.1998.10037

“The ultimate **survival** of the **Java Virtual Machine** is much less certain. Java’s inventors at Sun Microsystems have presented their creation as a single inseparable concept that encompasses a **language**, a rich class **library**, and a software distribution standard based on **a virtual machine**. These three parts are **not nearly as inseparable** as Sun claims. “

---

*Michael Franz, **The Java Virtual Machine: A Passing Fad?**, IEEE Software, November 1998.*[272]

[272] DOI: 10.1109/52.730834

"At **Hitachi Software**, our software has attained such **high quality** that **only 0.02 percent of all bugs** in a software program emerge at the **user's site**. … We **do not use sophisticated** tools or state-of-the-art methodology — **we simply test** programs and fix the bugs detected."

---

*Tsuneo Yamaura, **How to Design Practical Test Cases**, IEEE Software, November 1998.*[273]

[273] DOI: 10.1109/52.730835

“There is probably no hope of changing the view that **Wall Street** takes of treating investment in **people as an expense**. But companies that play this game will **suffer in the long run**. The converse is also true: Companies that manage their investment sensibly will prosper in the long run. Companies of knowledge workers have to realize that it is their **investment in human capital that matters most**. The good ones already do.”

---

*Timothy Lister, Tom DeMarco, **Human Capital**, IEEE Software, November 1998.*[274]

[274] DOI: 10.1109/52.730859

“Adler differentiates between **four levels of reading**.

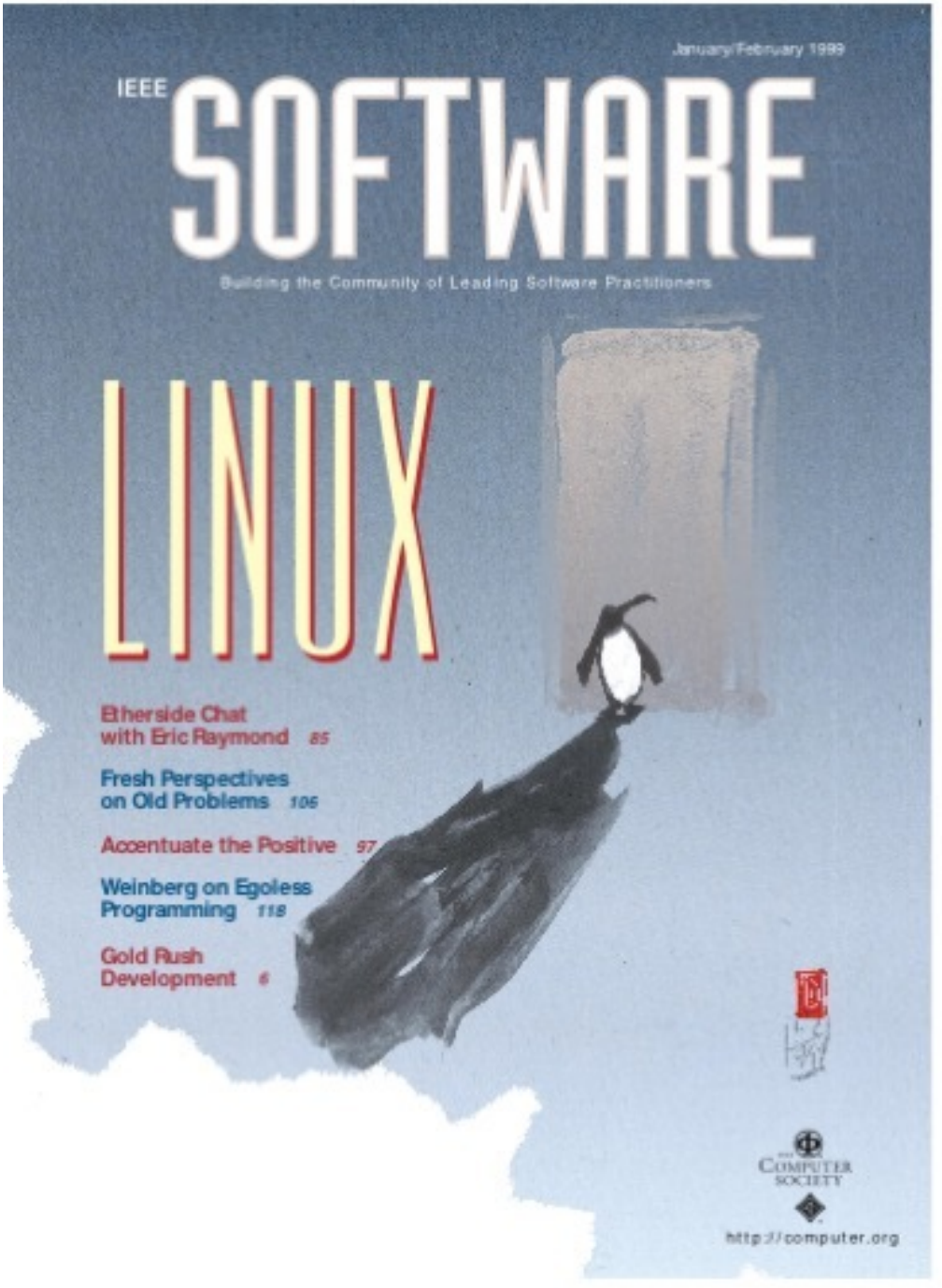
Elementary reading … recognize individual words on a page.

Inspectional reading is … trying to get the most out of a book or article within a **given amount of time**. **Analytical reading** is … by trying to get the most out of a book or article with **an unlimited amount of time**. **Syntopical reading** … involves **reading sets of books or articles** on a common topic in a way that enables you to extract information that might or might not be present in any of the individual materials studied.”

Steve McConnell, ***How To Read a Technical Article***, *IEEE Software, November 1998.*[275]

[275] DOI: 10.1109/MS.1998.10035

1999


January/February 1999
IEEE
SOFTWARE
Building the Community of Leading Software Practitioners
LINUX
Etherside Chat with Eric Raymond 85
Fresh Perspectives on Old Problems 105
Accentuate the Positive 97
Weinberg on Egoless Programming 118
Gold Rush Development 6
IEEE COMPUTER SOCIETY
http://computer.org

“**Linux** is a free, open-source operating system that **looks like Unix**, except that it runs on PCs as well as other platforms. Linux was created by **Linus Torvalds** in **1991**. Today, Linux is cooperatively improved by Torvalds and **thousands of volunteers** from around the world using open-source development methods.”

Terry Bollinger, Peter Beckman, ***Guest Editors’ Introduction: Linux on the Move****, IEEE Software, January 1999.*[276]

[276] DOI: 10.1109/MS.1999.744564

“Far from simply producing freeware clones of existing technologies, **Linux and the open-source** world now turn out **some of the best software** at any price.”

---

*Evan Leibovitch, **The Business Case for Linux**, IEEE Software, January 1999.*[277]

[277] DOI: 10.1109/52.744567

“**Linux** has been a boon to **computing** in the **developing world**. In **Pakistan**, we have used Linux productively in both academia and industry.”

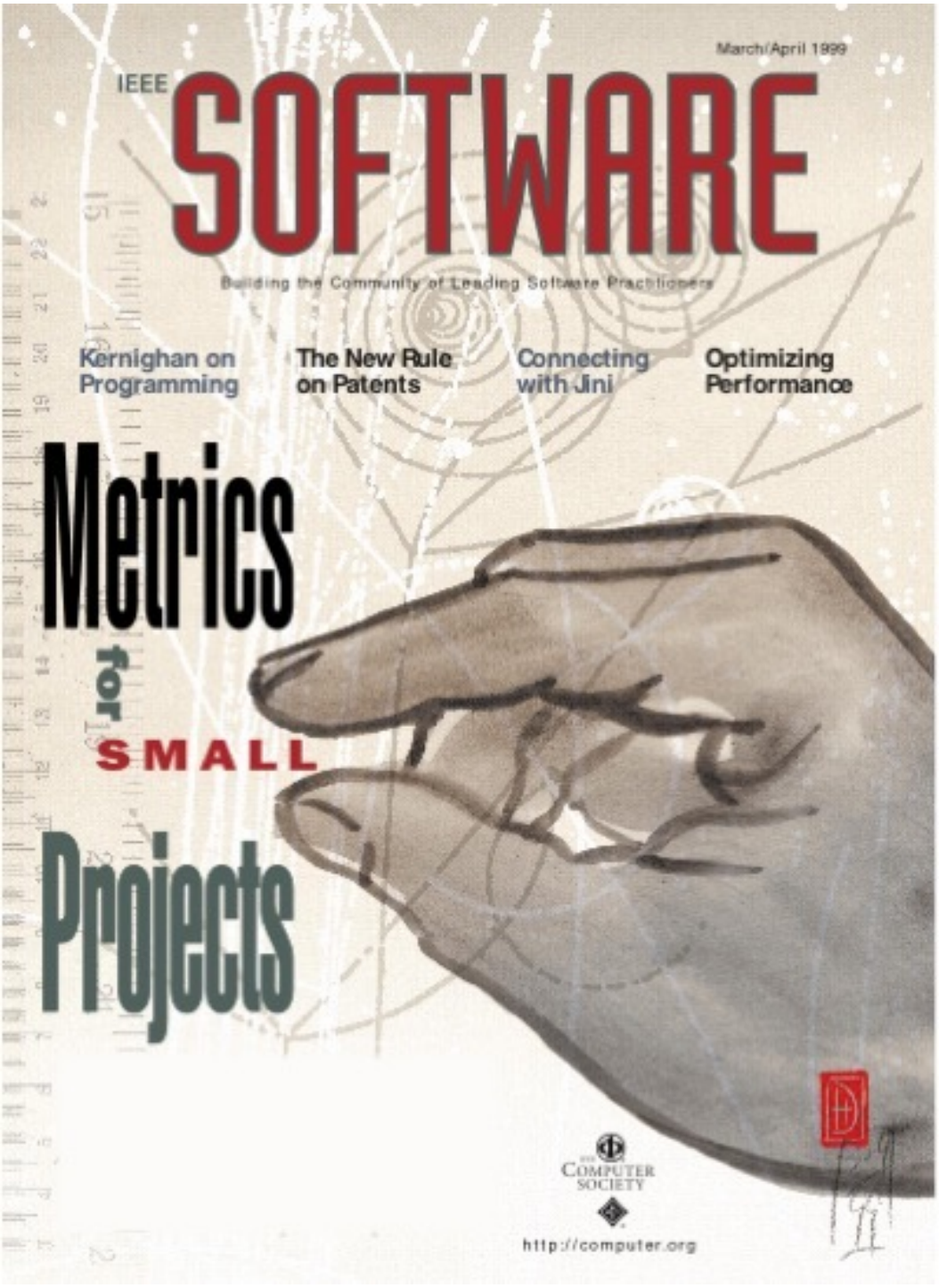
Rafeeq ur Rehman, Shahid H. Bokhari, ***Linux and the Developing World****, IEEE Software, January 1999.*[278]

[278] DOI: 10.1109/52.744570

“How much is the **Year 2000 problem** going to cost you? How long is it going to take you to get ready? Can you make it in time?”

Ware Myers, Lawrence H. Putnam, ***Year 2000 Work Comes Down to the Wire****, IEEE Software, January 1999.*[279]

[279] DOI: 10.1109/52.744575


March/April 1999
IEEE
SOFTWARE
Building the Community of Leading Software Practitioners
Kernighan on Programming
The New Rule on Patents
Connecting with Jini
Optimizing Performance
Metrics for SMALL Projects
IEEE COMPUTER SOCIETY
http://computer.org

"The key to **successful measurement programs** is to make the **metrics meaningful** and **tailor them** to the organization—however small it might be."

Karlheinz Kautz, ***Making Sense of Measurement for Small Organizations***, *IEEE Software*, *March 1999.*[280]

[280] DOI: 10.1109/52.754047

“An organized, comprehensive **metrics program** can **bring order to the chaos** of small-project management and form the foundation for a concerted process improvement effort.”

Ross Grable, Dale Divis, Casey Pogue, Jacquelyn Jernigan, ***Metrics for Small Projects: Experiences at the SED****, IEEE Software, March 1999.*[281]

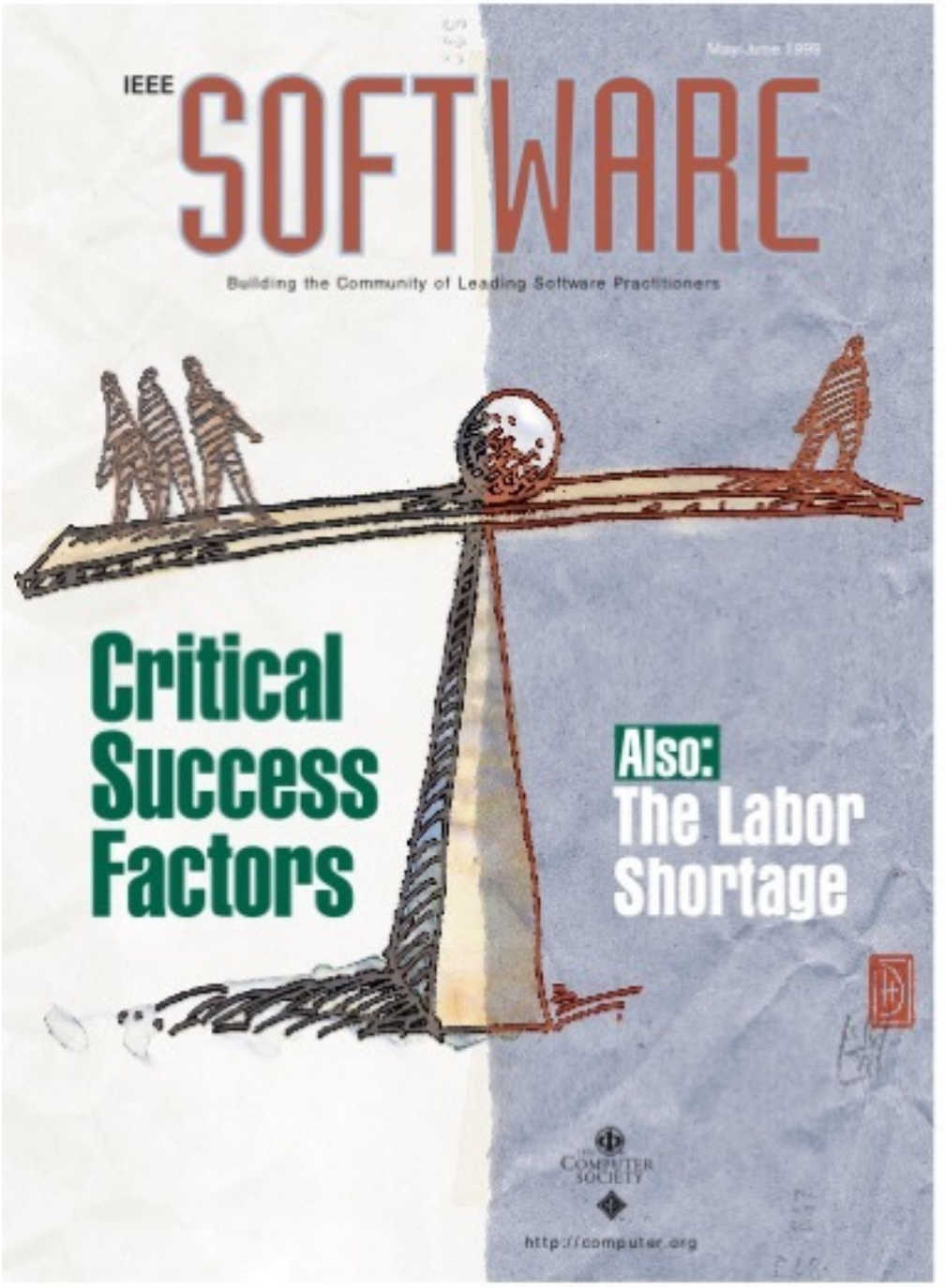
[281] DOI: 10.1109/52.754048

“**Design** is one of the most elusive yet fascinating topics in the software field. It is elusive because, no matter how thoroughly **academics** try to shape it into a teachable, testable, fact-based topic, **it just doesn’t fit**. It is fascinating because design holds the **key to the success** of most software projects.”

---

*Robert L. Glass, **On Design**, IEEE Software, March 1999.*[282]

[282] DOI: 10.1109/MS.1999.754066


IEEE
SOFTWARE
Building the Community of Leading Software Practitioners
Critical Success Factors
Also:
The Labor Shortage
COMPUTER SOCIETY
http://computer.org

“Everyone who ever taught project management seems to have **a favorite disaster story**, whether it’s the new **Denver airport** baggage handling system, the **London Stock Exchange**, or the **French Railways**. … we must also consider a wider viewpoint: **success** requires **much more than good engineering**.”

Andrew J. Bytheway, ***Guest Editor’s Introduction: Successful Software Projects and How to Achieve Them****, IEEE Software, May 1999.*[283]

[283] DOI: 10.1109/MS.1999.765781

"You **cannot go back** and **add quality**. By the time you figure out you have a quality problem, it is probably **too late to fix it**."

---

*John S. Reel, **Critical Success Factors In Software Projects**, IEEE Software, May 1999.*[284]

[284] DOI: 10.1109/52.765782

“The world has become used to having just about **every object, from cars to toasters**, equipped with computers. The hardware chip density has for many years followed Moore’s law of doubling in capacity every 18 months, and experts agree that there is no end in sight to this trend. The increase in hardware power will continue largely independent of advances in software technology. As hardware performance increases, so will the demand to feed this hardware with software.”

Wolfgang Strigel, ***Guest Editor’s Introduction: What’s the Problem: Labor Shortage or Industry Practices?****, IEEE Software, May 1999.*[285]

[285] DOI: 10.1109/MS.1999.765787

“Estimates of **unfilled software engineering**, management, and support jobs in the **US** range from **100,000 to more than 300,000**. Projects addressing **Y2K repairs** and euro currency conversion could push the total shortfall to **600,000 jobs**.”

---

*Capers Jones, **The Euro, Y2K, and the US Software Labor Shortage**, IEEE Software, May 1999.*[286]

[286] DOI: 10.1109/52.765788

“The worldwide demand for software services is increasing at a rate faster than the current output of qualified software engineers. **India** is poised to meet this demand with a **growing pool** of educated, trained **software professionals**.”

Subroto Bagchi, ***India’s Software Industry: The People Dimension***, *IEEE Software, May 1999.*[287]

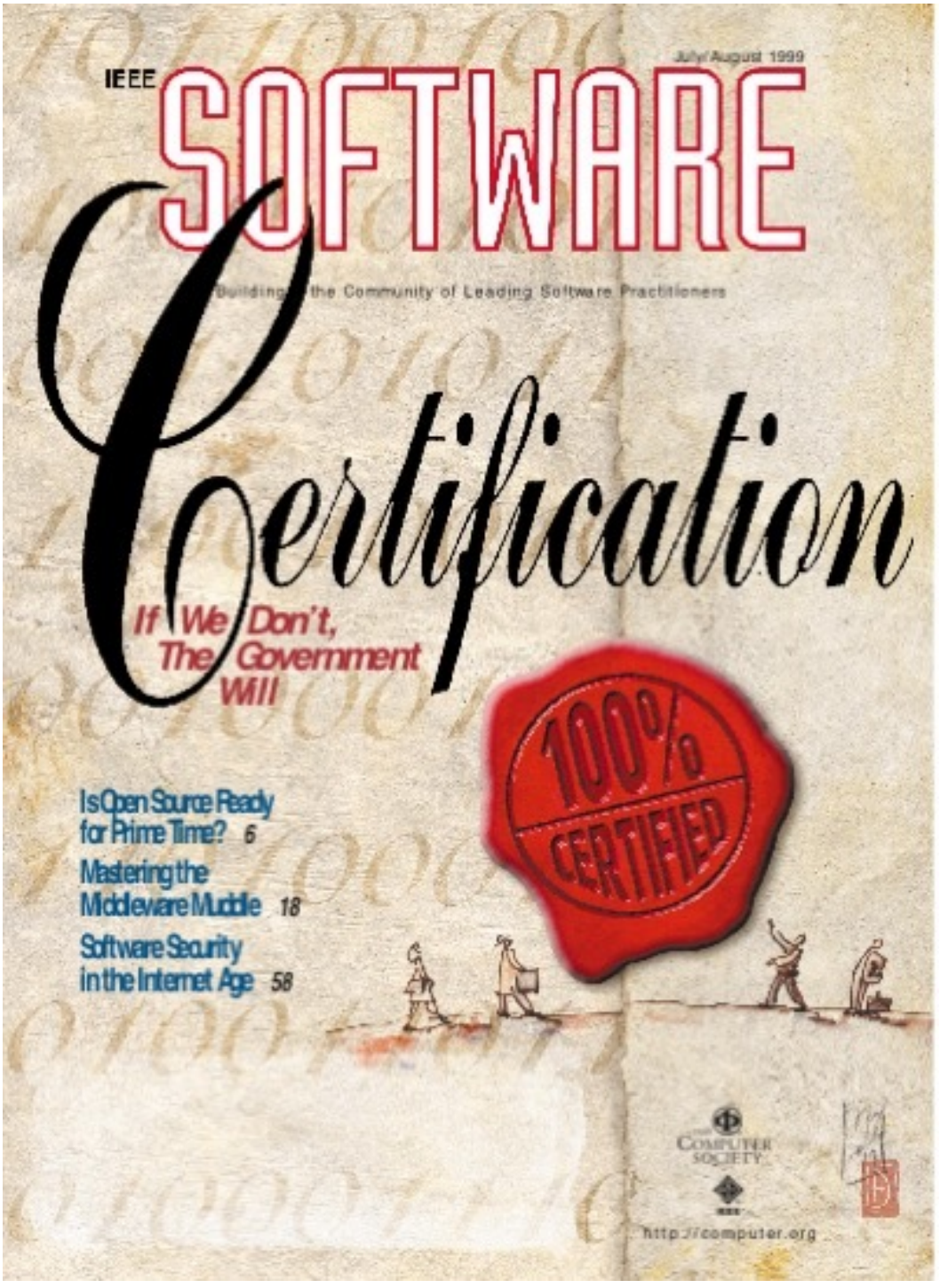
[287] DOI: 10.1109/52.765789

“First and foremost the **Unified Process** is a software development process … the set of activities needed to **transform a user’s requirements** into a **software system**. However, the **Unified Process** is … it is **a generic process framework** that **can be specialized** for a very large class of … systems, … application areas, … organizations, … competence levels, and … project sizes.”

---

*Grady Booch, Ivar Jacobson, James Rumbaugh, **The Unified Process**, IEEE Software, May 1999.*[288]

[288] DOI: 10.1109/MS.1999.10013


IEEE
SOFTWARE
July/August 1999
Building the Community of Leading Software Practitioners
Certification
If We Don't, The Government Will
100% CERTIFIED
Is Open Source Ready for Prime Time? 6
Mastering the Middleware Muddle 18
Software Security in the Internet Age 58
COMPUTER SOCIETY
http://computer.org

“Software development is increasingly an ‘acquire and glue’ process. How do you know when you can trust a COTS component to do what you expect it to in your system? **Software certification** is one viable answer. If our industry doesn’t act soon to police itself, **governments might step in** to fill that void.”

Jeffrey Voas, ***Guest Editor’s Introduction: Certification-Reducing the Hidden Costs of Poor Quality****, IEEE Software, July 1999.*[289]

[289] DOI: 10.1109/MS.1999.776944

“Designers, researchers, and developers are already using **virtual environments** in many ways and in many domains. The technology has matured enough that VEs are beginning to be used to **certify the systems they simulate**.”

Robyn R. Lutz, Carolina Cruz-Neira, ***Using Immersive Virtual Environments for Certification****, IEEE Software, July 1999.*[290]

[290] DOI: 10.1109/52.776945

“The **general-purpose computing** environment that characterizes the PC and Internet was **not designed for privacy** or integrity.”

John Michener, ***System Insecurity in the Internet Age***, *IEEE Software*, *July 1999.*[291]

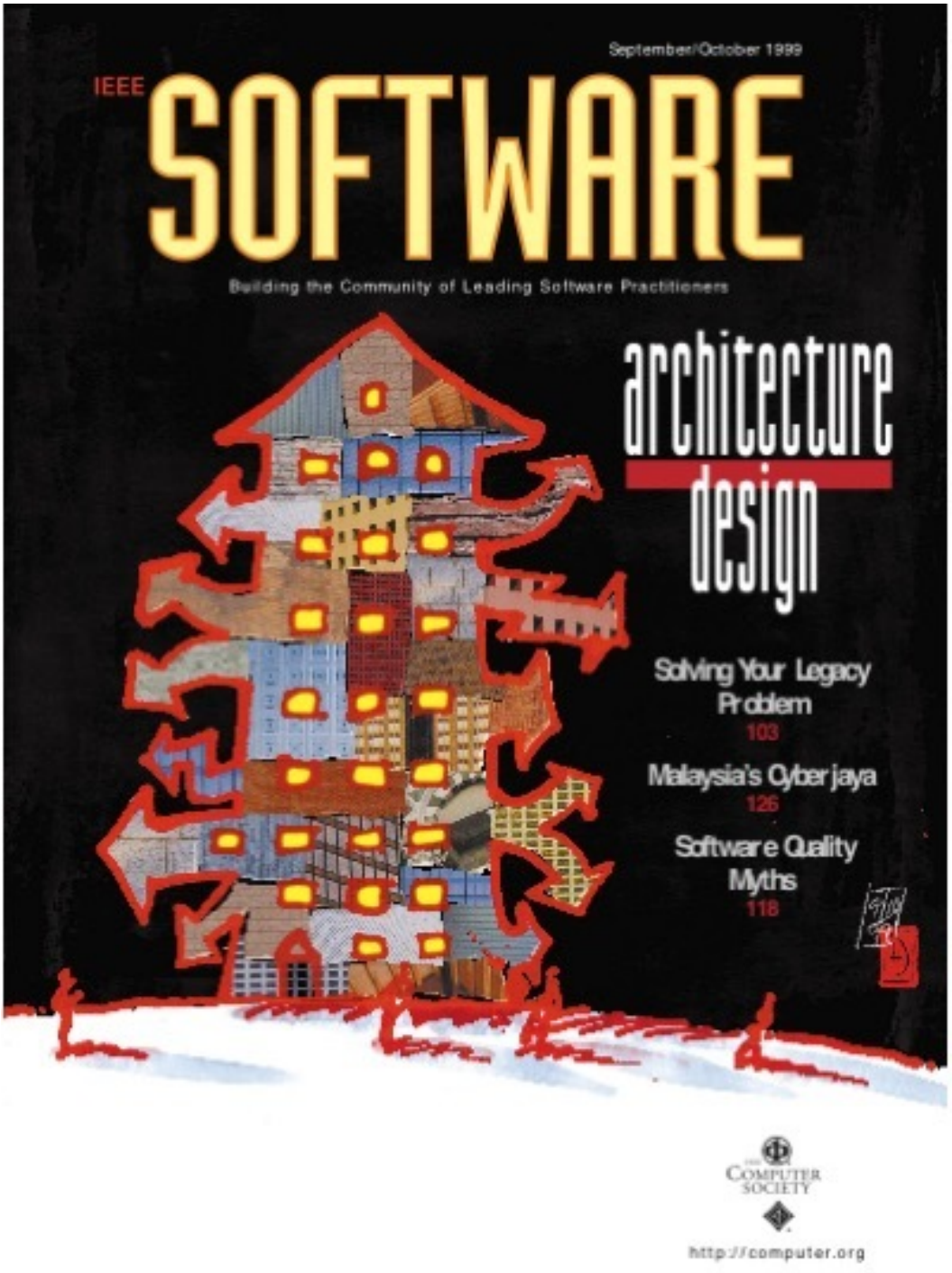
[291] DOI: 10.1109/52.776951

“One of the strangest stories in the software world centers around the programming language **Cobol**. Academics have **reviled it** for decades; its **demise** has been **predicted** since the 1960s; industry gurus have suggested that programmers who know only Cobol are **committing career suicide**. … Yet **the giant** lumbers on.”

---

*Robert L. Glass, **Cobol: A Historic Past, A Vital Future?**, IEEE Software, July 1999.*[292]

[292] DOI: 10.1109/52.776965


September/October 1999
IEEE
SOFTWARE
Building the Community of Leading Software Practitioners
architecture
design
Solving Your Legacy Problem
103
Malaysia's Cyberjaya
126
Software Quality Myths
118
IEEE COMPUTER SOCIETY
http://computer.org

“The very idea that **software reuse** is a legitimate **research discipline** is **a paradox**: in all other engineering disciplines, reuse is an **integral part of good engineering** design — so integral that it is not even noteworthy. “

---

*Ali Mili, Hafedh Mili, Sherif Yacoub, Edward Addy, **Toward an Engineering Discipline of Software Reuse**, IEEE Software, September 1999.*[293]

[293] DOI: 10.1109/52.795098

“The ability to **use Linux in personal-computer platforms**, plus the ready availability of Linux source code, makes the PC-based Linux workstation an ideal platform for **ATM multimedia development**. “

Richard L. Klevans, Steven A. Wright, Ze Zhang, Thomas C. Jepsen, ***Linux Update: An Experimental ATM Network****, IEEE Software, September 1999.*[294]

[294] DOI: 10.1109/52.795099

“**Architecture** is not so much **about the software**, but about the people who write the software. The **core principles** of architecture, such as coupling and cohesion, **aren’t about the cod**e. The **code doesn’t ‘care’** about how cohesive or decoupled it **is**; if anything, tightly coupled software lacks some of the performance snags found in more modular systems. But **people do care** about their **coupling to other team members**. “

James O. Coplien, ***Guest Editor’s Introduction: Reevaluating the Architectural Metaphor-Toward Piecemeal Growth****, IEEE Software, September 1999.*[295]

[295] DOI: 10.1109/MS.1999.795100

“Geographically **distributed** development **teams** face extraordinary communication and coordination problems. … common but unanticipated events can stretch project communication to the breaking point. Project **schedules can fall apart**, particularly during integration.”

---

*James D. Herbsleb, Rebecca E. Grinter, **Architectures, Coordination, and Distance: Conway’s Law and Beyond**, IEEE Software, September 1999.*[296]

[296] DOI: 10.1109/52.795103

“This is a pretty **strange situation** I find myself in. I hope you sympathize with me. I’m addressing a room full of people, a whole football field full of people. I don’t know hardly anything about what all of you do. So—**please be nice to me**.What is the connection between what I am doing in the field of architecture and what you are doing in computer science and trying to do in the new field of software design? “

Christopher Alexander, ***The Origins of Pattern Theory: The Future of the Theory, and the Generation of a Living World****, IEEE Software, September 1999.*[297]

[297] DOI: 10.1109/52.795104

“What I am proposing … is a **view of programming** as the **natural, genetic infrastructure** of a living world which you/we are capable of creating, managing, making available, and which could then have the result that a **living structure** in our **towns**, **houses**, **work** places, cities, becomes an attainable thing. That would be remarkable. It would turn the world around, and make **living structure** the norm once again, throughout society, and make the world worth living in again.”

Christopher Alexander, ***The Origins of Pattern Theory: The Future of the Theory, and the Generation of a Living World****, IEEE Software, September 1999.*[298]

[298] DOI: 10.1109/52.795104


November/December 1999
IEEE
SOFTWARE
Building the Community of Leading Software Practitioners
Professional Software Engineering:
FACT OR FICTION?
Brooks' Law Revealed 6
Linux in Antarctica 90
Shareware Explosion 104
IEEE COMPUTER SOCIETY
http://computer.org

“For many **programmers**, software development consists of **hacking**. As we mature, it is time to follow the example of other professional disciplines, to **put the engineering in software engineering**.”

Steve McConnell, Leonard Tripp, ***Guest Editors’ Introduction: Professional Software Engineering-Fact or Fiction?****, IEEE Software, November 1999.*[299]

[299] DOI: 10.1109/MS.1999.805468

"**'Software Engineering' programs** have become a source of contention in many **universities**. Computer Science departments, many of which have used that phrase to describe individual courses for decades, claim SE as part of their discipline. Yet some engineering faculties claim it as a new specialty among the engineering disciplines. … We need SE programs that **follow the traditional engineering approach** to professional education."

David Lorge Parnas, ***Software Engineering Programs Are Not Computer Science Programs****, IEEE Software, November 1999.*[300]

[300] DOI: 10.1109/52.805469

“The **IEEE Computer Society** and the **Association for Computing Machinery** are working on a joint project to develop a guide to **the Software Engineering Body of Knowledge (SWEBOK)**. Articulating a body of knowledge is an essential step toward developing a profession because it represents a broad consensus regarding the contents of the discipline. “

---

*Alain Abran, James W. Moore, Robert Dupuis, Pierre Bourque, Leonard Tripp, **The Guide to the Software Engineering Body of Knowledge**, IEEE Software, November 1999.*[301]

[301] DOI: 10.1109/52.805471

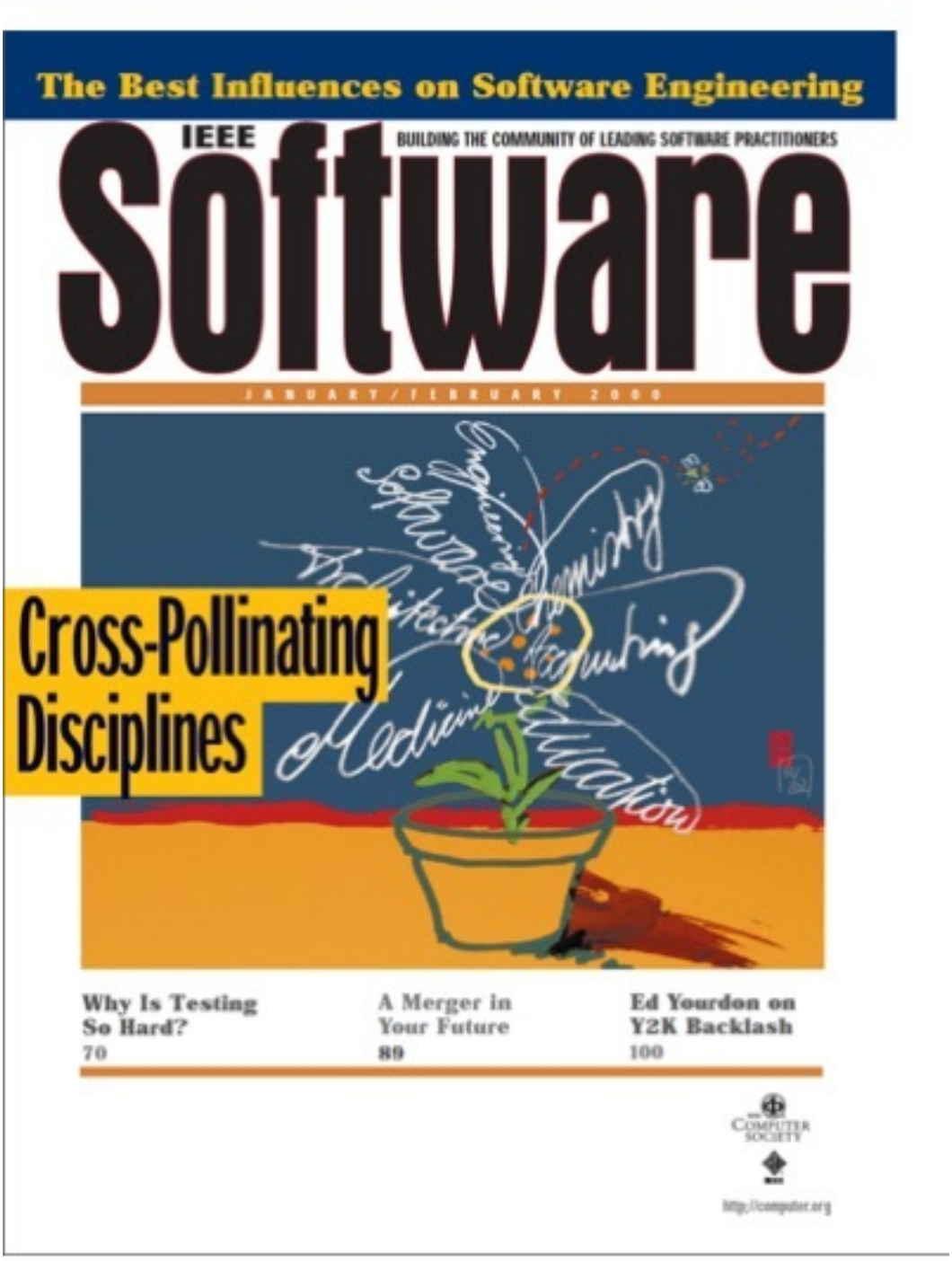
"In June **1998**, the **Texas Board of Professional Engineers** established **software engineering** as a recognized engineering discipline and established **licensing criteria** specifically suited to software engineers."

---

*John R. Speed, **What Do You Mean I Can't Call Myself a Software Engineer?**, IEEE Software, November 1999.*[302]

[302] DOI: 10.1109/52.805472

2000


The Best Influences on Software Engineering
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
JANUARY/FEBRUARY 2000
Cross-Pollinating Disciplines
Why Is Testing So Hard? 70
A Merger in Your Future 89
Ed Yourdon on Y2K Backlash 100
COMPUTER SOCIETY
http://computer.org

“We can learn much from the **business community** about **effective technology transfer**. In particular, understanding the **interests** of **different types of adopters** can suggest to us the different kinds of **evidence** needed to **convince** someone to try an **innovative technology**. At the same time, the **legal community** offers us advice about what kinds of **evidence** are needed to **build convincing cases** that an innovation is an **improvement** over current practice.”

---

*Shari Lawrence Pfleeger, Winifred Menezes, **Marketing Technology to Software Practitioners**, IEEE Software, January 2000.*[303]

[303] DOI: 10.1109/52.819965

“Rather than integrating methods from the social sciences into the systems design process as others have done, this **crosspollination** effort strives to **inform and improve** the development of software engineering itself through a **deeper understanding** of our community’s implicit **values and beliefs**.”

---

*Helen Sharp, Hugh Robinson, Mark Woodman, **Software Engineering: Community and Culture**, IEEE Software, January 2000.*[304]

[304] DOI: 10.1109/52.819967


Y2K and Other Software Noncrises
IEEE
Software
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
MARCH/APRIL 2000
What's Next for Cobol?
COBOL LIVES!
The UK Model for E-Commerce 90
Leadership, Army Style 92
ASPs—The Next Gold Rush 96
COMPUTER SOCIETY
http://computer.org

"That philosophy of **extinction** has been replaced with one of **extension** and inclusion. **Cobol applications** are, by and large, **too critical** and **too valuable** to consider replacing en masse."

Edmund C. Arranga, Wilson Price, ***Guest Editors' Introduction: Fresh from Y2K, What's Next for Cobol?****, IEEE Software, March 2000.*[305]

[305] DOI: 10.1109/MS.2000.841599

“Although recent Internet, Java, and OO trends threaten **Cobol’s dominance**, industry will continue to need the language and its programmers for development as well as maintenance–especially once OO Cobol becomes an official standard.”

Bill C. Hardgrave, E. Reed Doke, ***Cobol in an Object-Oriented World: A Learning Perspective****, IEEE Software, March 2000.*[306]

[306] DOI: 10.1109/52.841601

“Contrary to persistent myths, a **committee** initially **created Cobol in 1959**, not one person. … This material is based on documents from the 1959 committee work…”

---

*Jean E. Sammet, **The Real Creators of Cobol**, IEEE Software, March 2000.*[307]

[307] DOI: 10.1109/52.841602

"**Cobol 2002**, the **new Cobol standard**, is expected to be finalized in approximately 18 months. Cobol 2002 builds on Cobol's first-class data handling capabilities and introduces **object-oriented features**, environmental improvements, and many other modern constructs to the language."

---

*Don Schricker, **Cobol for the Next Millennium**, IEEE Software, March 2000.*[308]

[308] DOI: 10.1109/52.841606


The Sociology of Open Source
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
MAY/JUNE 2000
Requirements Engineering:
Getting the Details Right
Reverse-Engineering Air Traffic Control
63
Six Sigma versus CMM?
11
Update: UCITA
96
COMPUTER SOCIETY
http://computer.org

"The **Y2K issue** generated an enormous flood of activity for organizations worldwide. While we **survived Y2K** with **minor glitches**, the Y2K 'exercise' forced both developers and users of software to appreciate the need for software to function correctly in expected and unexpected situations. None of us is **immune to the problems** of the software industry."

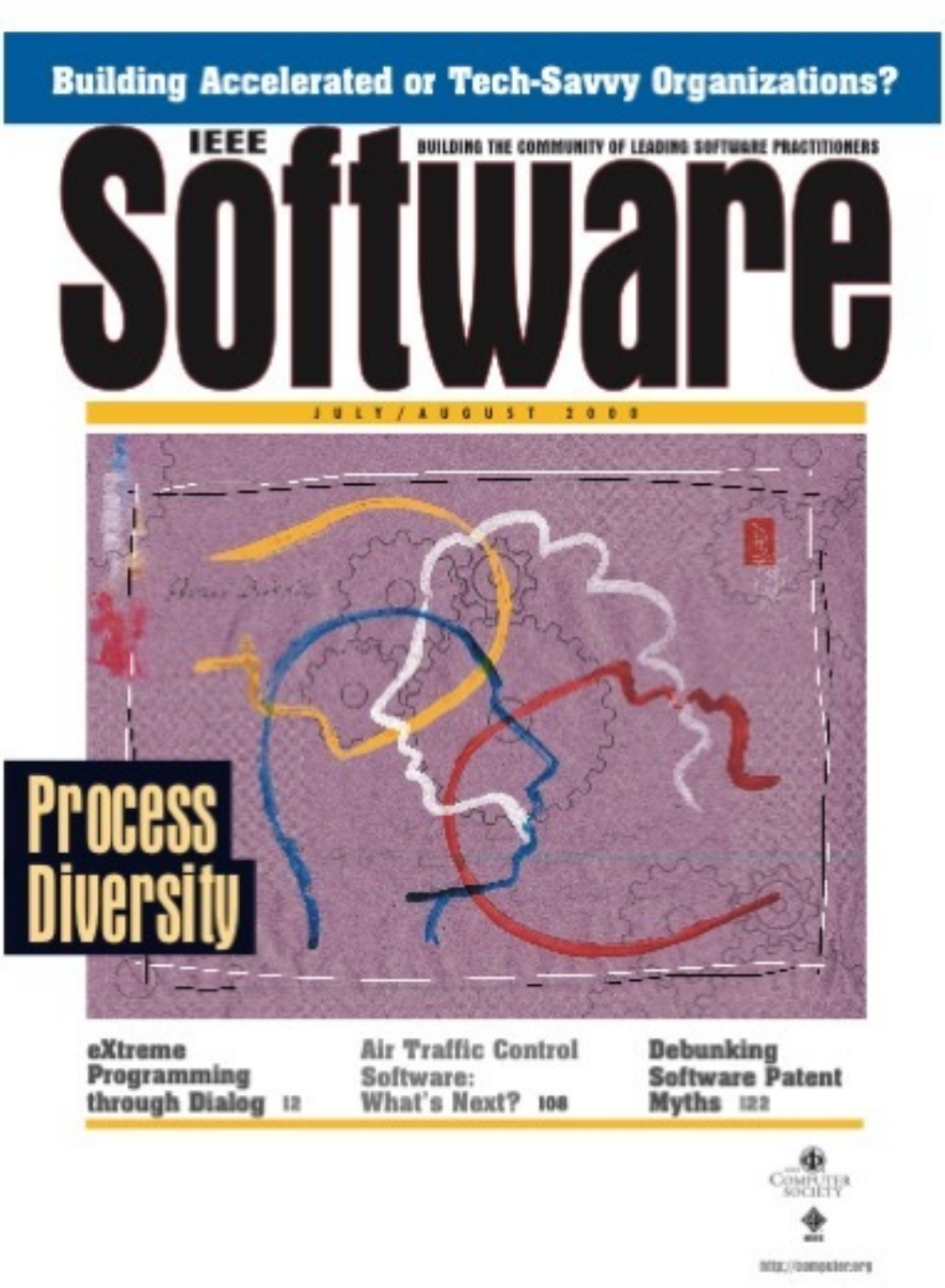
David M. Weiss, Betty H.C. Cheng, ***Guest Editors' Introduction: Requirements Engineering-Integrating Technology****, IEEE Software, May 2000.*[309]

[309] DOI: 10.1109/MS.2000.896245

"There's something I **don't understand** about the **open-source movement**. Oh, I understand open source **intellectually**. … What I **don't understand** is something more **sociological**. I don't understand who those folks are who want to do all that **code reading and reviewing** for **no recompense**. It goes against the grain of everything I know about the software field."

Robert L. Glass, ***The Sociology of Open Source: Of Cults and Cultures***, *IEEE Software*, *May 2000.*[310]

[310] DOI: 10.1109/MS.2000.10027


Building Accelerated or Tech-Savvy Organizations?
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
JULY/AUGUST 2000
Process Diversity
eXtreme Programming through Dialog 12
Air Traffic Control Software: What's Next? 106
Debunking Software Patent Myths 122
IEEE COMPUTER SOCIETY
http://computer.org

"A '**one size fits all**' approach **doesn't work** in software development. Processes work or are appropriate only under certain conditions."

Ioana Rus, Mikael Lindvall, ***Guest Editors' Introduction: Process Diversity in Software Development****, IEEE Software, July 2000.*[311]

[311] DOI: 10.1109/MS.2000.854063

"The software industry has practiced **pair programming**–two programmers working side by side at one computer on the same problem–for years. But people who haven't tried it often reject the idea as a waste of resources. … pair programming yields **better software products** in **less time**–and **happier**, **more confident** programmers."

Ward Cunningham, Laurie Williams, Robert R. Kessler, Ron Jeffries, ***Strengthening the Case for Pair Programming****, IEEE Software, July 2000.*[312]

[312] DOI: 10.1109/52.854064

"**Scrum** is a process for **incrementally building** software in complex environments. Scrum provides **empirical controls** that allow the development to occur as close to the edge of chaos as the developing organization can tolerate. … During the sprint, the team holds frequent (usually **daily**) Scrum **meetings**. These meetings address the observation made by **Brooks**: 'How does a project get to be a year late? One day at a time.' When the team comes together for a short, daily meeting, **any slip is immediately obvious** to everyone."

Norman S. Janoff, Linda Rising, ***The Scrum Software Development Process for Small Teams****, IEEE Software, July 2000.*[313]

[313] DOI: 10.1109/52.854065

"**Different methodologies** are inevitable, stemming directly from the questions of what constitutes a methodology and what are a methodology's underlying principles. Projects differ according to **size**, **composition**, **priorities**, and **criticality**."

Alistair Cockburn, ***Selecting a Project's Methodology****, IEEE Software, July 2000.*[314]

[314] DOI: 10.1109/52.854070

"Although the organizations included in this section **differ in continents and cultures**, several **common themes** emerge from these reports. All organizations had **quantifiable business targets** and managed their pursuit of these targets empirically."

Bill Curtis, ***Guest Editor's Introduction: The Global Pursuit of Process Maturity***, *IEEE Software, July 2000.*[315]

[315] DOI: 10.1109/MS.2000.854072

"By '**ad hoc**,' most computing people have meant something that is **chaotic and undisciplined**; it became a sort of computing dirty word. But a closer look at the dictionary says that '**ad hoc**' really means '**tailored to the problem at hand**.' Ad hoc approaches might or might not be chaotic, the dictionary is telling us, but what they are really about is using **a best approach** determined not by some kind of project-independent thinking, but by some very **project-focused thinking**."

Robert L. Glass, ***Process Diversity and a Computing Old Wives'/Husbands' Tale****, IEEE Software, July 2000.*[316]

[316] DOI: 10.1109/MS.2000.10036


Malicious IT: The Software vs. The People
IEEE
Software
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
SEPTEMBER/OCTOBER 2000
Engineering in the Small
Safe and Simple Software Cost Analysis 14
Napster: Legal Tortoise, Technology Hare 18
Making Telecommuting Work 124
COMPUTER SOCIETY
http://computer.org

"The Internet and public phone system (upon which the Internet sits) provide an **information highway** that also was **not designed to thwart** 'bad guys.' As a result, today we rely on an infrastructure that **enables** rogue **individuals and nations** to **remotely attack** information assets."

Nancy Mead, Jeffrey Voas, ***Guest Editor's Introduction: Malicious IT****, IEEE Software, September 2000.*[317]

[317] DOI: 10.1109/MS.2000.10040

"**Malicious code** is any code added, changed, or removed from a software system to **intentionally cause harm** or subvert the system's intended function."

Gary McGraw, Greg Morrisett, ***Attacking Malicious Code: A Report to the Infosec Research Council****, IEEE Software, September 2000.*[318]

[318] DOI: 10.1109/52.877857

“**Attacks** can involve numerous attackers targeting many **victims**. Defining what constitutes an attack is difficult because multiple perspectives are involved. The **attacker viewpoint** is typically characterized by **intent** and **risk of exposure**. From a **victim’s perspective**, intrusions are characterized by their **manifestations**, which might or might not include **damage**. “

Julia Allen, John McHugh, Alan Christie, ***Defending Yourself: The Role of Intrusion Detection Systems****, IEEE Software, September 2000.*[319]

[319] DOI: 10.1109/52.877859

"**Jslint** … statically **scans Java source code** looking for potentially **insecure coding practices**. Automated source code scanning tools can help programmers easily **prevent some types of bugs**."

Gary McGraw, Edward W. Felten, John Viega, Tom Mutdosch, ***Statically Scanning Java Code: Finding Security Vulnerabilities****, IEEE Software, September 2000.*[320]

[320] DOI: 10.1109/52.877869

“We now realize that **small-scale software engineering** is **not** just a **degenerate case of large-scale software engineering** but an important subfield in its own right. “

---

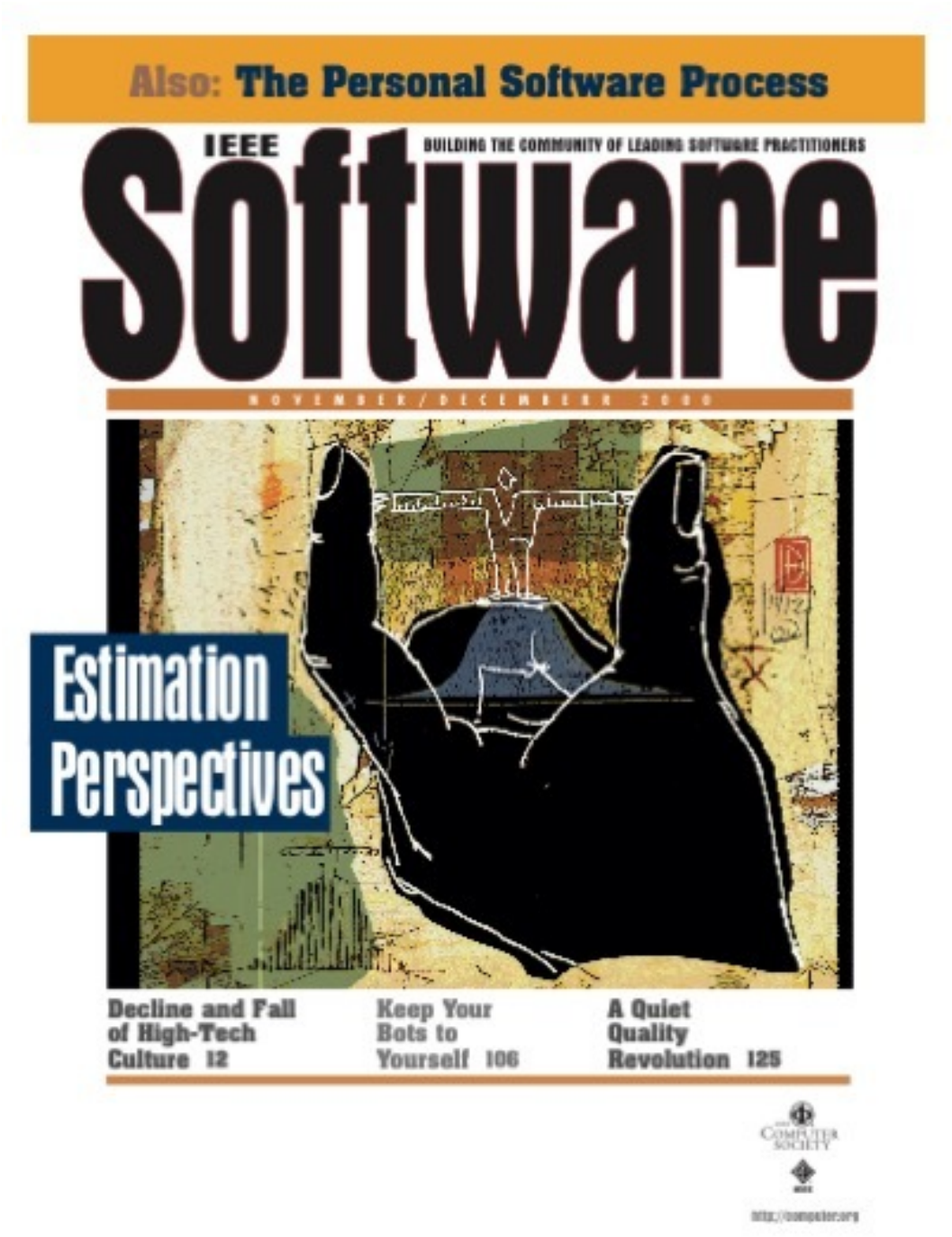
*Robert Ward, Mauri Laitinen, Mohamed Fayad, **Guest Editors' Introduction: Software Engineering in the Small**, IEEE Software, September 2000.*[321]

[321] DOI: 10.1109/MS.2000.10047

"**Computer science research** was where we would have expected **generalized solutions** to arise. But that is simply **not the case**. CS might have begun the domain generalization movement with its compiler-compiling work, but **the world of practice**, and especially the world of vendors, quickly took over that chore—and that dramatic success."

Robert L. Glass, ***The Generalization of an Application Domain***, *IEEE Software, September 2000.*[322]

[322] DOI: 10.1109/MS.2000.10043


Also: The Personal Software Process
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
NOVEMBER/DECEMBER 2000
Estimation Perspectives
Decline and Fall of High-Tech Culture 12
Keep Your Bots to Yourself 106
A Quiet Quality Revolution 125
IEEE COMPUTER SOCIETY
http://computer.org

“**Estimates** have a number of uses, and you can often get both better and simpler estimates if you **keep the use** of your estimate **in mind**.”

Richard E. Fairley, Barry W. Boehm, ***Guest Editors’ Introduction: Software Estimation Perspectives****, IEEE Software, November 2000.*[323]

[323] DOI: 10.1109/MS.2000.895164

“Several **estimation techniques and tools** are available for **predicting the amount of time** and effort needed to develop software systems. Most of these techniques require a wide variety of **input factors**, including **historical data**, system **complexity measures**, the development team’s **level of skill**, any project **constraints**, and an estimate of the **volume of code** (the project’s size). “

James Bielak, ***Improving Size Estimates Using Historical Data***, *IEEE Software, November 2000.*[324]

[324] DOI: 10.1109/52.895165

“**Product quality** directly relates to project **cost** and **schedule estimation**; for example, **undetected defects** in a key work product—such as a requirements document—might lead to time-consuming adjustments. “

Stefan Biffl, ***Using Inspection Data for Defect Estimation****, IEEE Software, November 2000.*[325]

[325] DOI: 10.1109/52.895166

“One of the most important factors in **improving worker** performance is **prompt and explicit feedback**. “

Watts S. Humphrey, ***Guest Editor’s Introduction: The Personal Software Process-Status and Trends***, *IEEE Software*, *November 2000.*[326]

[326] DOI: 10.1109/MS.2000.895171

"'**How important is mathematics** to the software **practitioner**?' According to this research, at least, the answer is not only 'not very much,' it's '**not nearly as much** as we academics have thought.'"

---

*Robert L. Glass, **A New Answer to 'How Important is Mathematics to the Software Practitioner?'**, IEEE Software, November 2000.*[327]

[327] DOI: 10.1109/MS.2000.10052

2001


Also: Who Needs Software Engineering?
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
JANUARY/FEBRUARY 2001
Usability Engineering
Which Way, SQA? 16
India, Software Superpower 77
Better than Guesstimating 86
IEEE COMPUTER SOCIETY
http://computer.org

"**Usability is not a luxury** but a basic ingredient in software systems: People's **productivity** and **comfort** relate directly to the usability of the software they use."

Helmut Windl, Natalia Juristo, Larry Constantine, ***Guest Editors' Introduction: Introducing Usability****, IEEE Software, January 2001.*[328]

[328] DOI: 10.1109/MS.2001.903155

“Contrary to what some might think, **usability** is not just the appearance of the user interface (UI). Usability relates to how the system interacts with the user, and it includes five basic attributes: **learnability**, **efficiency**, user **retention** over time, **error rate**, and **satisfaction**.”

Helmut Windl, Natalia Juristo, Xavier Ferré, Larry Constantine, ***Usability Basics for Software Developers****, IEEE Software, January 2001.*[329]

[329] DOI: 10.1109/52.903160

"**A cost-benefit analysis** might be a necessary first step in introducing usability into your organization or a particular project. In usability cost-benefit analyses, the goal is to estimate the **costs and benefits** of specific usability activities—such as **prototyping**, usability **testing**, heuristic **evaluation**, and so on—and contrast them with the likely **costs of not conducting** the activities."

---

*George M. Donahue, **Usability and the Bottom Line**, IEEE Software, January 2001.*[330]

[330] DOI: 10.1109/52.903161

“Over the last year, I’ve been struck by one of the underlying principles that leads to **better designs**: **remove duplication**. … Often, the **hard part** of eliminating duplication is **spotting it** in the first place. “

---

*Martin Fowler, **Avoiding Repetition**, IEEE Software, January 2001.*[331]

[331] DOI: 10.1109/52.903175


Also: Picking the Right Open Source Software
IEEE
Software
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
MARCH/APRIL 2001
The Global View
When Irish Guys Are Coding 87
How Little Can You Test? 98
So You Wanna Be a Cowboy 112
IEEE COMPUTER SOCIETY
http://computer.org

“**Global Software Development** requires close cooperation of individuals with different **cultural backgrounds**. Cultures differ on many critical dimensions, such as the need for **structure**, attitudes toward **hierarchy**, sense of **time**, and **communication** styles. “

Deependra Moitra, James D. Herbsleb, ***Guest Editors’ Introduction: Global Software Development****, IEEE Software, March 2001.*[332]

[332] DOI: 10.1109/52.914732

"Despite the considerable power of today's **asynchronous technologies** for dispersed work—email, voice mail, online discussion groups, project management tools, Software Configuration Management, and issue and defect-tracking databases—there are still **powerful reasons for synchronous**—if not **face-to-face**—communication. Synchronous communication includes telephone, audio conferencing, videoconferencing, application sharing, and sometimes synchronous online code walkthroughs."

Ritu Agarwal, Erran Carmel, ***Tactical Approaches for Alleviating Distance in Global Software Development****, IEEE Software, March 2001.*[333]

[333] DOI: 10.1109/52.914734

"**Synching** can be problematic because **distance still matters** in our supposedly borderless world. Distance particularly constrains the synching of **tacit knowledge**, **informal information**, and **cultural values**."

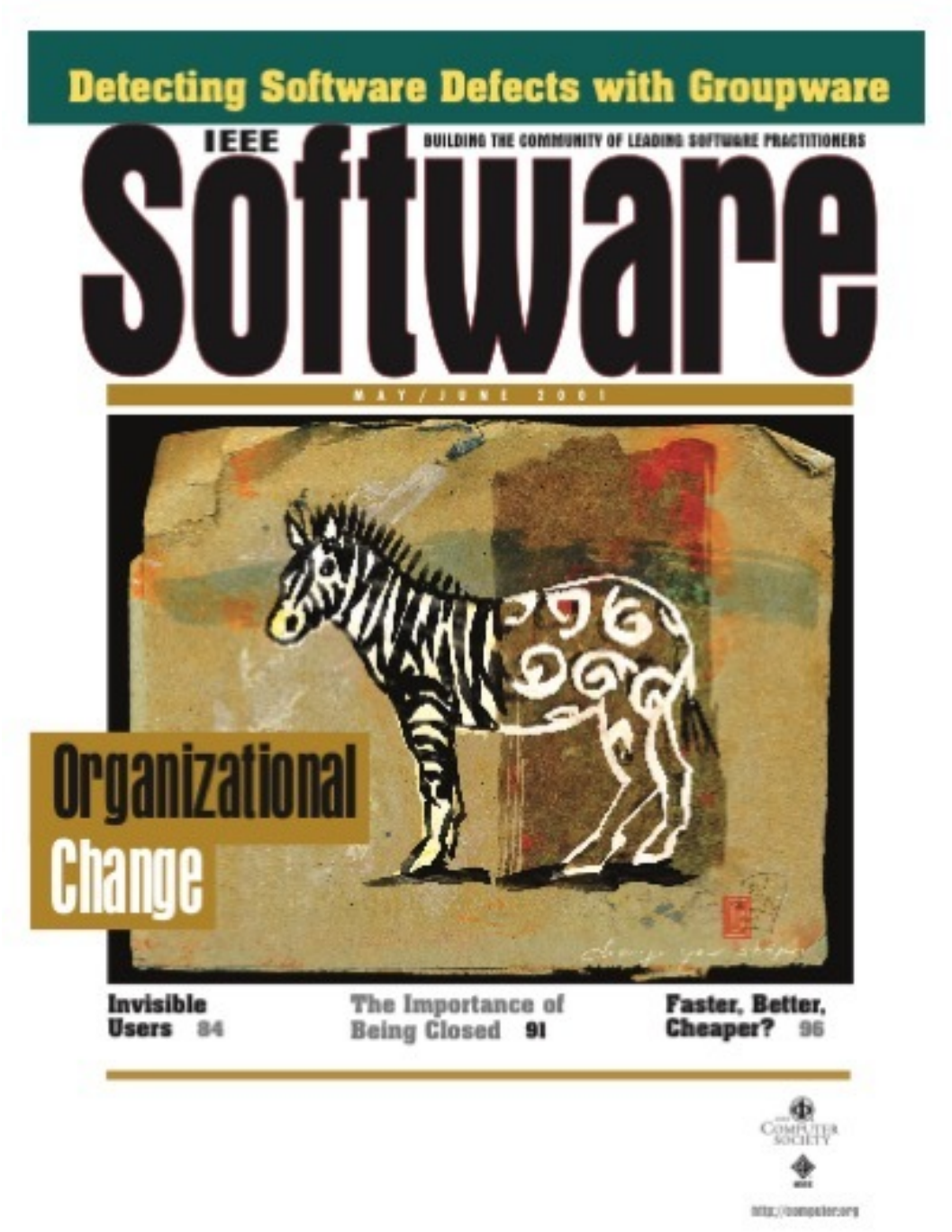
Brian Nicholson, Richard Heeks, Sundeep Sahay, S. Krishna, ***Synching or Sinking: Global Software Outsourcing Relationships****, IEEE Software, March 2001.*[334]

[334] DOI: 10.1109/52.914744

"I learned for myself a design principle that's served me well in software development: **Keep your user interface code separate from everything else**. It's a simple rule, embodied into more than one application framework, but it's often not followed, which causes quite a bit of trouble."

Martin Fowler, ***Separating User Interface Code****, IEEE Software, March 2001.*[335]

[335] DOI: 10.1109/52.914754


Detecting Software Defects with Groupware
IEEE
Software
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
MAY/JUNE 2001
Organizational Change
Invisible Users 84
The Importance of Being Closed 91
Faster, Better, Cheaper? 96
IEEE COMPUTER SOCIETY
http://computer.org

” In the software and information technology industry, **organizational change** has been a way of life. It is quite telling to listen to individuals discussing change in their organizations. Their words frame their philosophies: some **plan and lead** change, others **manage** it, still others **accommodate** change, and many simply try to **cope** with it. “

Ann Miller, ***Guest Editor’s Introduction: Organizational Change****, IEEE Software, May 2001.*[336]

[336] DOI: 10.1109/MS.2001.922720

“Software process improvement efforts will **fail** if we try to make development **processes completely uniform** across an organization.”

Michael Deck, ***Managing Process Diversity While Improving Your Practices***, *IEEE Software*, *May 2001.*[337]

[337] DOI: 10.1109/52.922721

"Traditional **software measurement**, like traditional software process improvement, is **misaligned** with two of the three **basic strategies**, **customer intimacy** and **product innovativeness**. Measurement initiatives can succeed if you understand your organization's strategic objectives and then tailor your measurement practices to fit."

Stan Rifkin, ***What Makes Measuring Software So Hard?****, IEEE Software, May 2001.*[338]

[338] DOI: 10.1109/52.922724

"Pressure to **achieve estimation targets** is common and tends to cause programmers to skip good software process. This constitutes an **absurd result** done for an **absurd reason**."

Robert L. Glass, ***Frequently Forgotten Fundamental Facts about Software Engineering***, *IEEE Software*, *May 2001*.[339]

[339] DOI: 10.1109/MS.2001.922739


Also: Interview with Tom DeMarco
IEEE
Software
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
JULY/AUGUST 2001
Fault Tolerance:
Making Software Behave
Developing Console Game Software 96
Reducing Coupling 102
DoD Weapon Systems Software 105
IEEE COMPUTER SOCIETY
http://computer.org

“Between the late 1960s and early 1990s, the software engineering community strove to formalize schemes that would lead to **perfectly correct software**. Although a noble undertaking at first, it soon became apparent that correct software was, in general, unobtainable. And furthermore, the costs, even if achievable, would be overwhelming. Modern software systems, even **if correct**, can still exhibit **undesirable behaviors** as they execute. “

Jeffrey Voas, ***Guest Editor’s Introduction: Software Fault Tolerance-Making Software Behave****, IEEE Software, July 2001.*[340]

[340] DOI: 10.1109/MS.2001.936212

"**Fault tolerance** is generally implemented by **error detection** and subsequent **system recovery**. Recovery consists of **error handling** (to eliminate errors from the system state) and **fault handling** (to prevent located faults from being activated again). **Fault handling** involves four steps: fault **diagnosis**, fault **isolation**, system **reconfiguration**, and system **reinitialization**."

---

*Jeffrey Voas, **Fault Tolerance**, IEEE Software, July 2001.*[341]

[341] DOI: 10.1109/MS.2001.936218

"There are several ways to describe **coupling**, but it boils down to this: **changing one module** in a program **requires changing another module** … **Duplication** always **implies coupling**, because changing one piece of duplicate code implies changing the other."

---

*Martin Fowler, **Reducing Coupling**, IEEE Software, July 2001.*[342]

[342] DOI: 10.1109/MS.2001.936226

Also: Does Open Source Improve System Security?

IEEE Software

BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

SEPTEMBER/OCTOBER 2001

Benchmarking: Learning from the Masters

IEEE COMPUTER SOCIETY

http://computer.org

“Margaret Boden … identifies basic **types of creative processes**: **exploratory creativity** explores a possible solution space and discovers new ideas, **combinatorial creativity** combines two or more ideas that already exist to create new ideas, and **transformational creativity** changes the solution space to make impossible things possible. Most requirements engineering RE activities are exploratory. “

Alexis Gizikis, Neil Maiden, ***Where Do Requirements Come From?****, IEEE Software, September 2001.*[343]

[343] DOI: 10.1109/52.951486

"Focus on **building an environment** that will **keep your engineers** … Give them **rewarding assignments**, build cohesive and **committed teams**, and know what each of them is doing. Then, every week if possible, show that you appreciate their efforts."

Watts S. Humphrey, ***Engineers Will Tolerate a Lot of Abuse****, IEEE Software, September 2001.*[344]

[344] DOI: 10.1109/52.951487

“Whether you are **benchmarking** an organization or simply a project, it all boils down to **one thing—data**. Do you have the necessary data in your company, and is that data valid and comparable?”

Katrina D. Maxwell, ***Collecting Data for Comparability: Benchmarking Software Development Productivity****, IEEE Software, September 2001.*[345]

[345] DOI: 10.1109/52.951490

“In 1999, an organization contributed a large group of enhancement projects to the **International Software Benchmarking Standards Group’s Data Repository**. The contributing organization received an individual benchmark report for each project, comparing it to the most relevant projects in the repository. … The benchmarking exercise aimed to provide valuable information to the organization and to measure the benchmarking exercise’s effectiveness given the **repository’s anonymous nature**.”

Peter R. Hill, Michael Stringer, Chris Lokan, Terry Wright, ***Organizational Benchmarking Using the ISBSG Data Repository****, IEEE Software, September 2001.*[346]

[346] DOI: 10.1109/52.951491

“Most commercial software producers **guard access to the source code** of their systems, making it difficult for anyone outside their organizations to apply a variety of measures that could potentially improve system security. But since an attacker could also examine public source code to find flaws, would source code access be a net gain or loss for security? … having **source code available** should on balance work **in favor of system security**.”

---

*Michael Caloyannides, Carl Landwehr, Brian Witten, **Does Open Source Improve System Security?**, IEEE Software, September 2001.*[347]

[347] DOI: 10.1109/52.951496

"There are two widely different views on the **nature of software work**: (1) It is easy, is automatable, and **can be done by anyone**. (2) It is **the most complex undertaking** humanity has ever tried."

---

*Robert L. Glass, **A Story about the Creativity Involved in Software Work**, IEEE Software, September 2001.*[348]

[348] DOI: 10.1109/MS.2001.951506


Also: Cracking the 500-Language Problem
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
NOVEMBER/DECEMBER 2001
Reports from the Field
Using Extreme Programming and Other Experiences
Explicit Design 10
The Keys to Knowledge Management 66
Russia's Software Industry 98
COMPUTER SOCIETY
http://computer.org

“Where **design counts** is often not in how the software runs but in **how easy it is to change**. When how it runs is important, **ease of change** can be the biggest factor in ensuring good performance. This drive toward changeability is why it’s so important for a design to **clearly show** what the program does—and how it does it. After all, it’s hard to change something when you **can’t see** what it does.”

---

*Martin Fowler, **To Be Explicit**, IEEE Software, November 2001.*[349]

[349] DOI: 10.1109/52.965796

"**Extreme Programming** XP is not the ultimate silver bullet that offers an answer to all development problems. But it has **gained significant momentum** and an increasing number of software teams are ready to give it a try."

Wolfgang Strigel, ***Guest Editor's Introduction: Reports from the Field–Using Extreme Programming and Other Experiences***, *IEEE Software, November 2001.*[350]

[350] DOI: 10.1109/MS.2001.965797

“The **SW-CMM** focuses on both the **management issues** … **XP**, on the other hand, is a specific **set of practices**—a ‘methodology’—that is effective in the context of small, colocated teams with rapidly changing requirements. **Taken together**, the two methods can create **synergy**, particularly in conjunction with other good engineering and management practices. “

Mark C. Paulk, ***Extreme Programming from a CMM Perspective****, IEEE Software, November 2001.*[351]

[351] DOI: 10.1109/52.965798

“All programmers **learn from experience**. A few are rather fast at it and learn to avoid repeating mistakes after once or twice. Others are slower and repeat mistakes hundreds of times. Most programmers’ behavior falls somewhere in between: They reliably **learn from their mistakes**, but the process is **slow and tedious**. The probability of making a structurally similar mistake again decreases slightly during each of some dozen repetitions. Because of this a programmer often takes years to learn a certain rule—positive or negative—about his or her behavior. “

Lutz Prechelt, ***Accelerating Learning from Experience: Avoiding Defects Faster****, IEEE Software, November 2001.*[352]

[352] DOI: 10.1109/52.965803

"**Extreme Programming** is a **fascinating collection** of elements, some good and some bad."

Robert L. Glass, ***Extreme Programming: The Good, the Bad, and the Bottom Line****, IEEE Software, November 2001.*[353]

[353] DOI: 10.1109/MS.2001.965816

2002

Also: The US Software Industry

IEEE Software

BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

JANUARY/FEBRUARY 2002

Building Software Securely

http://computer.org

IEEE COMPUTER SOCIETY

“Software designers in a **networked world** cannot pretend to work in isolation. **People** are a **critical part** of the full software security equation, and software that makes unrealistic or **unreasonable security-related demands** on users (for example, requiring them to memorize too many passwords that change too often) is software whose security will **inevitably be breached**.”

James A. Whittaker, Chuck Howell, Anup K. Ghosh, ***Building Software Securely from the Ground Up****, IEEE Software, January 2002.*[354]

[354] DOI: 10.1109/MS.2002.976936

“**Buffer overflow vulnerabilities** are perhaps the single most important security problem of the past decade. “

David Larochelle, *David Evans*, ***Improving Security Using Extensible Lightweight Static Analysis***, *IEEE Software*, *January 2002.*[355]

[355] DOI: 10.1109/52.976940

“Shout it from the rooftops! **Computing and software are maturing** into amazing, useful, and — hooray, hooray! — dependable disciplines.”

---

*Robert L. Glass, **Failure Is Looking More like Success These Days**, IEEE Software, January 2002.*[356]

[356] DOI: 10.1109/MS.2002.976953


Managing Requirements for Business Value
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
MARCH/APRIL 2002
Engineering Internet Software
Public versus Published Interfaces p. 18
Software Archaeology p. 20
Long-Term Data Preservation p. 98
http://computer.org
IEEE
COMPUTER SOCIETY

"In **real archaeology**, you're investigating some situation, trying to understand what you're looking at and how it all fits together. To do this, you must be careful to **preserve the artifacts** you find and **respect and understand** the cultural forces that produced them. … **Code** becomes legacy code just about as soon as it's written, and suddenly we have exactly the **same issues as the archaeologists**: What are we looking at? How does it fit in with the rest of the world? And what were they thinking?"

Dave Thomas, Andy Hunt, ***Software Archaeology****, IEEE Software, March 2002.*[357]

[357] DOI: 10.1109/52.991327

“The real departure for **Web-based enterprise applications** is the possibility of **wide-ranging accessibility**. A system that might be deployed in-house at a manufacturing company can now be deployed to all the dealers of that manufacturer’s products. Indeed, it can be deployed to all the customers.”

Martin Fowler, Elizabeth Hendrickson, ***The Software Engineering of Internet Software: Guest Editors’ Introduction****, IEEE Software, March 2002.*[358]

[358] DOI: 10.1109/MS.2002.991328

"In **only four or five years**, the **world wide web** has changed from a **static collection** of HTML web pages to a **dynamic engine** that powers e-commerce, collaborative work, and distribution of information and entertainment. … Web sites that depend on **unreliable software** will **lose customers**, and the businesses could lose much money. Companies that want to do business over the Web must spend resources to ensure high reliability. Indeed, they cannot afford not to."

Jeff Offutt, ***Quality Attributes of Web Software Applications****, IEEE Software, March 2002.*[359]

[359] DOI: 10.1109/52.991329

"Relatively **few design principles** are required to **design scalable systems** … : **divide and conquer** (D&C), **asynchrony**, **encapsulation**, **concurrency**, **parsimony**."

Colleen Roe, Sergio Gonik, ***Server-Side Design Principles for Scalable Internet Systems****, IEEE Software, March 2002.*[360]

[360] DOI: 10.1109/52.991330

“Despite breathless declarations that the Web represents a new paradigm defined by new rules, professional developers are realizing that **lessons learned** in the **pre-Internet days** of software development **still apply**. Web pages are user interfaces, HTML programming is programming, and browser-deployed applications are software systems that can **benefit from basic software engineering principles**.”

Larry L. Constantine, Lucy A.D. Lockwood, ***Usage-Centered Engineering for Web Applications****, IEEE Software, March 2002.*[361]

[361] DOI: 10.1109/52.991331

"Many current **Web technologies** lend themselves to **bad practices**, including **cut-and-paste** reuse, **ad-hoc scripts**, **direct-to-database code**, and **fragmented business logic**. Good design practices are increasingly important in Web development … a **Model-View-Controller** … framework lets developers focus on writing application code instead of dealing with servlets, requests, or session variables."

---

*Alan Knight, Naci Dai, **Objects and the Web**, IEEE Software, March 2002.*[362]

[362] DOI: 10.1109/52.991332

"How to **test output from the server** is another classic problem because of the nature of Web applications. How do you test HTML? Certainly we don't want tests that assert that the output is some **long string of HTML**. … Testing the Web page look is not usually the goal; **testing the data in the output** is. Any **slight change** to the output—for example, a cosmetic change such as making a word appear in red—**should not break an output test**."

Robert Mee, Edward Hieatt, ***Going Faster: Testing The Web Application****, IEEE Software, March 2002.*[363]

[363] DOI: 10.1109/52.991333


Also: Patenting Software Components
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
MAY/JUNE 2002
Knowledge Management
Mock Objects for Your Toolbox p. 22
Brazil's Software Industry p. 84
Ivar Jacobson Interview p. 93
http://computer.org
IEEE
COMPUTER SOCIETY

"It is difficult to make **decisions about performance** from **just looking at the design**. Rather, you have to actually **run** the code and **measure performance**."

---

*Martin Fowler, **Yet Another Optimization Article**, IEEE Software, May 2002.*[364]

[364] DOI: 10.1109/MS.2002.1003448

"Using **mock objects**, you can **test code in splendid isolation**, simulating all those messy real-world things that would otherwise make automated testing impossible. And, as with many other testing practices, the discipline of using mock objects **can improve your code's structure**."

---

*Dave Thomas, Andy Hunte, **Mock Objects**, IEEE Software, May 2002.*[365]

[365] DOI: 10.1109/MS.2002.1003449

“Software organizations’ main assets are not plants, buildings, or expensive machines. A software organization’s main asset is its **intellectual capital**, as it is in sectors such as consulting, law, investment banking, and advertising. The major problem with intellectual capital is that **it has legs** and walks home every day. At the same rate experience walks out the door, inexperience walks in the door. Whether or not many software organizations admit it, they face the challenge of **sustaining the level of competence** needed to win contracts and fulfill undertakings.”

Ioana Rus, Mikael Lindvall, ***Guest Editors’ Introduction: Knowledge Management in Software Engineering****, IEEE Software, May 2002.*[366]

[366] DOI: 10.1109/MS.2002.1003450

“An emerging trend is to develop **knowledge management** and **knowledge sharing initiatives** within organizations. For example, **NASA** formed a Knowledge Management Team, comprised of NASA representatives. NASA Goddard Space Flight Center (GSFC) has several knowledge management initiatives underway on the expert and knowledge retention side.”

---

*Jay Liebowitz, **A Look at NASA Goddard Space Flight Center’s Knowledge Management Initiatives**, IEEE Software, May 2002.*[367]

[367] DOI: 10.1109/MS.2002.1003451

“**Postmortem analysis** is a practical method for initiating knowledge management by **capturing experience** and improvement suggestions from completed projects.”

Andreas Birk, Torgeir Dingsøyr, Tor Stålhane, ***Postmortem: Never Leave a Project without It****, IEEE Software, May 2002.*[368]

[368] DOI: 10.1109/MS.2002.1003452

“In an effort to improve software development and acquisition processes and explicitly reuse knowledge from previous software projects, **DaimlerChrysler** created a **Software Experience Center**.”

Kurt Schneider, Jan-Peter von Hunnius, Victor Basili, ***Experience in Implementing a Learning Software Organization****, IEEE Software, May 2002.*[369]

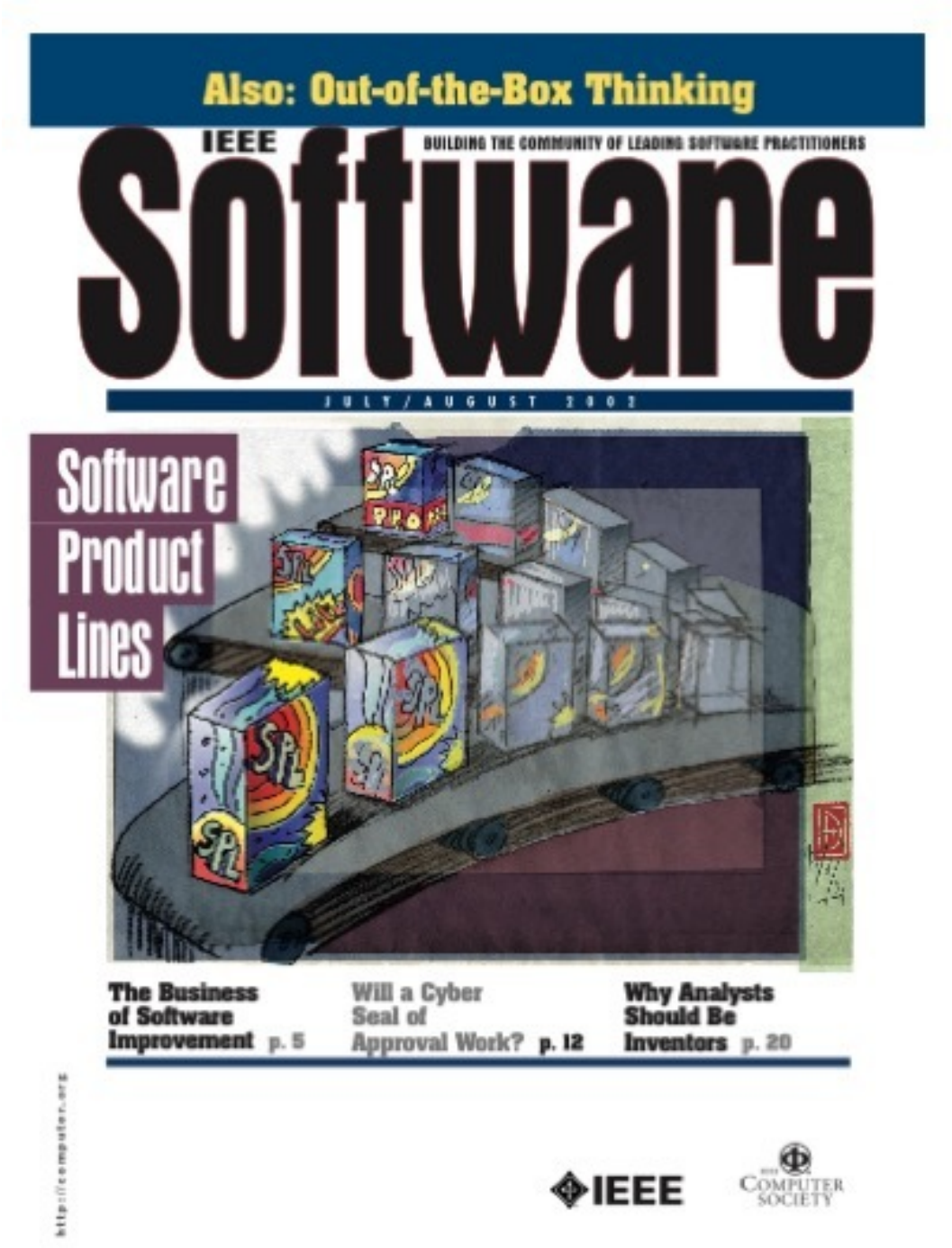
[369] DOI: 10.1109/MS.2002.1003453

"The software field has been subjected, over the years, to **excessive claims of benefits** for almost every new technology. Fourth-generation languages were to lead to '**programming without programmers**,' CASE tools would bring about '**the automation of programming**,' and object orientation was to be a dramatic new methodological approach to systems development that would replace all the other, older, methodologies."

---

*Robert L. Glass, **The Naturalness of Object Orientation: Beating a Dead Horse?**, IEEE Software, May 2002.*[370]

[370] DOI: 10.1109/MS.2002.1003467


Also: Out-of-the-Box Thinking
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
JULY/AUGUST 2002
Software Product Lines
The Business of Software Improvement p. 5
Will a Cyber Seal of Approval Work? p. 12
Why Analysts Should Be Inventors p. 20
http://computer.org
IEEE
COMPUTER SOCIETY

“A **product line**’s scope is specified such that the products have a **high degree of commonality**. A product line organization realizes economically significant reuse…”

John D. McGregor, Linda M. Northrop, Salah Jarrad, Klaus Pohl, ***Guest Editors’ Introduction: Initiating Software Product Lines****, IEEE Software, July 2002.*[371]

[371] DOI: 10.1109/MS.2002.1020282

“**Product line software engineering** is an emerging paradigm that guides organizations toward **developing products from core assets** instead of developing them one by one from scratch.”

Kyo C. Kang, Jaejoon Lee, Patrick Donohoe, ***Feature-Oriented Product Line Engineering****, IEEE Software, July 2002.*[372]

[372] DOI: 10.1109/MS.2002.1020288

“Developers at **Nokia** recently initiated and used a product line to create and deliver **mobile browser products**. They learned that, **to succeed**, a software product line must be **product and application driven**, rather than reuse or platform driven. “

---

*Ari Jaaksi, **Developing Mobile Browsers in a Product Line**, IEEE Software, July 2002.*[373]

[373] DOI: 10.1109/MS.2002.1020290

“**The discipline of completeness** is a willingness and ability to search for and deal not just with every **conceivable mode** of failure but with as many **inconceivable mode**s as you can find through exploration and testing.”

Terry Bollinger, ***Guest Editor’s Introduction: Breaking Out of the Software Engineering Mind-Mold****, IEEE Software, July 2002.*[374]

[374] DOI: 10.1109/MS.2002.1020294

“Two dichotomies characterize software process improvement efforts and approaches: **disciplined** vs. **creative** work and **procurer risks** vs. **user satisfaction**. “

Reidar Conradi, Alfonso Fuggetta, ***Improving Software Process Improvement****, IEEE Software, July 2002.*[375]

[375] DOI: 10.1109/MS.2002.1020295

“**Books** on the subject **favor the ‘light’ side** of the discipline: **project management**, software **process improvement**, schedule and cost estimation, and so forth. The real **technology** necessary to build software is often **described abstractly**, given as obvious, **or ignored** altogether. But software **development is a fundamentally technical problem** for which management solutions can only be partially effective.”

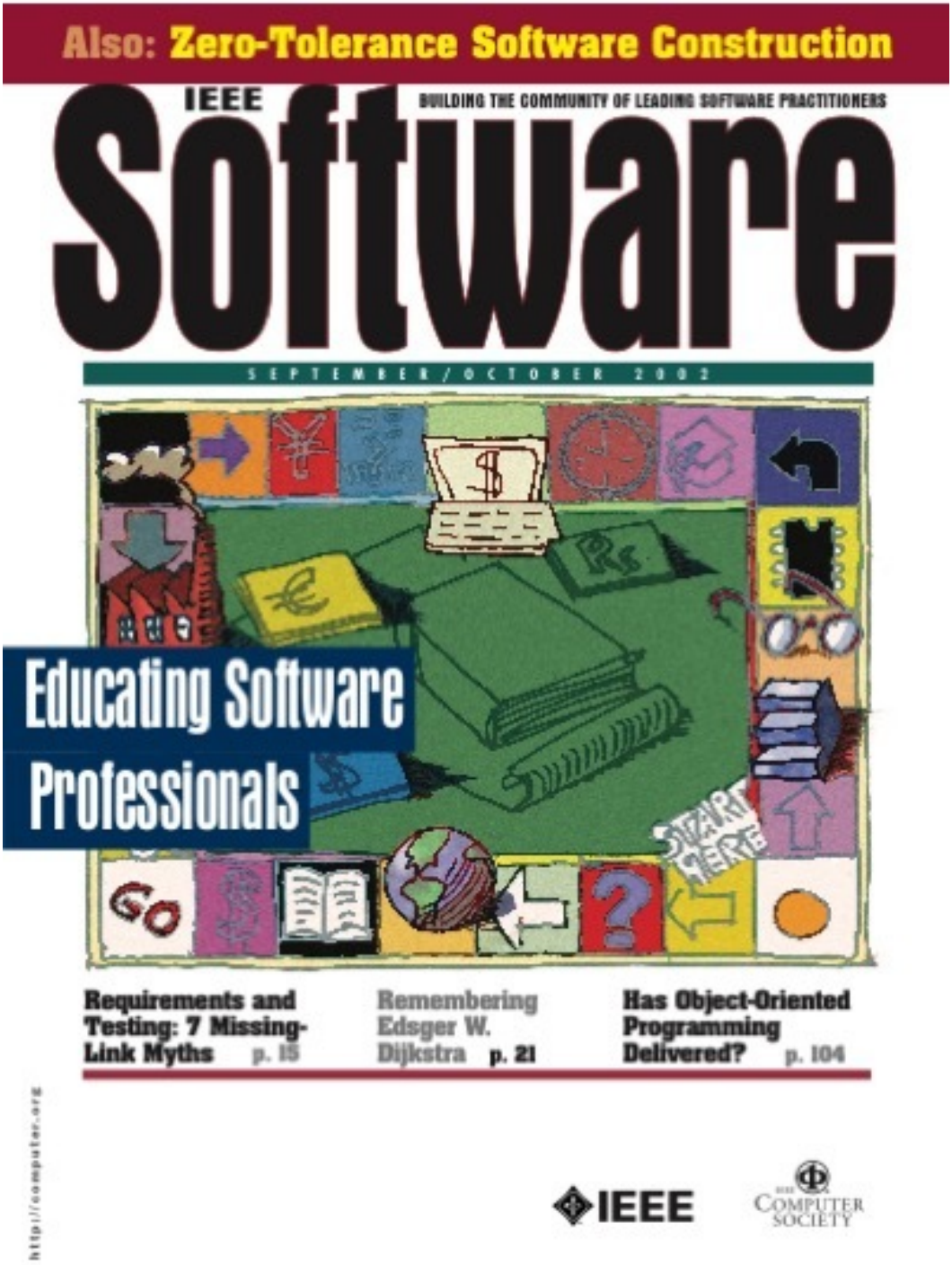
James A. Whittaker, Steven Atkin, ***Software Engineering is Not Enough****, IEEE Software, July 2002.*[376]

[376] DOI: 10.1109/MS.2002.1020297

"**Software measurement** has the potential to play an important role in risk management during product development. **Metrics** incorporated into **predictive models** can give **advanced warning** of potential risks."

Norman Fenton, Paul Krause, Martin Neil, ***Software Measurement: Uncertainty and Causal Modeling****, IEEE Software, July 2002.*[377]

[377] DOI: 10.1109/MS.2002.1020298


Also: Zero-Tolerance Software Construction
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
SEPTEMBER/OCTOBER 2002
Educating Software Professionals
Requirements and Testing: 7 Missing-Link Myths p. 15
Remembering Edsger W. Dijkstra p. 21
Has Object-Oriented Programming Delivered? p. 104
http://computer.org
IEEE
COMPUTER SOCIETY

"A **good article** says something **new** or says something **old in a new way.**"

Steve McConnell, ***From the Editor: How to Write a Good Technical Article***, *IEEE Software*, *September 2002.*[378]

[378] DOI: 10.1109/MS.2002.10006

"The **steady decline in computer science and engineering enrollments** suggests that these more rigorous methods must be justified or student enrollments will continue to drop."

Thomas B. Hilburn, Watts S. Humphrey, ***The Impending Changes in Software Education****, IEEE Software, September 2002.*[379]

[379] DOI: 10.1109/MS.2002.1032848

"We must foster **stronger communication** between diverse groups, such as various **faculty groups**, and between **universities** and **industry**. Myths tend to develop when there is little communication or when the communication that exists reflects our **preconceived notions** rather than **objective assessment**."

Donald J. Bagert, Hossein Saiedian, Nancy R. Mead, ***Software Engineering Programs: Dispelling the Myths and Misconceptions****, IEEE Software, September 2002.*[380]

[380] DOI: 10.1109/MS.2002.1032852

"A **difficult thing** to achieve in a curriculum **is realism** — real products signifying tangible, relevant achievements and real people signifying collaborative effort."

Andrew Macfarlane, Helen Hays, Ken Surendran, ***Simulating a Software Engineering Apprenticeship****, IEEE Software, September 2002.*[381]

[381] DOI: 10.1109/MS.2002.1032854

“Common sense says that, from time to time, we in the software field ought to be **stepping on the brakes**, slowing down to **learn the lessons** we have just rushed through. And **project retrospectives** would be a good thing to do while that mad, headlong pace has been slowed.”

Robert L. Glass, ***Project Retrospectives, and Why They Never Happen****, IEEE Software, September 2002.*[382]

[382] DOI: 10.1109/MS.2002.1032872

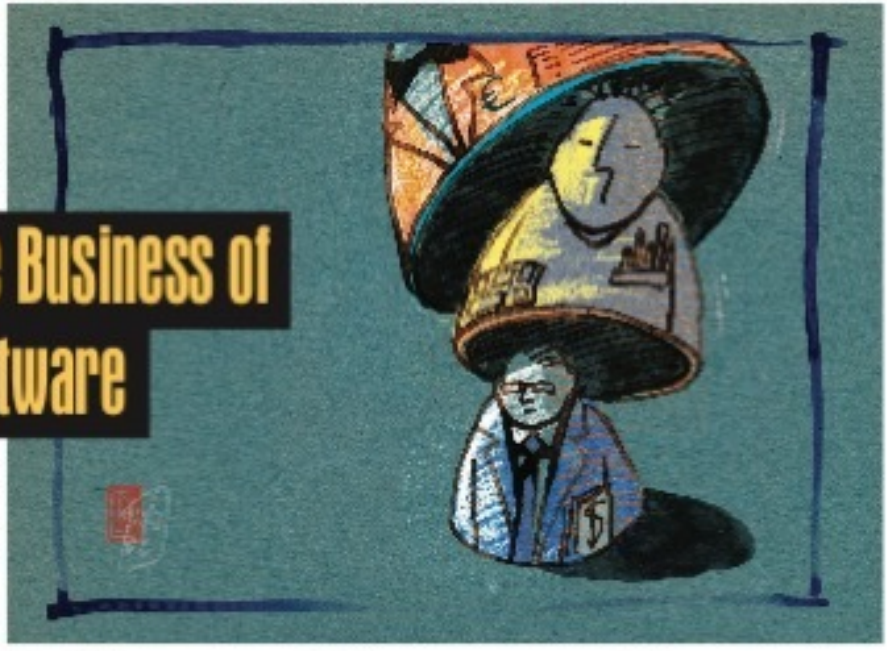
Also: Attaining CMM Level 5

IEEE

BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

Software

NOVEMBER/DECEMBER 2002



The Business of Software

http://computer.org

IEEE COMPUTER SOCIETY

“We can **use the metadata** in two ways: **reflective programming** and **code generation**. … the obvious question is when to use each style. … Many people find **reflection somewhat hard to use**, and it might **defeat some of your environment’s tooling**, such as intelligent reference searches and automated refactorings. .. You need **discipline** to ensure that developers don’t **hand-edit the generated files**. “

---

*Martin Fowler, **Using Metadata**, IEEE Software, November 2002.*[383]

[383] DOI: 10.1109/MS.2002.1049381

“Not only is **software increasing** in size, complexity, and percentage of functionality, it is increasing in contribution to the **balance sheet** and **profit-and-loss statements**. “

Ann Miller, Christof Ebert, ***Guest Editors? Introduction: Software Engineering as a Business****, IEEE Software, November 2002.*[384]

[384] DOI: 10.1109/MS.2002.1049382

"Traditional performance metrics no longer suffice to measure results and guide organizations in today's fast-changing economies. Firms need to **link performance metrics** to **strategic objectives** that will promote positive future results and accurately capture past performance."

---

*Steven Mair, **A Balanced Scorecard for a Small Software Group**, IEEE Software, November 2002.*[385]

[385] DOI: 10.1109/MS.2002.1049383

“Most companies now find that **retiring an existing software product** is **nearly impossible**. To build a replacement, you need requirements that **match the product’s current version**, and they probably don’t exist! They’re **not in the documentation**, because it wasn’t kept up to date. You won’t get them from the original customers, users, or developers, because **those folks are long gone** …”

---

*Robert Glass, **Predicting Future Maintenance Cost, and How We?re Doing It Wrong**, IEEE Software, November 2002.*[386]

[386] DOI: 10.1109/MS.2002.1049400

2003

Also: C# and .NET—Ready for Real Time?

IEEE Software

BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

JANUARY/FEBRUARY 2003

Requirements Engineering

http://computer.org

IEEE

THE COMPUTER SOCIETY

“When in doubt, **make a new type**.”

---

*Martin Fowler, **When to Make a Type**, IEEE Software, January 2003.*[387]

[387] DOI: 10.1109/MS.2003.1159023

"Today, many organizations and companies have established **explicit roles for requirements engineers**. Adequate techniques and tools for **RE tasks** (such as **elicitation**, **validation**, **negotiation**, **specification**, and **documentation**) have emerged and continuously been improved based on industrial feedback."

Eric Dubois, Klaus Pohl, ***Guest Editors' Introduction: RE 02–A Major Step toward a Mature Requirements Engineering Community****, IEEE Software, January 2003.*[388]

[388] DOI: 10.1109/MS.2003.1159024

"9126-1 quality standard … we selected for the following reasons: (1) Due to its generic nature, the standard **fixes some high-level quality concepts**, and therefore quality models can be **tailored to specific package domains**. This is a crucial point, because quality models can dramatically differ from one domain to another. (2) The standard lets us **create hierarchies** of quality features, which are essential for building **structured quality models**. (3) The standard is **widespread**."

Juan Pablo Carvallo, Xavier Franch, ***Using Quality Models in Software Package Selection****, IEEE Software, January 2003.*[389]

[389] DOI: 10.1109/MS.2003.1159027

"**A misuse case** is the **negative** form of **a use case**; it documents a **negative** scenario. Its actor is an agent with hostile intent toward the system under design. The relationships between use and misuse cases **document threats** and **their mitigations**."

Ian Alexander, ***Misuse Cases: Use Cases with Hostile Intent***, *IEEE Software*, *January 2003*.[390]

[390] DOI: 10.1109/MS.2003.1159030


Also: Accelerating COTS Middleware Acquisition
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
MARCH/APRIL 2003
Process
The Marriage of Research and Practice p. 5
Pragmatic Software Configuration Management p. 15
What You Don't Measure Can Hurt You p. 49
http://computer.org
IEEE
COMPUTER SOCIETY

“I consider **four models** for **software system ownership** — that is, models for assigning software systems to their human owners: 1. **Product specialist**: A single individual manages all code with occasional help from other individuals. 2. **Subsystem ownership**: Each subsystem has a specific owner, and each team member owns one or more subsystems. 3. **Chief architect**: A chief programmer (architect) has primary ownership of all code. The team takes supporting roles in fleshing out the team leader’s vision. 4. **Collective ownership**: All code is collectively owned. Schedules and responsibilities are set such that every team member has a chance to contribute to every subsystem and is free to work across subsystems as needed.”

---

*Martin E. Nordberg III, **Managing Code Ownership**, IEEE Software, March 2003.*[391]

[391] DOI: 10.1109/MS.2003.1184163

"The initial **modeling activity** is a **structural decomposition** that continues toward the existing **components' granularity level**. Iteration will occur through alternate decomposition and composition activities until the specifications of **abstract** modules **agree with existing** components."

Ali H. Dogru, Murat M. Tanik, ***A Process Model for Component-Oriented Software Engineering****, IEEE Software, March 2003.*[392]

[392] DOI: 10.1109/MS.2003.1184164

“**Tata Consultancy Services** blended **Six Sigma** concepts with the various **SW-CMM** Key Process Areas into a quality management system that has helped it to improve its **customer focus** and sustain the process improvement initiatives by explicitly **linking** them to **business goals**.”

Mala Murugappan, Gargi Keeni, ***Blending CMM and Six Sigma to Meet Business Goals***, *IEEE Software, March 2003.*[393]

[393] DOI: 10.1109/MS.2003.1184165

“In **manufacturing**, the **observed** and **actual number** of **defects** is not significantly different. In **software development**, these two numbers routinely **vary significantly**.
“

Nancy Eickelmann, Animesh Anant, ***Statistical Process Control: What You Don?t Measure Can Hurt You!****, IEEE Software, March 2003.*[394]

[394] DOI: 10.1109/MS.2003.1184166

“**Patterns** are **not good or bad**—rather, they’re either **appropriate or not** for some situations. I don’t think it’s wrong to experiment with using a pattern when you’re unsure, but you should be **prepared to rip it out** if it doesn’t contribute enough.”

---

*Martin Fowler, **Patterns**, IEEE Software, March 2003.*[395]

[395] DOI: 10.1109/MS.2003.1184168

“Colleagues in my research group and in collaborating institutions typically model software designs using graphical tools such as **Rational Rose**, **Together**, and **Visio**. I often witness them toiling to adjust a graph’s appearance with the mouse or laboriously visiting each class to change a single field’s type. This need not be so. Design models should be **composed textually**, and graphs should be **automatically generated**. “

Diomidis Spinellis, ***On the Declarative Specification of Models****, IEEE Software, March 2003.*[396]

[396] DOI: 10.1109/MS.2003.1184181

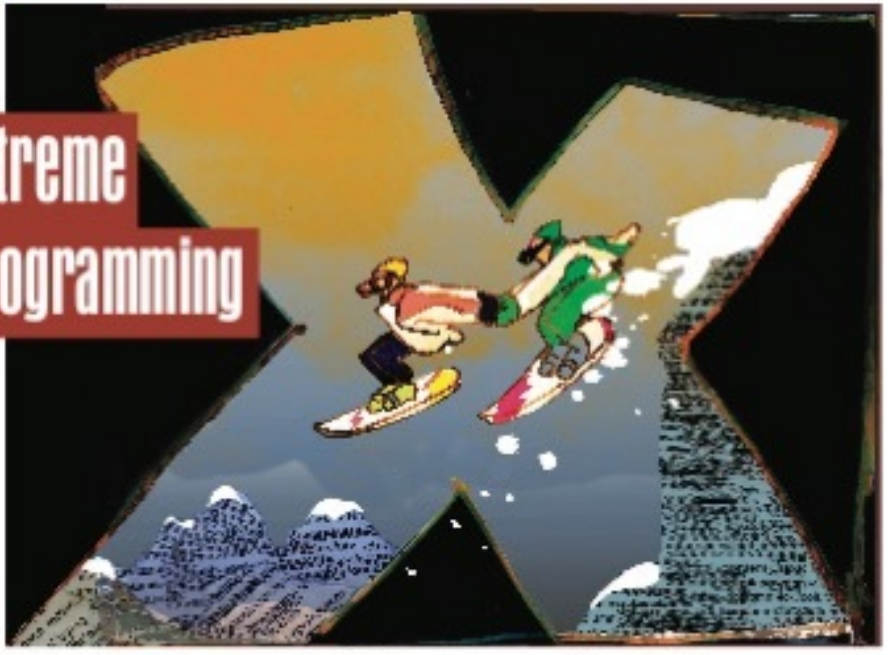
Also: Assuring Software Quality Assurance

IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

IEEE Software

MAY/JUNE 2003

Extreme Programming



http://computer.org

IEEE COMPUTER SOCIETY

“The **Extreme Programming** methodology exudes this same advice: **Be communicative with everyone** on the team—including **customers**, end **users**, and **business** folks. Consistently and doggedly strive to understand and deliver what your **customers want** with the highest possible quality. **Frequently offer feedback**—a one-minute manager would never hold back until a formal appraisal cycle to give feedback to employees. “

Laurie Williams, ***Guest Editor’s Introduction: The XP Programmer–The Few-Minutes Programmer***, *IEEE Software*, *May 2003.*[397]

[397] DOI: 10.1109/MS.2003.1196315

“**ThoughtWorks** introduced **Extreme Programming** into an organization and successfully completed a bleeding-edge technology project **with client staff** that had no previous experience using an agiledevelopment approach. “

Jonathan Rasmusson, ***Introducing XP into Greenfield Projects: Lessons Learned***, *IEEE Software*, *May 2003.*[398]

[398] DOI: 10.1109/MS.2003.1196316

"The **cultural** environment at a **government research center** differs from the **customer-centric** business view. Consequently, eight of XP's 12 practices are seemingly **incompatible** with the existing research **culture**. … Despite initial awkwardness … **XP can function** in situations for which it appears to be ill suited."

William L. Kleb, William A. Wood, ***Exploring XP for Scientific Research****, IEEE Software, May 2003.*[399]

[399] DOI: 10.1109/MS.2003.1196317

"We can trace **SQA's roots** back to the **1960s**, when **IBM** used the term in the context of **final product testing**. SQA also has deep roots in the **US Department of Defense**, which created a family of military specification standards required of all software vendors seeking DoD contracts (the most famous of which is probably MIL-STD 2167A). However, **not all people** buy into the **belief** that **SQA is needed** or is scientific …"

Jeffrey Voas, ***Guest Editor's Introduction: Assuring Software Quality Assurance****, IEEE Software, May 2003.*[400]

[400] DOI: 10.1109/MS.2003.1196320

"**Statistical process control** tools enable proactive software process management. One such tool, **the control chart**, can be used for managing, controlling, and improving the code review process."

S.K. Pillai, Alice Leslie Jacob, ***Statistical Process Control to Improve Coding and Code Review****, IEEE Software, May 2003.*[401]

[401] DOI: 10.1109/MS.2003.1196321

“One of the most important issues in analyzing safety-critical systems is **code verification** through an **inspection checklist**, whose items must be applied to **the source code**. The attention given to this list will help ensure obedience to **good coding rules** and represents an important factor in the design of safety-critical systems.”

Jorge Rady de Almeida Jr., S?rgio Miranda Paz, Jo?o Batista Camargo Jr., Bruno Abrantes Basseto, ***Best Practices in Code Inspection for Safety-Critical Software****, IEEE Software, May 2003.*[402]

[402] DOI: 10.1109/MS.2003.1196322

".. **Accelerate the testing** of scheduled functions by triggering them through **automated tests**, either by periodically **advancing the system clock** or through a **programmatic event interface**."

---

*Vaughn T. Rokosz, **Long-Term Testing in a Short-Term World**, IEEE Software, May 2003.*[403]

[403] DOI: 10.1109/MS.2003.1196323

“Although **rigorous measurement** has become a necessity in the software industry, many measurement programs **fail to deliver** any real **benefit to software managers**. The required data is often missing or invalid, or it just arrives too late.”

---

*Jim Lawler, Barbara Kitchenham, **Measurement Modeling Technology**, IEEE Software, May 2003.*[404]

[404] DOI: 10.1109/MS.2003.1196324

“That we question the hype spewing from the hype purveyors, especially vendors. That we question the **advocacy** spewing **from all too many computer science researchers**, who seem to feel that research papers should ‘**conceive** of a concept, **advocate** the concept, and **scold** practitioners who refuse to use the concept.’”

---

*Robert L. Glass, **Questioning the Software Engineering Unquestionables**, IEEE Software, May 2003.*[405]

[405] DOI: 10.1109/MS.2003.1196338

Increasing UML's Power

IEEE Software

BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

JULY/AUGUST 2003

Software Inspection

http://computer.org

IEEE COMPUTER SOCIETY

“Despite more than 30 years’ effort to improve software quality, companies still release programs containing **numerous errors**. Many major products have **thousands of bugs**. It’s not for lack of trying; all major software developers stress software quality assurance and try to remove bugs before release. The problem is the **code’s complexity**. It’s easy to review code but **fail to notice significant errors**.”

David L. Parnas, Mark Lawford, ***Guest Editors’ Introduction: Inspection’s Role in Software Quality Assurance****, IEEE Software, July 2003.*[406]

[406] DOI: 10.1109/MS.2003.1207449

"**Reading techniques** must specifically address delocalization–the distribution of related functionality throughout an object-oriented system–and the fact that the **static (compile time)** and **dynamic (run time) views** of an object-oriented system are **largely distinct**."

Marc Roper, Alastair Dunsmore, Murray Wood, ***Practical Code Inspection Techniques for Object-Oriented Systems: An Experimental Comparison****, IEEE Software, July 2003.*[407]

[407] DOI: 10.1109/MS.2003.1207450

"Principles from **software inspection**, **use cases**, and operational profile testing are combined into the **usage-based reading** technique. The goal is to provide an efficient reading technique for software inspections, which takes the user viewpoint on the software and the faults it might contain. The user reads, for example, a design document guided by **prioritized use cases**. An experimental evaluation shows that the UBR method is **more effective** and efficient in finding **faults, critical to the user**, compared to checklist-based methods."

Claes Wohlin, Thomas Thelin, Per Runeson, ***Prioritized Use Cases as a Vehicle for Software Inspections****, IEEE Software, July 2003.*[408]

[408] DOI: 10.1109/MS.2003.1207451

"**Software inspection** reduces the number of defects early in the software life cycle. **Cost savings** are realized because errors are significantly more expensive to eliminate with each successive development phase. An important part of the inspection process is a detailed inspection of the source code …. Recent advances in research on **static program analysis** can be used to address this aspect of **software inspection**."

Paul Anderson, Tim Teitelbaum, Thomas Reps, Mark Zarins, ***Tool Support for Fine-Grained Software Inspection****, IEEE Software, July 2003.*[409]

[409] DOI: 10.1109/MS.2003.1207453

"We can divide software systems architecturally along two broad dimensions. The first is the **tarchitecture** or '**technical architecture**' and the second is the **marketecture** or '**marketing architecture**.' I refer to the traditional software architect or chief technologist as the **tarchitect** and the product-marketing manager, business manager, or program manager responsible for the system as the **marketect**."

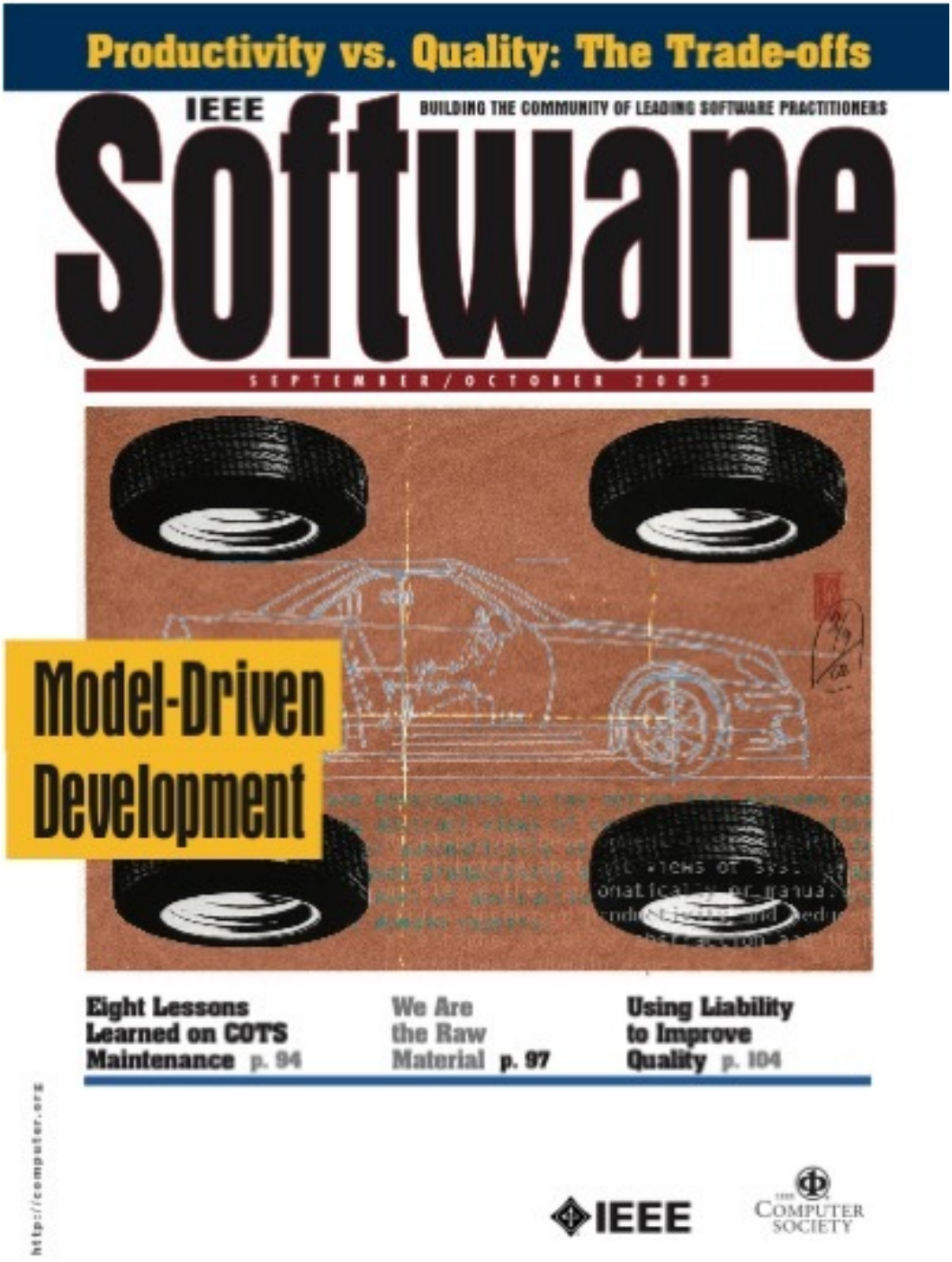
Luke Hohmann, ***The Difference between Marketecture and Tarchitecture****, IEEE Software, July 2003.*[410]

[410] DOI: 10.1109/MS.2003.1207454

"Let's allow SE programs to **hire the best-qualified candidates** regardless of whether they've acquired a PhD."

Robert L. Glass, ***A Big Problem in Academic Software Engineering and a Potential Outside-the-Box Solution****, IEEE Software, July 2003.*[411]

[411] DOI: 10.1109/MS.2003.1207486


Productivity vs. Quality: The Trade-offs
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
SEPTEMBER/OCTOBER 2003
Model-Driven Development
Eight Lessons Learned on COTS Maintenance p. 94
We Are the Raw Material p. 97
Using Liability to Improve Quality p. 104
http://computer.org
IEEE
COMPUTER SOCIETY

"I think that one of an **architect's** most important **tasks** is to remove architecture by finding ways to **eliminate irreversibility** in software designs."

Martin Fowler, ***Who Needs an Architect?****, IEEE Software, September 2003.*[412]

[412] DOI: 10.1109/MS.2003.1231144

"**Model-driven development** is simply the notion that we can construct a **model of a system** that we can then **transform into the real thing**."

Anthony N. Clark, Takao Futagami, Stephen J. Mellor, ***Guest Editors' Introduction: Model-Driven Development****, IEEE Software, September 2003.*[413]

[413] DOI: 10.1109/MS.2003.1231145

"The potential benefits of **using models** are significantly greater in **software** than in **other engineering disciplines** because of the potential for a seamless link between models and the systems they represent. Unfortunately, models have **rarely produced anticipated benefits**. The key lies in **resolving pragmatic issues** related to the artifacts and culture of the previous generation of software technologies."

Bran Selic, ***The Pragmatics of Model-Driven Development****, IEEE Software, September 2003.*[414]

[414] DOI: 10.1109/MS.2003.1231146

"A **model** is a **set of statements** about some **system under study**. Here, **statement** means some expression about the SUS that can be considered **true or false** (although no truth value has to necessarily be assigned at any particular point in time). We can use a model **to describe** an SUS. In this case, we consider the model **correct** if all its statements are true for the SUS. … Alternatively, we can use a **model as a specification** for an SUS or a class of SUS. In this case, we consider a specific SUS **valid** relative to this specification if no statement in the model is false for the SUS. "

---

*Ed Seidewitz, **What Models Mean**, IEEE Software, September 2003.*[415]

[415] DOI: 10.1109/MS.2003.1231147

“**UML** can be used in **many formats**, including presented as **text**, parsed into a **standardized repository**, and **compiled** to multiple programming languages.”

---

*Conrad Bock, **UML without Pictures**, IEEE Software, September 2003.*[416]

[416] DOI: 10.1109/MS.2003.1231148

“It’s helpful to identify two separate orthogonal dimensions of metamodeling, giving rise to two distinct forms of instantiation. One dimension is concerned with **language definition** and hence uses **linguistic instantiation**. The other dimension is concerned with **domain definition** and thus uses **ontological instantiation**. “

Colin Atkinson, Thomas K?, ***Model-Driven Development: A Metamodeling Foundation****, IEEE Software, September 2003.*[417]

[417] DOI: 10.1109/MS.2003.1231149


20 YEARS!
20th Anniversary Issue
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
NOVEMBER/DECEMBER 2003
The State of the Practice
Past EICs Reflect on 20 Years p. 5
Project Failures in Small Companies p. 94
The Compleat Requirements Analyste p. 99
http://computer.org
IEEE
COMPUTER SOCIETY

“For most of software engineering’s history, **authors** have eagerly **told practitioners** what they **ought to be doing**. But **rarely** have those ‘oughts’ been **predicated** on what practitioners **actually are doing**. … Defining the **state of the art** (which I identify as theory + ‘best’ **practice**) is **easy**; conferences, journals, and books do that for us all the time. Yet defining the state of the **practice** is **difficult**. “

---

*Robert L. Glass, **Guest Editor’s Introduction: The State of the Practice of Software Engineering**, IEEE Software, November 2003.*[418]

[418] DOI: 10.1109/MS.2003.1241361

"We **didn't find** that any specific design **method** or programming **language** guaranteed either a **successful** or **troubled** project outcome. … **Good quality control** is the **best** overall **indicator** of a successful project."

Capers Jones, ***Variations in Software Development Practices***, *IEEE Software*, *November 2003.*[419]

[419] DOI: 10.1109/MS.2003.1241362

“No **Indian** or **Japanese** company has yet to make any real global mark in widely recognized software innovation, long the province of **US** and a **few European** software firms.”

Michael Cusumano, Bill Crandall, Alan MacCormack, Chris F. Kemerer, ***Software Development Worldwide: The State of the Practice****, IEEE Software, November 2003.*[420]

[420] DOI: 10.1109/MS.2003.1241363

"The studies confirm the widely held belief that most **software engineers don't update** most software **documentation** in a timely manner. The only notable exception is documentation **types** that are **highly structured** and easy to maintain, such as test cases and inline comments."

---

*Andrew Forward, Timothy C. Lethbridge, Janice Singer, **How Software Engineers Use Documentation: The State of the Practice**, IEEE Software, November 2003.*[421]

[421] DOI: 10.1109/MS.2003.1241364

“One of the most important things about **good design** is **modularity** — dividing a system into separate pieces so that you can **modify one module** without the **changes rippling** all over the system. Early on, David **Parnas** observed that modules should be arranged **around system secrets**, each module hiding its secret from the other modules. Then if the secret thing changes, you avoid a ripple effect. One of the **most common secrets** to hide these days is **data structures**. “

---

*Martin Fowler, **Data Access Routines**, IEEE Software, November 2003.*[422]

[422] DOI: 10.1109/MS.2003.1241375

2004


Quality Resolutions for the New Year
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
JANUARY/FEBRUARY 2004
Developing with Open Source Software
7 Hot Outsourcing Practices p. 14
3 Ways to Save a Project p. 18
Automated Bug Tracking Pros & Cons p. 100
www.computer.org
IEEE
COMPUTER SOCIETY

"In a **criminal trial** (at least in the US), the **failure to follow** an established **best practice** in an investigation could result in an **acquittal**. … the **American Society for Quality's definition** probably comes closest to what we in software development mean when we use the term. The ASQ defines 'best practice' as a **superior method** or innovative practice that contributes to the improved performance of an organization, usually **recognized** as 'best' **by other peer** organizations."

Warren Harrison, ***From the Editor: Best Practices–Who Says?****, IEEE Software, January 2004.*[423]

[423] DOI: 10.1109/MS.2004.1320864

"'We resolve to **keep all** program design **documentation** complete, precise, and **up to date**'. - David Lorge Parnas"

Nancy Leveson, Barry Boehm, Shari Lawrence Pfleeger, Nancy R. Mead, Elaine Weyuker, Al Davis, Watts S. Humphrey, David Lorge Parnas, Victor R. Basili, John D. Musa, ***New Year's Resolutions for Software Quality****, IEEE Software, January 2004.*[424]

[424] DOI: 10.1109/MS.2004.1259165

“When you dig into the knowledge base of experience, you find the following **seven best practices** for **outsourcing** common to reported successes in the literature … **Never** outsource a **core competency** … Establish **win-win conditions** with your suppliers … **Nurture your relationships** with your suppliers … **Measure** performance as **quantitatively** as possible … Make **exceptional performance** financially worthwhile … Treat outsourcing as a **technology transfer** opportunity …”

---

*Donald J. Reifer, **Seven Hot Outsourcing Practices**, IEEE Software, January 2004.*[425]

[425] DOI: 10.1109/MS.2004.1259166

“This month, the column looks at how to **ruin a software project** in just **three easy steps**. … (1) **Don’t bother to check** if the code is doing what you think it’s doing (2) **Never let go** of code … (3) Whether you are compiling, testing, creating a release, or doing end-user product installation, **do it differently every time**. … Fortunately, three simple practices can **save a project** from these and other common mishaps: **version control**, **unit testing**, and **automation**.”

Dave Thomas, Andy Hunt, ***Three Legs, No Wobble****, IEEE Software, January 2004.*[426]

[426] DOI: 10.1109/MS.2004.1259177

"The open source movement is affecting software development products and processes. … Process integration and the **coevolution of multiple open source and proprietary projects** are still open problems."

Clemens Szyperski, *Diomidis Spinellis*, ***Guest Editors' Introduction: How Is Open Source Affecting Software Development?***, *IEEE Software*, *January 2004.*[427]

[427] DOI: 10.1109/MS.2004.1259204

“**Many** software development methodologies **are called ‘open source.’** However simply stating that a project is open source **doesn’t precisely describe** the approach used to support the project.”

Budi Arief, Cristina Gacek, ***The Many Meanings of Open Source****, IEEE Software, January 2004.*[428]

[428] DOI: 10.1109/MS.2004.1259206

“Mission operators at **NASA’s Jet Propulsion Laboratory** use Science Activity Planners to analyze data acquired by rovers and direct their activities. In designing the SAP for the **Mars Exploration Rovers** project, developers **relied** heavily **on open source** components. They found that using open source software components not only helped keep the project within budget but also resulted in a **more robust and flexible** tool.”

Jeffrey S. Norris, Poul-Henning Kamp, ***Mission-Critical Development with Open Source Software: Lessons Learned****, IEEE Software, January 2004.*[429]

[429] DOI: 10.1109/MS.2004.1259211

“The importance of changing the mindset in relation to the new support paradigm implied by **open-source software** (OSS) is also significant. By and large, reliance on a standard maintenance contract isn’t an option, and bulletin boards might be the main source of support. Thus, it is hardly surprising that support from top management is critical. Also, even though OSS may be available at little or no cost, organizations should not expect maintenance and support to be available at a lesser cost than would apply for commercial software. Indeed, OSS represents a good opportunity for small software companies all around the world to treat it as **an infrastructure component**, like the highway or telecommunications lines, and then use it as a bootstrap to build a service and support business model on top.”

Tony Kenny, Brian Fitzgerald, ***Developing an Information Systems Infrastructure with Open Source Software****, IEEE Software, January 2004.*[430]

[430] DOI: 10.1109/MS.2004.1259216

"Any company dealing with **OSS** needs a **few simple rules** for using it in product development: Control the introduction and use of OSS; it must be **explicitly authorized** on a per-version basis. Disseminate **technical**, **managerial**, and **legal** information widely in your company. Systematically qualify OSS components before integrating them."

Christof Ebert, Michel Ruffin, ***Using Open Source Software in Product Development: A Primer****, IEEE Software, January 2004.*[431]

[431] DOI: 10.1109/MS.2004.1259227

“**Outsourcing**, overseas development, and **foreign workers displacing American ones**: These are all interesting issues, but in a sense they’re all **so ‘last century.’** Or are they?”

---

*Robert L. Glass, **Sources for Software Development: A Mugwumpish View**, IEEE Software, January 2004.*[432]

[432] DOI: 10.1109/MS.2004.1259286


Agile Requirements: Opportunity or Oxymoron?
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
MAY/JUNE 2004
Return on Investment
MDA: Revenge or Utopia? p. 15
Correctness Tools Coming of Age p. 92
Are Gold Practices Best Practices? p. 108
www.computer.org
IEEE
COMPUTER SOCIETY

“Ubiquitous and reckless **promotion by vendors**, consultants, and marketing gurus has **diluted** the expression **return on investment** into an **umbrella term** that can mean anything from profits to competitive advantage to simply ‘something good.’ Consequently, the software community looks upon ROI with **increasing suspicion** as a vague and slippery gimmick used chiefly to make the **sales pitch** (invariably unsubstantiated) for a particular product or initiative.”

John Favaro, Hakan Erdogmus, Wolfgang Strigel, ***Guest Editors’ Introduction: Return on Investment****, IEEE Software, May 2004.*[433]

[433] DOI: 10.1109/MS.2004.1293068

"You may have to **reorganize** so that your organizational structure is set up to best **produce the commonality** and to best take advantage of it. Your people will need some **training**. You'll need to set up **new processes** to make this all work, and those processes will **evolve** as you get better at them. You'll want to **collect data** so that you can see if this new approach is meeting your goal."

Paul Clements, Klaus Schmid, G? B?ckle, Dirk Muthig, John D. McGregor, ***Calculating ROI for Software Product Lines****, IEEE Software, May 2004.*[434]

[434] DOI: 10.1109/MS.2004.1293069

“Calculating **cost and benefits** is a prerequisite for **investment decision** making. This is just as true for SPI as for any other investment. “

Rini van Solingen, ***Measuring the ROI of Software Process Improvement***, *IEEE Software*, *May 2004.*[435]

[435] DOI: 10.1109/MS.2004.1293070

"The last few years have seen intense scrutiny of the **flawed business premises** underlying the **dot-com bubble** of the late 1990s. The prevailing attitude then was that software investment could be repaid through the company's increased capital value in expectation of **future profits**. The current IT environment is greatly changed. Not only are organizations no longer willing to invest in software development without clear expectations for returns, but they also demand those **returns in much less time**."

Jane Cleland-Huang, Mark Denne, ***The Incremental Funding Method: Data-Driven Software Development****, IEEE Software, May 2004.*[436]

[436] DOI: 10.1109/MS.2004.1293071

“For commercial software sold in the **general marketplace**, many software-planning and design decisions are based not only on meeting user needs but also on various **marketplace issues**. Many of these issues fall into one of three standard ROI categories: **revenue**, **cost**, and **risk**. “

---

*David G. Messerschmitt, Clemens Szyperski, **Marketplace Issues in Software Planning and Design**, IEEE Software, May 2004.*[437]

[437] DOI: 10.1109/MS.2004.1293074

"The trait that I value in effective quality assurance analysts is the ability to question what others often too readily accept. This admirable characteristic manifests itself in three ways: **know what you know**, **ask when you don't**, and **ask when you do**."

---

*Jane Huffman Hayes, **On the Virtues of Not Knowing**, IEEE Software, May 2004.*[438]

[438] DOI: 10.1109/MS.2004.1293076

"Keep it **DRY**, keep it **shy**, and **tell** the other guy."

Andy Hunt, Dave Thomas, ***OO in One Sentence: Keep It DRY, Shy, and Tell the Other Guy****, IEEE Software, May 2004.*[439]

[439] DOI: 10.1109/MS.2004.1293081

"**Software maintenance** and software maintainers **deserve more respect**."

Robert L. Glass, ***Learning to Distinguish a Solution from a Problem***, *IEEE Software*, *May 2004.*[440]

[440] DOI: 10.1109/MS.2004.1293084

Also: New Open Source Department

IEEE Software

BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

JULY/AUGUST 2004

Successful Process Change

www.computer.org

IEEE COMPUTER SOCIETY

"Although using professional programmers doesn't guarantee **correctness**, **security**, or **maintainability**, the **lack of real understanding** about software development by **end-user programmers** poses a **danger** to stakeholders associated with mission-critical systems from the standpoints of both correctness and security."

Warren Harrison, ***From the Editor: The Dangers of End-User Programming****, IEEE Software, July 2004.*[441]

[441] DOI: 10.1109/MS.2004.13

"Evaluating process maturity on the basis of results, or the improvement in cost structure in the four quality-cost categories: **prevention**, **appraisal**, **internal failures**, and **external failures**."

Nancy Eickelmann, ***Measuring Maturity Goes beyond Process****, IEEE Software, July 2004.*[442]

[442] DOI: 10.1109/MS.2004.21

"The activity of '**design**' includes many things, but certainly one of the most important aspects is **interface specification**. … Make interfaces **easy** to use **correctly** and **hard** to use **incorrectly**."

---

*Scott Meyers, **The Most Important Design Guideline?**, IEEE Software, July 2004.*[443]

[443] DOI: 10.1109/MS.2004.29

“From **Chile** to **Sweden** to **Georgia** to **Hong Kong**, for very small teams to large organizations, for basic repeatability to complex technology, the question is the same: Why isn’t process change easier? … each part of an organization’s staff must **share the values** of process change to succeed. Company leadership must have **a vision** of the benefit. Development teams must **see the value**. Process engineers must recognize that the change will be both **interactive** and **iterative**.”

Annie Combelles, David Dorenbos, ***Introduction: Lessons Learned around the World: Key Success Factors to Enable Process Change****, IEEE Software, July 2004.*[444]

[444] DOI: 10.1109/MS.2004.19

“Traditional approaches to measuring software process improvement are typically lengthy, data intensive, and cost prohibitive. A **simple indicator**, the extent to which **engineering practices change**, can provide enough information to guide initiatives toward success.”

Lars Mathiassen, Anna B?rjesson, ***Successful Process Implementation****, IEEE Software, July 2004.*[445]

[445] DOI: 10.1109/MS.2004.27

“**Failures**, **faults**, and **errors** are often collectively referred to as **defects**, and defect handling deals with **recording**, **tracking**, and **resolving** these defects. “

Günes Koru, Jeff Tian, ***Defect Handling in Medium and Large Open Source Projects****, IEEE Software, July 2004.*[446]

[446] DOI: 10.1109/MS.2004.12

"**Incremental change** activities include: change request, concept extraction, concept location, impact analysis, actualization, incorporation, change propagation, refactoring, and role splitting ..."

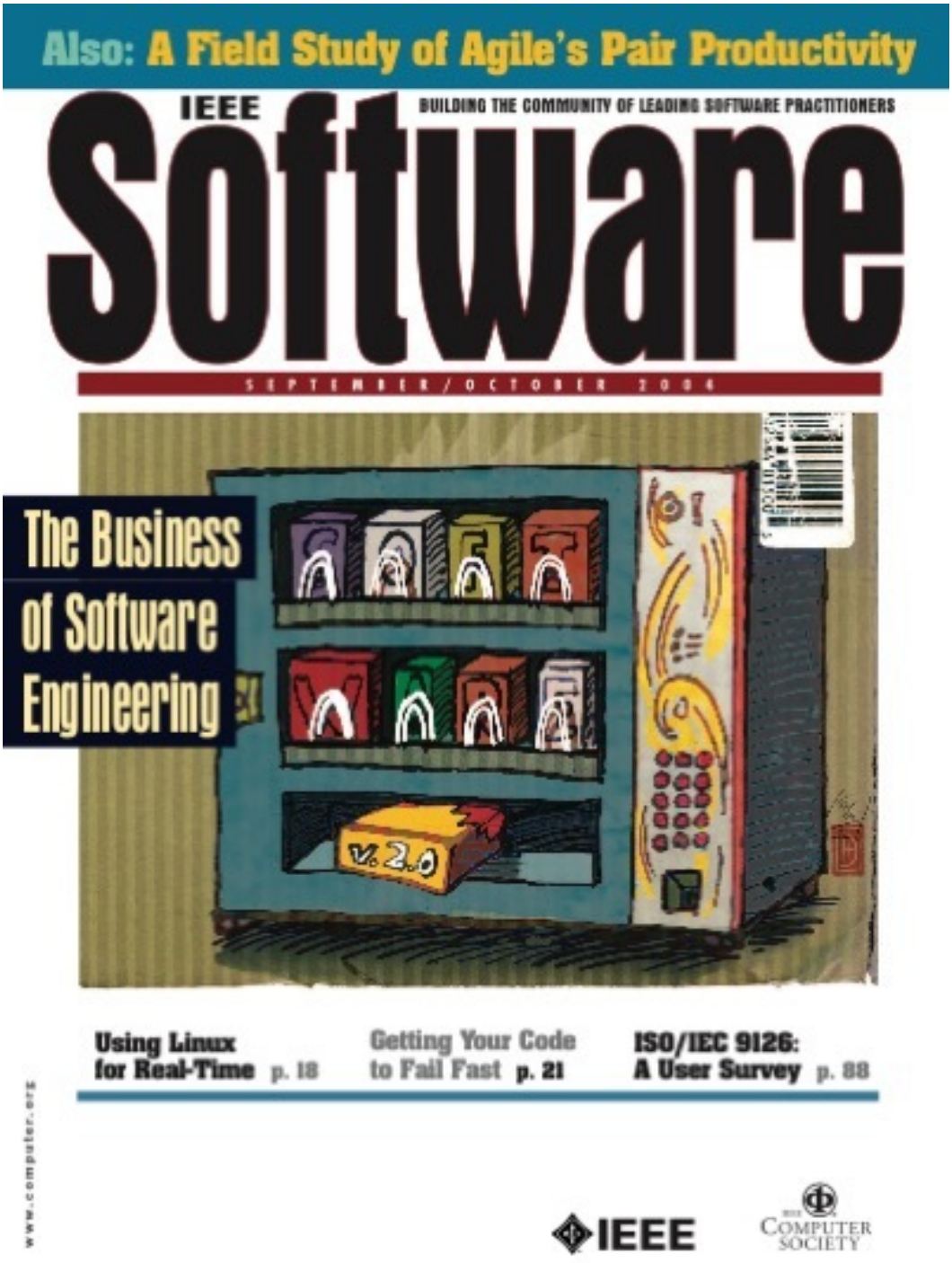
Prashant Gosavi, V?clav Rajlich, ***Incremental Change in Object-Oriented Programming****, IEEE Software, July 2004.*[447]

[447] DOI: 10.1109/MS.2004.17

“We software engineers have engaged in a **rush to standardization**, a rush to getting everyone to use the same set of facts and principles.”

Robert L. Glass, ***Some Heresy Regarding Software Engineering***, *IEEE Software, July 2004.*[448]

[448] DOI: 10.1109/MS.2004.26


Also: A Field Study of Agile's Pair Productivity
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
SEPTEMBER/OCTOBER 2004
The Business of Software Engineering
v. 2.0
Using Linux for Real-Time p. 18
Getting Your Code to Fail Fast p. 21
ISO/IEC 9126: A User Survey p. 88
www.computer.org
IEEE
COMPUTER SOCIETY

“There’s a **simple technique** that will dramatically reduce the number of **these bugs** in your software. … The technique is to build your software to ‘**fail fast**.’”

Jim Shore, ***Fail Fast***, *IEEE Software*, *September 2004*.[449]

[449] DOI: 10.1109/MS.2004.1331296

"**Core business principles** refer to those practices that a company, institution, or government agency uses to enable the **organization's viability**. Whether Web services, supply chain, payroll, timesheets, or other application, there must be a **relationship** between your organization's **core business principles** and how **its software's functionality** is defined, developed, deployed, tested, and maintained."

---

*Jeffrey Voas, **Software Engineering's Role in Business**, IEEE Software, September 2004.*[450]

[450] DOI: 10.1109/MS.2004.1331297

“**Poor performers** cause much more **damage** than is apparent to management. “

Shahrukh A. Irani, Ho Woo Lee, Peter Middleton, ***Why Culling Software Colleagues Is Popular****, IEEE Software, September 2004.*[451]

[451] DOI: 10.1109/MS.2004.1331298

"**Software systems** are the only major organizational asset with no real support for managing them based on information technology. … **no one keeps** any **basic information** concerning a vastly expensive corporate asset."

Garry S. Marliss, Mordechai Ben-Menachem, ***Inventorying Information Technology Systems: Supporting the 'Paradigm of Change'****, IEEE Software, September 2004.*[452]

[452] DOI: 10.1109/MS.2004.1331300

“**Network effects** can lead to a ‘**social dilemma**,’ in which the actions of consumers can result in serious **negative consequences** for the **same consumers** and the society as a whole in the long term. “

Nirup M. Menon, Glenn J. Browne, ***Network Effects and Social Dilemmas in Technology Industries****, IEEE Software, September 2004.*[453]

[453] DOI: 10.1109/MS.2004.1331301

"The well-known **black-box model** of software **development outsourcing** is typically effective. The approach assumes that the vendor can successfully solve a client organization's business problem without either organization having to deeply understand the other's domain. … key finding is that the black-box approach usually works **well in routine projects** but **fails** in projects involving **novelty**."

Amrit Tiwana, ***Beyond the Black Box: Knowledge Overlaps in Software Outsourcing****, IEEE Software, September 2004.*[454]

[454] DOI: 10.1109/MS.2004.1331302

"The international standard **ISO/IEC 9126** defines a **quality model** for **software products**. The model categorizes software product attributes into six characteristics, which are further subdivided into 27 subcharacteristics."

Ho-Won Jung, Seung-Gweon Kim, Chang-Shin Chung, ***Measuring Software Product Quality: A Survey of ISO/IEC 9126****, IEEE Software, September 2004.*[455]

[455] DOI: 10.1109/MS.2004.1331309

“As an industry, we love to build the **grand frameworks** that can solve **all the world’s problems** in one unified package. “

---

*Andy Hunt, Dave Thomas, **Imaginate**, IEEE Software, September 2004.*[456]

[456] DOI: 10.1109/MS.2004.1331311

“Phillips suggests that ‘when we are **behind schedule** and **under pressure**, we **stop breathing**.’ And that, in turn, leads to some pretty **bad outcomes** …”

---

*Robert L. Glass, **Anarchy and the Effects of Schedule Pressure**, IEEE Software, September 2004.*[457]

[457] DOI: 10.1109/MS.2004.1331316


XP for Mission-Critical Products
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
NOVEMBER/DECEMBER 2004
Persistent Software Attributes
Protecting Our Software Infrastructure p. 59
Building Apps with Apache's Ant Tool p. 89
Is the Industry's Productivity Declining? p. 92
www.computer.org
IEEE
COMPUTER SOCIETY

“**Software changes** are a lot **like rain**. While a **few drops** here and there usually **aren’t** that much of a **problem**, a **steady downpour** can be **damaging**—and a deluge can wipe out everything you’ve carefully built. “

---

*Maarten Boasson, Terry Bollinger, Jeffrey Voas, **Persistent Software Attributes**, IEEE Software, November 2004.*[458]

[458] DOI: 10.1109/MS.2004.46

"In contemporary societies, **individuals** and **organizations** increasingly **depend on** services delivered by sophisticated **software**-intensive systems. **Dependability** has become a key **systems property**, which needs to be engineered and guaranteed regardless of continuous, rapid, and unpredictable technological and context changes."

Paolo Donzelli, Victor Basili, Sima Asgari, ***A Unified Model of Dependability: Capturing Dependability in Context****, IEEE Software, November 2004.*[459]

[459] DOI: 10.1109/MS.2004.30

“If you don’t carefully and accurately **manage customizations** … the result can be **skyrocketing development costs**, poor customer support, and an inability to respond quickly to new needs.”

Sonia Calzada, Ismael Ciordia, Fernando Alonso, Nicol? Serrano, ***Automated Management of Multicustomer Code Bases****, IEEE Software, November 2004.*[460]

[460] DOI: 10.1109/MS.2004.35

"If **developers** are willing to **work hard** and carefully **introduce an agile process** into their organization, their efforts should yield **positive results**."

David Noftz, Rekha Raghu, Jerry Drobka, ***Piloting XP on Four Mission-Critical Projects****, IEEE Software, November 2004.*[461]

[461] DOI: 10.1109/MS.2004.47

"**Routine tasks** are the easiest ones for programmers to **avoid**. So, to develop a single program—even a program with only one file — it's common to have a script that compiles it with the desired options and executes it with some arguments. When your project has hundreds or thousands of files and a large team of developers, the script isn't a utility, but a necessity; it becomes even more critical when the project's structures are complicated and the dependencies are hard to remember."

Ismael Ciordia, Nicol? Serrano, ***Ant: Automating the Process of Building Applications****, IEEE Software, November 2004.*[462]

[462] DOI: 10.1109/MS.2004.33

"Many software projects involve **at least one stakeholder who secretly wants the project to fail**. Finding this stakeholder can be difficult but is important to ensuring the project's success."

Johann Rost, ***Political Reasons for Failed Software Projects***, *IEEE Software, November 2004.*[463]

[463] DOI: 10.1109/MS.2004.48

2005


Evidence-Based Software Engineering
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
JANUARY/FEBRUARY 2005
Innovation in Requirements Engineering
Peer-Based In-Service Training p. 5
A New Column: Tools of the Trade p. 10
Reviewing Source Code Review Systems p. 74
www.computer.org
IEEE
COMPUTER SOCIETY

"In many domains, the **problem space** is too **large** to **explore** up front. **Expert designers** often **explore** the **problem and solution** spaces **in parallel**, using the emerging solution space to decide what information to elicit next about the problem space. They also often look for chinks—that is, **omissions and inconsistencies**—in problems and requirement specifications that would enable them to discover more innovative solutions."

Neil Maiden, Christof Ebert, Suzanne Robertson, ***Guest Editors' Introduction: Shake, Rattle, and Requirements****, IEEE Software, January 2005.*[464]

[464] DOI: 10.1109/MS.2005.8

"Some **activities** are fundamental to all **RE** requirements engineering processes: **Elicitation**. Identify sources of information … **Analysis**. Understand the requirements … **Validation**. Go back to the system stakeholders and check … **Negotiation**. … reconcile conflicting views …. **Documentation**. Write down the requirements … **Management**. Control the requirements changes … "

Ian Sommerville, ***Integrated Requirements Engineering: A Tutorial***, *IEEE Software, January 2005.*[465]

[465] DOI: 10.1109/MS.2005.13

“**Patterns** are an established and well-known **format** for **capturing engineering knowledge**. … practitioners use patterns to describe **reference solutions** to engineering problems and as guidelines for engineering procedures”

---

*Lars Hagge, Kathrin Lappe, **Sharing Requirements Engineering Experience Using Patterns**, IEEE Software, January 2005.*[466]

[466] DOI: 10.1109/MS.2005.17

“The development of large, complex software products aimed for a broad **market** involves a continuous, massive **inflow** of **customers’ wishes** (collected from the **market**) and product **requirements** (generated **inside** the developing **organization**). “

---

*Johan Natt och Dag, Sjaak Brinkkemper, Bj? Regnell, Vincenzo Gervasi, **A Linguistic-Engineering Approach to Large-Scale Requirements Management**, IEEE Software, January 2005.*[467]

[467] DOI: 10.1109/MS.2005.1

"Producing good systems relies on asking '**the right questions**' to discover users' real requirements. **Family therapy** provides **interview techniques** and different types of questions that can **generate new knowledge**."

---

*Susanne Kandrup, **On Systems Coaching**, IEEE Software, January 2005.*[468]

[468] DOI: 10.1109/MS.2005.15

"The word '**all**' and the plural can be **misused** in ways that create **ambiguities** and other problems in computer-based system specification documents."

Erik Kamsties, Daniel M. Berry, ***The Syntactically Dangerous All and Plural in Specifications****, IEEE Software, January 2005.*[469]

[469] DOI: 10.1109/MS.2005.22

“We like to think of **science** with certainty, where results are clear when rules are understood and followed. But **science** is **rife with uncertainty**, and we must acknowledge its role and the **resulting risks** we take, both when we **generate evidence** and when we use it to **build arguments**. **Lawyers** recognize the uncertainty associated with various types of evidence, so they look for pieces of evidence that in concert have more ‘**evidential force**’ than when used separately. “

---

*Jason Remillard, **Source Code Review Systems**, IEEE Software, January 2005.*[470]

[470] DOI: 10.1109/MS.2005.20

"So there you have it. I'm **mad as hell**, I don't want to take this anymore, and I suppose the truth of the matter is that I can't do anything about it!"

---

*Robert L. Glass, **Viruses Are Beginning to Get to Me!**, IEEE Software, January 2005.*[471]

[471] DOI: 10.1109/MS.2005.24


Hypermedia Systems Development Practices
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
MARCH/APRIL 2005
Postmodern Software Design
What's a Good Bug Tracker? p. 11
The Two-Phase Commit p. 64
"Best" Practices in Practice p. 96
www.computer.org
IEEE
COMPUTER SOCIETY

"**Bug-tracking systems** help us **identify** the error in our software, **resolve** it, and **learn** from it."

Ismael Ciordia, Nicol? Serrano, ***Bugzilla, ITracker, and Other Bug Trackers****, IEEE Software, March 2005.*[472]

[472] DOI: 10.1109/MS.2005.32

“Any **bold advance** needs some **time to mature**—to ‘cross the chasm,’ as Geoffrey Moore eloquently described, to get a critical mass of practitioners across our industry beyond the eager **early-adopter** stage. Techniques, practices, and methods must be **taught in schools** and must be supported by **tools**. They must **prove their value** beyond any reasonable doubt and sometimes even be enshrined in some industry **standard**.”

Philippe Kruchten, ***Editor’s Introduction: Software Design in a Postmodern Era****, IEEE Software, March 2005.*[473]

[473] DOI: 10.1109/MS.2005.38

"Explicitly **documenting** major architecture **decisions** makes the architecture development process more **structured** and **transparent**. Additionally, it clarifies the architects' **rationale** for stakeholders, designers, and other architects."

Art Akerman, Jeff Tyree, ***Architecture Decisions: Demystifying Architecture****, IEEE Software, March 2005.*[474]

[474] DOI: 10.1109/MS.2005.27

"Elements such as **principles**, **heuristics**, **best practices**, '**bad smells**,' and **refactorings** are **not clearly defined**. Many of these elements are synonymous, and others are just vague concepts."

Mario Piattini, Javier Garz?, ***An Ontology for Microarchitectural Design Knowledge****, IEEE Software, March 2005.*[475]

[475] DOI: 10.1109/MS.2005.26

"**Architecture reviews** … **identify project** problems **before** they become **costly to fix** and provide timely information to upper management so that they can make **better-informed decisions**."

David M. Weiss, Sandra A. Rozsypal, Joseph F. Maranzano, Gus H. Zimmerman, Guy W. Warnken, Patricia E. Wirth, ***Architecture Reviews: Practice and Experience****, IEEE Software, March 2005.*[476]

[476] DOI: 10.1109/MS.2005.28

“One of the key issues in **Model Driven Architecture** is **model mapping**–that is, the **transformation** of models from one formalism to another.”

Jean-Louis Sourrouille, Guy Caplat, ***Model Mapping Using Formalism Extensions****, IEEE Software, March 2005.*[477]

[477] DOI: 10.1109/MS.2005.45

"**Software engineers** use certain common terms, such as **design**, analysis, and documentation, in significantly **different ways** from other engineers. … '**design**' in **software engineering** is **more limited** in scope than in other fields."

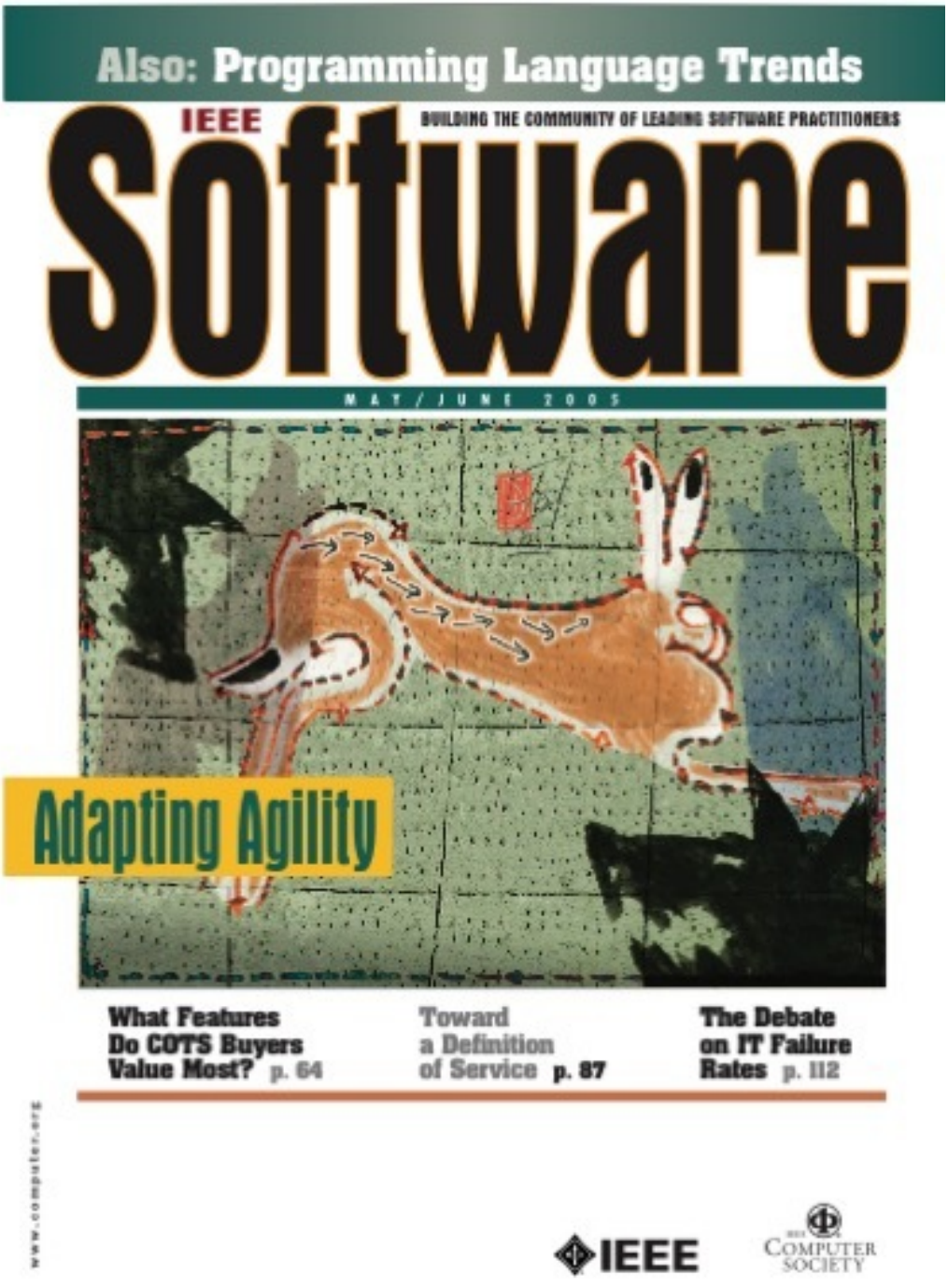
Philippe Kruchten, ***Casting Software Design in the Function-Behavior-Structure Framework****, IEEE Software, March 2005.*[478]

[478] DOI: 10.1109/MS.2005.33

“You **know** you’re a **geek** when **going to the coffee shop** gets you thinking about interaction patterns between loosely coupled systems. … Interestingly, the **optimization for throughput** results in a **concurrent and asynchronous processing model**: when you place your order, the cashier marks a coffee cup with your order and places it into a queue.
“

Gregor Hohpe, ***Your Coffee Shop Doesn’t Use Two-Phase Commit***, *IEEE Software, March 2005.*[479]

[479] DOI: 10.1109/MS.2005.52


Also: Programming Language Trends
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
MAY/JUNE 2005
Adapting Agility
What Features Do COTS Buyers Value Most? p. 64
Toward a Definition of Service p. 87
The Debate on IT Failure Rates p. 112
www.computer.org
IEEE
COMPUTER SOCIETY

“**Agile** programming is **design for change**, without refactoring and rebuilding.”

---

*Dave Thomas, **Agile Programming: Design to Accommodate Change**, IEEE Software, May 2005.*[480]

[480] DOI: 10.1109/MS.2005.54

“**Kent Beck** wrote **eXtreme Programming eXplained** … but the book has also caused an extraordinary degree of vitriol. … The reasons appear to include a **focus on** writing **programs** rather than analysis or design (also known as modeling); a **disdain for documentation** as such; and the **Communist** notion of working only **40 hours** a week.”

Stephen J. Mellor, ***Editor’s Introduction: Adapting Agile Approaches to Your Project Needs****, IEEE Software, May 2005.*[481]

[481] DOI: 10.1109/MS.2005.61

“People have claimed that **plan-based** and **agile** companies use very different project **management** techniques. … managers using **agile** methods **focus on people** and **process** more than other managers. … adopting **agile methods** appears to offer a good solution for **improving** the **management** of the development **process** and **customer relationships**.”

Stefano De Panfilis, Alberto Sillitti, Martina Ceschi, Giancarlo Succi, ***Project Management in Plan-Based and Agile Companies****, IEEE Software, May 2005.*[482]

[482] DOI: 10.1109/MS.2005.75

“We categorize projects into **dogs** (simple projects with low uncertainty), **colts** (simple projects with high uncertainty), **cows** (complex projects with low uncertainty), or **bulls** (complex projects with high uncertainty). We adapt our agile process by adding practices according to a project’s profile.”

Todd Little, ***Context-Adaptive Agility: Managing Complexity and Uncertainty****, IEEE Software, May 2005.*[483]

[483] DOI: 10.1109/MS.2005.60

"Requirements engineers must … view **requirements** as a **sociotechnical discipline** and draw skills, techniques, and knowledge from other disciplines."

---

*Suzanne Robertson, **Learning from Other Disciplines**, IEEE Software, May 2005.*[484]

[484] DOI: 10.1109/MS.2005.68

“One of the **most robust findings in forecasting**, human judgment, and software estimation studies is that ‘**combination works**.’ Apparently it doesn’t matter whether the combination involves a **simple average** of estimates from different methods or a sophisticated weighting algorithm. A **simple average** offers a robust combination method unless one estimation method or expert is obviously more reliable than another. … An expert’s **technical skill** level can be a **poor indicator** of accuracy, and it’s rarely obvious, in advance, which expert will be the better estimator. This is one reason a **simple average** of outputs from different estimation experts and methods frequently offers the **most robust** and accurate combination method.”

Magne Jørgensen, ***Practical Guidelines for Expert-Judgment-Based Software Effort Estimation***, *IEEE Software, May 2005.*[485]

[485] DOI: 10.1109/MS.2005.73

"**Text mining** is a relatively new research area associated with the creation of novel information resources from electronic text repositories. An **expert-witness database** based on text from legal, medical, and news documents demonstrates the successful application of text-mining techniques."

Christopher Dozier, Peter Jackson, ***Mining Text for Expert Witnesses****, IEEE Software, May 2005.*[486]

[486] DOI: 10.1109/MS.2005.70

“It’s **difficult to have a problem** with anything **Martin Fowler** writes. He’s obviously a skilled designer, he practices what he preaches, and he has a forceful command of the English language without being stuffy. “

Stephen Mellor, Christof Ebert, Fernando Berzal, ***UML Distilled: From Difficulties to Assets****, IEEE Software, May 2005.*[487]

[487] DOI: 10.1109/MS.2005.81

"I want to question the unquestionable status of that **Standish report**. That's because, you see, my own observations lead me to believe that **something is terribly wrong** with those Standish findings."

---

*Robert L. Glass, **IT Failure Rates - 70% or 10-15%?**, IEEE Software, May 2005.*[488]

[488] DOI: 10.1109/MS.2005.66


China's Shifting Software Industry
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
JULY/AUGUST 2005
COTS Integration
The Saboteur Within p. 5
Tool Writing: A Forgotten Art? p. 9
Pattern-Based Testing p. 68
www.computer.org
IEEE
COMPUTER SOCIETY

“Writing **stand-alone** tools that you can **combine** efficiently with others to handle more demanding tasks appears to be becoming a **forgotten art**.”

Diomidis Spinellis, ***Tool Writing: A Forgotten Art?***, *IEEE Software*, *July 2005.*[489]

[489] DOI: 10.1109/MS.2005.111

“**JUnit** is an open source Java library that purports to make **unit testing** so **much fun** that programmers will actually want to write tests for their code.”

Panagiotis Louridas, ***JUnit: Unit Testing and Coding in Tandem***, *IEEE Software*, *July 2005.*[490]

[490] DOI: 10.1109/MS.2005.100

"One strategy that **originally** seemed **promising** was the notion of '**buy not build**.' Using **COTS** products is one way to implement this strategy, because software development then becomes the process of '**simply**' **integrating** COTS components. However, it turns out that dealing with COTS is a **high-risk** activity …"

Alexander Egyed, Dewayne E. Perry, Hausi A. Muller, ***Guest Editors' Introduction: Integrating COTS into the Development Process****, IEEE Software, July 2005.*[491]

[491] DOI: 10.1109/MS.2005.93

"The **availability** of the components **before the system** is built provides **early data** on their performance properties."

Murray Woodside, Xiuping Wu, Erik Putrycz, ***Performance Techniques for COTS Systems****, IEEE Software, July 2005.*[492]

[492] DOI: 10.1109/MS.2005.102

"What's the **state of the practice** of software engineering? If you look at current software engineering **books**, **journals**, and **conferences**, you **won't find much** of an answer."

---

*Robert L. Glass, **A Sad SAC Story about the State of the Practice**, IEEE Software, July 2005.*[493]

[493] DOI: 10.1109/MS.2005.82


Following the Research Money
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
SEPTEMBER/OCTOBER 2005
Modeler
Manager
Architect
Tester
Project Management
How to Do Software Experiments p. 8
Rich-Media Scenarios p. 89
Learning from Software Failure p. 115
www.computer.org
IEEE
COMPUTER SOCIETY

"Twenty years is a very long time in the computing field. Yet, SEPM (Software Engineering Projects Management)'s progress has been agonizingly slow in many ways, probably because it's driven more by human behavior than by technology. **People change their behavior much more slowly than technology advances**."

Arthur B. Pyster, Richard H. Thayer, ***Guest Editors' Introduction: Software Engineering Project Management 20 Years Later****, IEEE Software, September 2005.*[494]

[494] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2005.137

“Because we must balance the enterprise architect’s goals with the needs of the agile development organization and the users driving the application’s development, it helps to have **enterprise architects join the development team**. This provides opportunities to address the needs of all stakeholders: architects, developers, and business users.”

---

*Rebecca J. Parsons, **Enterprise Architects Join the Team**, IEEE Software, September 2005.*[495]

[495] DOI: 10.1109/MS.2005.119

“**Twenty years** is a very long time in the computing field. Yet, **SEPM’s** software engineering project management progress has been **agonizingly slow** in many ways, probably because it’s driven more by **human behavior** than by technology. People change their behavior much more slowly than technology advances. “

Arthur B. Pyster, Richard H. Thayer, ***Guest Editors’ Introduction: Software Engineering Project Management 20 Years Later****, IEEE Software, September 2005.*[496]

[496] DOI: 10.1109/MS.2005.137

“Our discussions with **traditional developers and managers** concerning **agile software development** practices nearly always contain two somewhat contradictory ideas. They find that on small, stand-alone projects, agile practices are less burdensome and more in tune with the software industry’s increasing needs for rapid development and coping with continuous change. However, they’re frustrated with **the difficulty of scaling up** and integrating them into traditional, top-down systems development organizations.”

Richard Turner, Barry Boehm, ***Management Challenges to Implementing Agile Processes in Traditional Development Organizations****, IEEE Software, September 2005.*[497]

[497] DOI: 10.1109/MS.2005.129

“Use a **steering leadership** style rather than the detailed **plan-and-track** leadership style encouraged by conventional wisdom.”

Walker Royce, ***Successful Software Management Style: Steering and Balance***, *IEEE Software, September 2005.*[498]

[498] DOI: 10.1109/MS.2005.138

“An enormous **intellectual distance** exists between the fields of **computer science** and **information systems**, which needs to be fixed soon.”

Robert L. Glass, ***Never the CS and IS Twain Shall Meet?***, *IEEE Software, September 2005.*[499]

[499] DOI: 10.1109/MS.2005.130


Economics of Learning and Flexibility
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
NOVEMBER/DECEMBER 2005
Predictor Models
The Art & Science of Release Planning p. 47
Opportunistic Problem Solving p. 60
Improving Small Organizations: A Case Study p. 68
www.computer.org
IEEE
COMPUTER SOCIETY

"Software engineering is a **decision-intensive discipline**. It still struggles with basic questions regarding the **utility of models**. Can researchers help software practitioners by building models that make explicit the knowledge hidden in various software resources?"

Bojan Cukic, ***Guest Editor's Introduction: The Promise of Public Software Engineering Data Repositories***, *IEEE Software, November 2005.*[500]

[500] DOI: 10.1109/MS.2005.153

"The '**art of release planning**' refers to relying on human intuition, communication, and capabilities to negotiate between conflicting objectives and constraints. The '**science of release planning**' refers to formalizing the problem and applying computational algorithms to generate best solutions."

G?nther Ruhe, Moshood Omolade Saliu, ***The Art and Science of Software Release Planning****, IEEE Software, November 2005.*[501]

[501] DOI: 10.1109/MS.2005.164

"The field of software engineering would greatly benefit from detailed research on **why some software builders perform better than others**, but this isn't happening."

---

*Robert L. Glass, **A Follow-the-Leader Story with a Strange Ending**, IEEE Software, November 2005.*[502]

[502] DOI: 10.1109/MS.2005.144

2006


Victor Basili and Software Quality
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
JANUARY/FEBRUARY 2006
ASPECT ORIENTED COMPUTING
Portable Project Assets p. 100
In Search of the System Concept p. 102
Integrity and Software Engineering p. 120
www.computer.org
IEEE
IEEE computer society
60th anniversary

“Although design is a highly creative activity, we can still **learn fundamental design skills** - and accomplish a lot with them.”

Rebecca J. Wirfs-Brock, ***Looking for Powerful Abstractions***, *IEEE Software, January 2006.*[503]

[503] DOI: 10.1109/MS.2006.22

"**Basili's** contributions cover three broad areas: research in the 1970s and early 1980s on **software measurement** and the **Goal Question Metric (GQM) model**, research in the 1980s and 1990s on these measurement ideas' maturation into a software engineering model of empirical studies, including the development of the **Quality Improvement Paradigm (QIP)** and the influence of the NASA Goddard Space Flight Center Software Engineering Laboratory, and research since 1990 in the Experience Factory as a model for creating learning organizations for continuous software process improvement."

Marvin Zelkowitz, Carolyn Seaman, Forrest Shull, ***Victor R. Basili's Contributions to Software Quality****, IEEE Software, January 2006.*[504]

[504] DOI: 10.1109/MS.2006.33

"**Aspect-oriented programming** technologies aim to improve system modularity by **modularizing crosscutting concerns**. Global properties and programming and design issues can lead to crosscutting concerns–for example, error handling or transaction code, interacting features, and reliability and security. "

Christa Schwanninger, Gail Murphy, ***Guest Editors' Introduction: Aspect-Oriented Programming****, IEEE Software, January 2006.*[505]

[505] DOI: 10.1109/MS.2006.12

"**Service-oriented architectures** are designed to support loose coupling between interacting software applications. Using Web services technology, SOAs support the creation of distributed applications in a heterogeneous environment."

Viviane Jonckers, Bart Verheecke, Wim Vanderperren, ***Unraveling Crosscutting Concerns in Web Services Middleware****, IEEE Software, January 2006.*[506]

[506] DOI: 10.1109/MS.2006.31

“Aspect-oriented programming languages such as AspectJ offer new mechanisms for decomposing systems into modules and composing modules into systems. Common ways of using these mechanisms **couple aspects** to complex, changeable **implementation details**, which can **compromise modularity**. “

Macneil Shonle, Yuanfang Cai, Hridesh Rajan, Nishit Tewari, William G. Griswold, Yuanyuan Song, Kevin Sullivan, ***Modular Software Design with Crosscutting Interfaces****, IEEE Software, January 2006.*[507]

[507] DOI: 10.1109/MS.2006.24

"**Aspects** are evident **earlier in the life cycle**, such as during requirements gathering and architecture development. Identifying these early aspects ensures that you can appropriately capture **aspects related to the problem domain** (as opposed to merely the implementation)."

Awais Rashid, Joao Araújo, Paul C. Clements, Ana Moreira, Elisa Baniassad, Bedir Tekinerdogan, ***Discovering Early Aspects****, IEEE Software, January 2006.*[508]

[508] DOI: 10.1109/MS.2006.8

"There's a **subtle way to lose your integrity**: trying to do a task as someone else would have you do it, rather than as you believe it should be done."

---

*Robert L. Glass, **Of Health, Trust, Money ... and Integrity**, IEEE Software, January 2006.*[509]

[509] DOI: 10.1109/MS.2006.25

Grady Booch's New Architecture Column

IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

Software

MARCH/APRIL 2006

Software Architecture

www.computer.org

IEEE
IEEE computer society 60th anniversary

"The **service-oriented paradigm** is founded on an assumption of **well-specified** and well-understood **contracts** that **isn't realized in practice**."

Brian A. Malloy, Jeffrey M. Voas, Jason O. Hallstrom, Nicholas A. Kraft, ***Improving the Predictable Assembly of Service-Oriented Architectures****, IEEE Software, March 2006.*[510]

[510] DOI: 10.1109/MS.2006.49

“By studying … architectural **patterns** and thus exposing … systems’ **inner beauty**, I hope to **inspire developers** who want to build on the experience of other well-engineered systems.”

Grady Booch, ***On Architecture***, *IEEE Software*, *March 2006.*[511]

[511] DOI: 10.1109/MS.2006.52

“Project teams can **improve requirements** just by making several easy steps, including defining **mission and scope**; identifying **stakeholders**, goals, and goal **conflicts**; describing **scenarios**, requirements, **justifications**, and **assumptions**; and agreeing on **priorities** and acceptance **criteria**.”

---

*Ian Alexander, **10 Small Steps to Better Requirements**, IEEE Software, March 2006.*[512]

[512] DOI: 10.1109/MS.2006.34

"It's been 10 years since **David Garlan** and **Mary Shaw** wrote their seminal book **Software Architecture Perspective on an Emerging Discipline**, since **Maarten Boasson** edited a special issue of **IEEE Software on software architecture**, and since the first International **Software Architecture Workshop** took place. What has happened over these 10 years? What have we learned? Where do we look for information? What's the community around this discipline? And where are we going from here?"

Philippe Kruchten, Henk Obbink, Judith Stafford, ***The Past, Present, and Future for Software Architecture****, IEEE Software, March 2006.*[513]

[513] DOI: 10.1109/MS.2006.59

“In the near future, **software architecture** will attain the status of all truly successful technologies: It will be **taken for granted**.”

---

*Paul Clements, Mary Shaw, **The Golden Age of Software Architecture**, IEEE Software, March 2006.*[514]

[514] DOI: 10.1109/MS.2006.58

“**UML** has been around since 1997. … UML is **used** rather **loosely** and that UML models are often **incomplete**. This leads to miscommunication and other implementation and maintenance problems.”

Michel Chaudron, Johan Muskens, Christian Lange, ***In Practice: UML Software Architecture and Design Description****, IEEE Software, March 2006.*[515]

[515] DOI: 10.1109/MS.2006.50

"Including **architecture-centric design** and analysis methods in the **Extreme Programming framework** can help software developers address quality attributes in an explicit, methodical, engineering-principled way."

Robert L. Nord, James E. Tomayko, ***Software Architecture-Centric Methods and Agile Development****, IEEE Software, March 2006.*[516]

[516] DOI: 10.1109/MS.2006.54

"**Service-Oriented Architecture (SOA)** represents a paradigm consisting of a set of **architectural principles** for **building loosely coupled software systems**. Actually, the SOA paradigm applies not only to XML Web services but also to **other technologies** such as email clients and servers and message-oriented middleware."

Michael Stal, ***Using Architectural Patterns and Blueprints for Service-Oriented Architecture****, IEEE Software, March 2006.*[517]

[517] DOI: 10.1109/MS.2006.60

"**Wikis** have become one of the most popular tool shells. You can find them just about everywhere that demands effective **collaboration** and **knowledge sharing** at a **low budget**. Wikipedia has certainly enhanced their popularity, but they also have a place in intranet-based applications such as **defect** tracking, **requirements** management, **test-case** management, and project portals."

Panagiotis Louridas, ***Using Wikis in Software Development****, IEEE Software, March 2006.*[518]

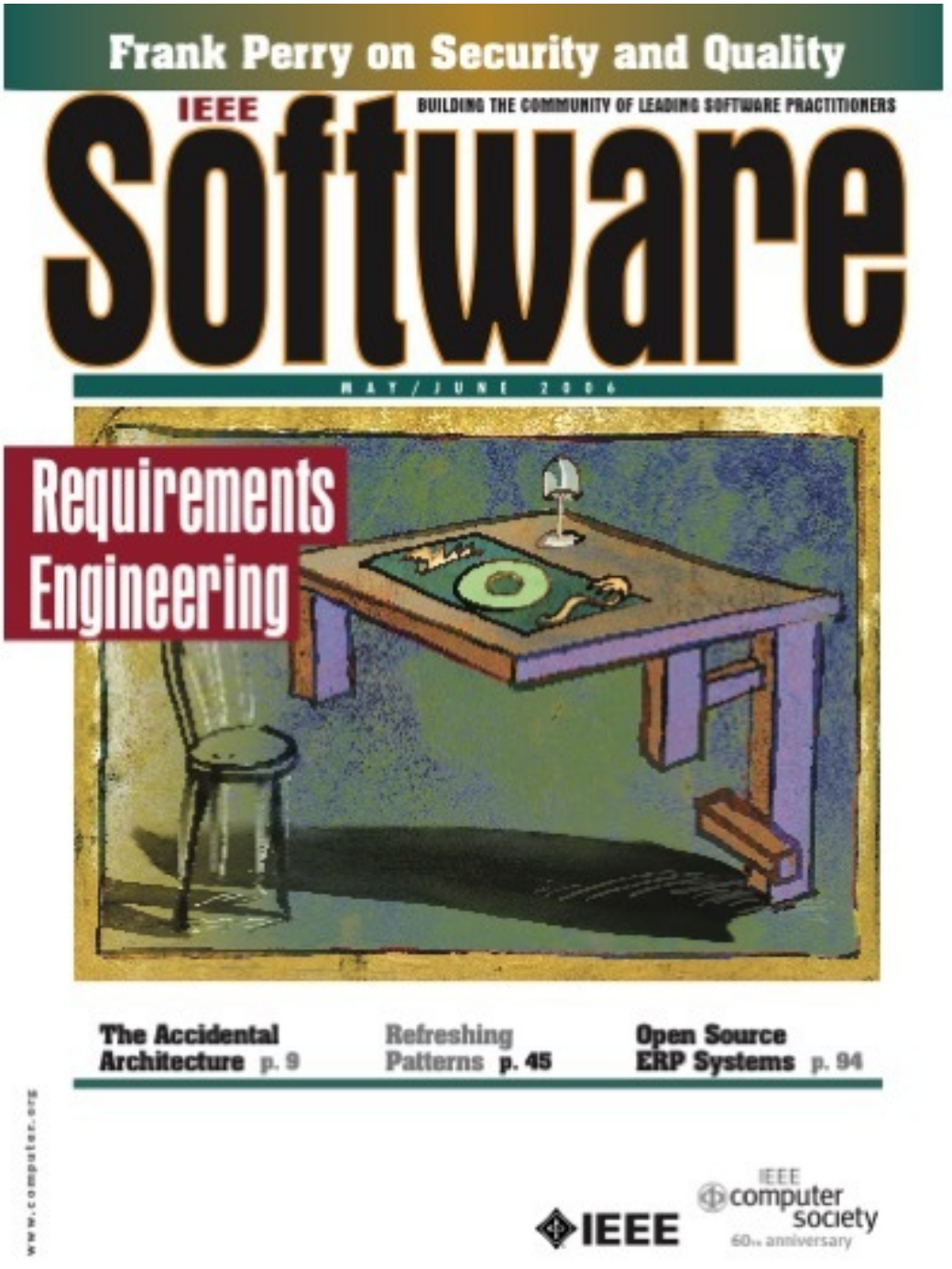
[518] DOI: 10.1109/MS.2006.62

“One way to **deal with bugs** is to **avoid them entirely** through stringent quality control. To conserve our valuable resources, we can use tools to catch the bugs before they end-up in production code. We can use **type-safe** languages, pay attention to **compiler warnings**, adopt specialized **bug-finding tools**, or we can adjust our code to make it locate bugs during **testing**.”

---

*Diomidis Spinellis, **Bug Busters**, IEEE Software, March 2006.*[519]

[519] DOI: 10.1109/MS.2006.40


Frank Perry on Security and Quality
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
MAY/JUNE 2006
Requirements Engineering
The Accidental Architecture p. 9
Refreshing Patterns p. 45
Open Source ERP Systems p. 94
www.computer.org
IEEE
IEEE computer society
60th anniversary

“Every interesting software-intensive system has an **architecture**. While some of these architectures are intentional, most appear to be **accidental**. Philippe **Kruchten** has observed that ‘the life of a software architect is a long and rapid **succession** of **suboptimal design** decisions taken partly **in the dark**.’ “

Grady Booch, ***The Accidental Architecture****, IEEE Software, May 2006.*[520]

[520] DOI: 10.1109/MS.2006.86

"The ramifications of **failing** to completely and correctly **address security** can **devastate an organization**, not only in compromised data and financial cost but also in the time and energy spent to recover."

Jane Huffman Hayes, Nancy Eickelmann, E. Ashlee Holbrook, Frank Perry, ***Security and Software Quality: An Interview with Frank Perry****, IEEE Software, May 2006.*[521]

[521] DOI: 10.1109/MS.2006.83

"Organizations frequently commit to requirements and contracts to **boost short-term revenues** without properly aligning sales, product management, project management, and marketing. Such misalignment results in **insufficient capacity planning** or product-development resource allocation, thus delaying projects."

Christof Ebert, ***Understanding the Product Life Cycle: Four Key Requirements Engineering Techniques****, IEEE Software, May 2006.*[522]

[522] DOI: 10.1109/MS.2006.88

"**Ethnographies** - using video to **observe users** in their own work environments–can support **requirements elicitation**."

Paul Luff, Marina Jirotka, ***Supporting Requirements with Video-Based Analysis****, IEEE Software, May 2006.*[523]

[523] DOI: 10.1109/MS.2006.84

“In recent years, the software engineering community has focused on **organizing** its **existing knowledge** and finding opportunities to transform that knowledge into a university curriculum. … they’ve also focused largely on SE’s engineering aspects, at the **expense of its human and social dimensions**.”

Hans van Vliet, ***Reflections on Software Engineering Education****, IEEE Software, May 2006.*[524]

[524] DOI: 10.1109/MS.2006.80

"**Debuggers** are cheap and effective tools. Typically we use them in a bottom-up fashion starting from the problem going to its source, but when this strategy fails, we might have to resort to a more tedious top-down breadth-first search. To locate bugs, we can also use **hardware-assisted data** and code **breakpoints**. For bugs that are difficult to reproduce, attaching a debugger to a running process as well as **postmortem** and **remote debugging** are some alternatives. Finally, we can **permanently embed** debugging knowledge in a program's source code, through **logging statements**."

Diomidis Spinellis, ***Debuggers and Logging Frameworks****, IEEE Software, May 2006.*[525]

[525] DOI: 10.1109/MS.2006.70

How Developers Use the Eclipse IDE

IEEE Software

BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

JULY/AUGUST 2006

Software Testing

Mobile Devices for Gathering Requirements p. 16

Choosing a Programming Language p. 62

Project Visualization for Software p. 84

www.computer.org

IEEE computer society 60th anniversary

"'Do you use **the same password** for multiple Web sites?' … **41 percent** of the respondents said they always use **the same password**, **45 percent** said they have a **few different passwords**, and **14 percent** said they **never use the same password** on multiple Web sites. Those 14 percent probably either **don't have an Internet connection** or are '**security professionals**.' The problem is that where the 'security professional' sees prudent, responsible behavior, users **simply see overhead** that gets in the way of performing whatever task they're trying to do."

---

*Warren Harrison, **Passwords and Passion**, IEEE Software, July 2006.*[526]

[526] DOI: 10.1109/MS.2006.110

“When **things go right**, software hums along like **well-oiled machinery**–receive an event, twiddle with inputs, send a flurry of messages, change the system state, interact with the environment or users, then wait for the next chunk of work. **Smooth**. Mechanical. Predictable. But what happens when **something goes wrong**? How should you design your software to **detect**, **react**, and **recover** from **exceptional conditions**?”

---

*Rebecca Wirfs-Brock, **Designing for Recovery**, IEEE Software, July 2006.*[527]

[527] DOI: 10.1109/MS.2006.98

"Software **development** is ultimately an **engineering activity**, whose primary activity is to **deliver executable artifacts** in a manner that **balances the forces** on that system. The forces that swirl around a software-intensive system include not only its purely **functional** requirements but also a multitude of **nonfunctional** ones, such as reliability, portability, and scalability (often called a system's -ilities)."

---

*Grady Booch, **From Small to Gargantuan**, IEEE Software, July 2006.*[528]

[528] DOI: 10.1109/MS.2006.102

“**Software testing** is still one of the more neglected practices within the software development life cycle.”

Natalia Juristo, Ana M. Moreno, Wolfgang Strigel, ***Guest Editors’ Introduction: Software Testing Practices in Industry***, *IEEE Software, July 2006.*[529]

[529] DOI: 10.1109/MS.2006.104

"Most companies and testing books use the **term unit testing**, but its **semantics varies** widely in different organizations. … Unit testing means **testing the smallest separate module** in the system. Some people … stress that it's the smallest specified module, but opinions differ about the need for specifications. Regardless, unit testing is **technically oriented**, with in/out parameters."

---

*Per Runeson, **A Survey of Unit Testing Practices**, IEEE Software, July 2006.*[530]

[530] DOI: 10.1109/MS.2006.91

"There's no language suitable for all tasks, and there probably won't ever be one. When **choosing a programming language**, you have to balance programmer **productivity**, **maintainability**, efficiency, portability, **tool support**, and software and hardware interfaces. … for some tasks, adopting an existing **domain-specific** language, building a new one, or using a **general-purpose** declarative language can be the right choice."

Diomidis Spinellis, ***Choosing a Programming Language****, IEEE Software, July 2006.*[531]

[531] DOI: 10.1109/MS.2006.97

"**Evidence-based reasoning** is becoming common in many fields. It's widely enshrined in the practice and teaching of medicine, law, and management, for example. **Evidence-based approaches** demand that, among other things, practitioners **systematically track** down the **best evidence** relating to some practice; **critically appraise** that evidence for validity, impact, and applicability; and **carefully document** it."

Tim Menzies, Jairus Hihn, ***Evidence-Based Cost Estimation for Better-Quality Software****, IEEE Software, July 2006.*[532]

[532] DOI: 10.1109/MS.2006.99

“Oddly enough, many software engineering issues and conflicts that were **relevant 20 years** ago are still **relevant today**.”

Robert L. Glass, ***How Much of the Software Engineering Old Still Remains New?***, *IEEE Software*, *July 2006.*[533]

[533] DOI: 10.1109/MS.2006.106


Brokering Technology Transfer
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
SEPTEMBER/OCTOBER 2006
Global Software Development
The Cone of Uncertainty Revisited p. 8
Listen to Your Tools and Materials p. 74
Risk Analysis in Software Testing p. 88
www.computer.org
IEEE
IEEE computer society
60th anniversary

“Despite the challenges and **complexities** involved in organizing and managing **globally distributed software development**, this phenomenon’s pace has been remarkable. Global software development seems to have become a **business necessity** for various reasons, including cost, scarcity of resources, and the need to locate development closer to the customers. In fact, it is fast becoming a **pervasive business phenomenon**.”

Deependra Moitra, Daniela Damian, ***Guest Editors’ Introduction: Global Software Development: How Far Have We Come?****, IEEE Software, September 2006.*[534]

[534] DOI: 10.1109/MS.2006.126

"Three closing comments are in order. … practices that promote **local knowledge** movement (such as informal discussions with peers) make **global knowledge management difficult**. … a common error we've encountered is having different, **unconnected KMSs** in place. … trying to **unify different cultures** in global organizations (except under crisis-management circumstances) might **not be the right approach**."

Peter Baloh, Kevin C. Desouza, Yukika Awazu, ***Managing Knowledge in Global Software Development Efforts: Issues and Practices****, IEEE Software, September 2006.*[535]

[535] DOI: 10.1109/MS.2006.135

"The availability of **testing infrastructure** is a major factor in product development project costs. Software **virtualization** is a powerful mechanism for **simulating a test setup** on a few desktops that would otherwise require 'real' equipment."

---

*Swaminathan Seetharaman, Krishna Murthy B.V.S., **Test Optimization Using Software Virtualization**, IEEE Software, September 2006.*[536]

[536] DOI: 10.1109/MS.2006.143

“**Participation in open source projects** can **make us better programmers** by exposing us to maintenance, new technologies, and different application domains, and make us better system administrators by forcing us to tinker with complex system setups. “

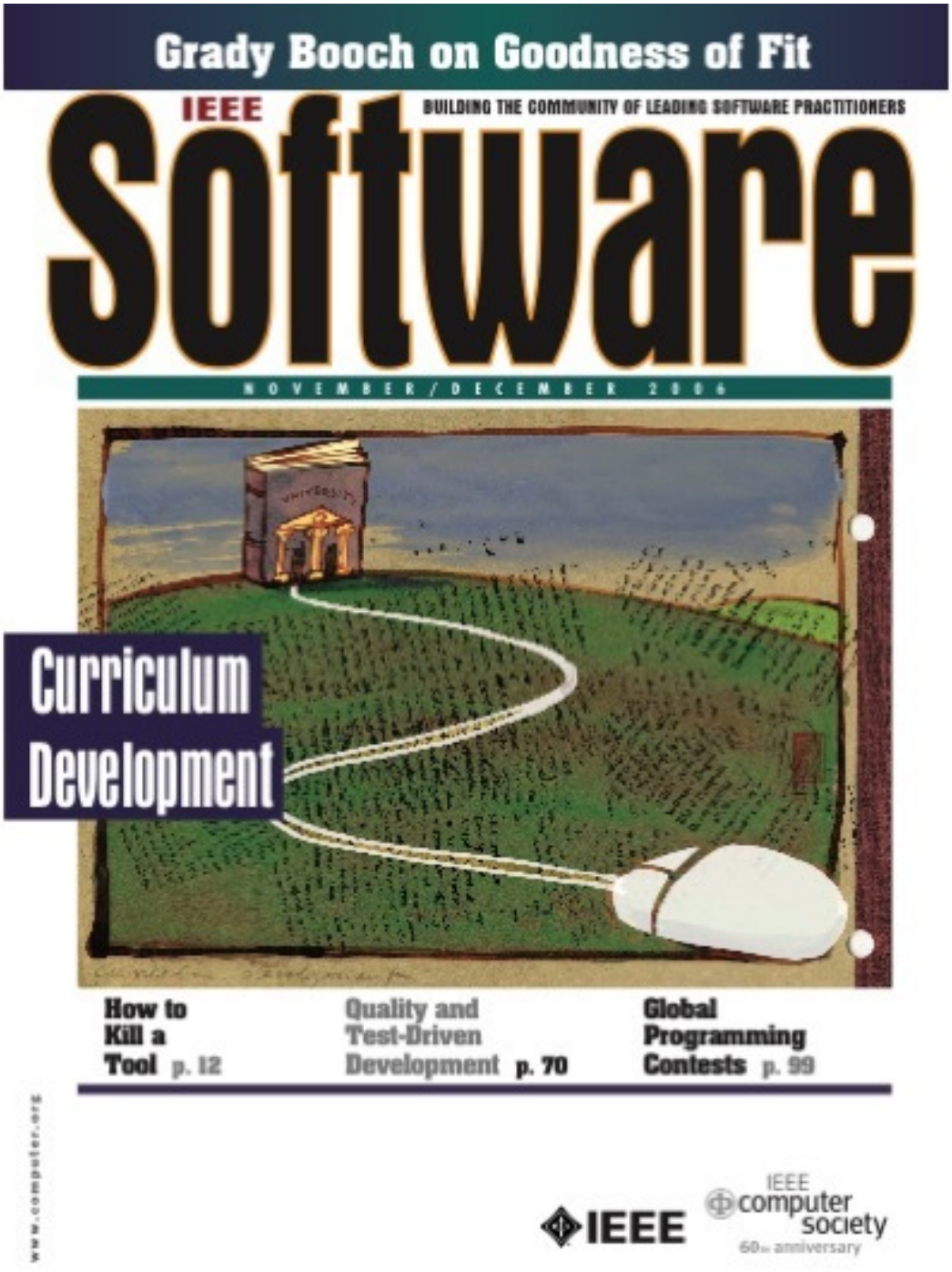
Diomidis Spinellis, ***Open Source and Professional Advancement****, IEEE Software, September 2006.*[537]

[537] DOI: 10.1109/MS.2006.136

“The following is a **familiar scenario**: management, feeling pressure from corporate headquarters and the marketplace, **dictates** that a product will be **released ahead of** the agreed-upon **schedule**. This is an all-too-common example of a company’s political climate in which influential people base their desire for success more on **personal agendas** than on quality. This scenario illustrates two key issues: the **frustrations** involved in **producing quality products** and the **conflicts** between self and team. “

Fran Boehme Mackin, Scott Stribrny, ***When Politics Overshadow Software Quality****, IEEE Software, September 2006.*[538]

[538] DOI: 10.1109/MS.2006.145


Grady Booch on Goodness of Fit
IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS
Software
NOVEMBER/DECEMBER 2006
Curriculum Development
How to Kill a Tool p. 12
Quality and Test-Driven Development p. 70
Global Programming Contests p. 99
www.computer.org
IEEE
IEEE computer society
60th anniversary

“On behalf of the Computer History Museum and the ACM, **Grady Booch** recently interviewed **John Backus**, who led the IBM team that created Fortran in the 1950s. Backus went on to coinvent the **Backus-Naur Form** (which was first applied to the definition of ALGOL), then later pioneered important advances in **functional programming**.”

---

*Grady Booch, **Goodness of Fit**, IEEE Software, November 2006.*[539]

[539] DOI: 10.1109/MS.2006.162

"The first **software engineering programs** were at the graduate level, primarily as terminal master's degrees for those already developing commercial and industrial software. By the early 1990s, educators began to consider software engineering's role at the undergraduate level."

Donald Bagert, Michael J. Lutz, ***Guest Editors' Introduction: Software Engineering Curriculum Development****, IEEE Software, November 2006.*[540]

[540] DOI: 10.1109/MS.2006.164

"The recommendations in **Software Engineering 2004**: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, form a volume of the larger Computing Curriculum project of the IEEE-CS and ACM. SE2004 evolved from an analysis of **desired student outcomes** for a software engineering graduate as compared to those for computer science and computer engineering graduates."

Timothy C. Lethbridge, Thomas B. Hilburn, Jorge L. D?az-Herrera, Ann E. Kelley Sobel, Richard J. LeBlanc Jr, ***SE2004: Recommendations for Undergraduate Software Engineering Curricula****, IEEE Software, November 2006.*[541]

[541] DOI: 10.1109/MS.2006.171

"There are many challenges in delivering a **software engineering curriculum** by **distance learning**. … Theses programs are characterized as part-time, open, large-scale **distance learning**, professionally accredited, and primarily aimed at practitioners in the IT industry."

Pete Thomas, Michel Wermelinger, Juan Fern?ndez-Ramil, Brendan Quinn, Leonor Barroca, Lucia Rapanotti, Bashar Nuseibeh, ***Learning Software Engineering at a Distance****, IEEE Software, November 2006.*[542]

[542] DOI: 10.1109/MS.2006.169

"**Open source software** offers a unique opportunity for **improving learning** outcomes for software engineering and computer science education."

---

*Kal Toth, **Experiences with Open Source Software Engineering Tools**, IEEE Software, November 2006.*[543]

[543] DOI: 10.1109/MS.2006.158

"In May 2000, the World Wide Web Consortium issued the specification for version 1.1 of the **Simple Object Access Protocol**. The name SOAP stuck; in version 1.2, the W3C ceased to consider **SOAP to be an acronym**. SOAP is just SOAP, a way applications can use XML to exchange structured and typed information. SOAP-based services are the foundation of the current push toward service-oriented Web architecture. Unfortunately, getting from the foundation to the complete edifice still involves a lot of work."

---

*Panagiotis Louridas, **SOAP and Web Services**, IEEE Software, November 2006.*[544]

[544] DOI: 10.1109/MS.2006.172

"**Quantification** helps you better understand your **requirements**, improving them and sometimes even revealing new **requirements** or stakeholders."

Neil Maiden, ***Improve Your Requirements: Quantify Them****, IEEE Software, November 2006.*[545]

[545] DOI: 10.1109/MS.2006.165

"The art of telling **a compelling design story** is understanding what your audience knows and what they need to know about your design, and then plotting your **storyline** accordingly."

Rebecca J. Wirfs-Brock, ***Explaining Your Design***, *IEEE Software, November 2006.*[546]

[546] DOI: 10.1109/MS.2006.159

“Software development currently seems to take a ‘**Roman**’ approach; that is, it focuses on the contributions of a group of programmers at an organization. Instead, it should take a ‘**Greek**’ approach, focusing on the contributions of individual, self-motivated programmers. But don’t let any unorganized **Barbarian** programmers near!”

---

*Robert L. Glass, **Greece vs. Rome: Two Very Different Software Cultures**, IEEE Software, November 2006.*[547]

[547] DOI: 10.1109/MS.2006.163

2007

IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

Software

SE Challenges in Small Companies

JANUARY/FEBRUARY 2007

IEEE computer society

www.computer.org/software

“Software systems usually have the **same basic architectural pattern** as their earlier incarnations, manifesting in decreasingly refined forms as we move back in time. Similarly, when a new problem confronts us, we try many different approaches, but over time, for the same kind of problem, **solutions tend to converge** to the same, more constrained, solution space.”

---

*Grady Booch, **It Is What It Is Because It Was What It Was**, IEEE Software, January 2007.*[548]

[548] DOI: 10.1109/MS.2007.19

"**Experienced programmers plan**, while **junior programmers jump** into their work. Some simpler **personal planning techniques** can help you eliminate waste when you work, write less code, design more simply, inject fewer defects, and generally **deliver sooner**. … The best way I know to **deliver sooner** is to **do less**."

---

*J.B. Rainsberger, **Personal Planning**, IEEE Software, January 2007.*[549]

[549] DOI: 10.1109/MS.2007.23

"**Small software organizations**–independently financed and organized companies with **fewer than 50 employees**–are fundamental to many national economies' growth. In the US, Brazil, Canada, China, India, Finland, Ireland, Hungary, and many other countries, small companies represent up to 85 percent of all software organizations. However, to persist and grow, small software companies need **efficient, effective software** engineering solutions."

Christiane Gresse von Wangenheim, Ita Richardson, ***Guest Editors' Introduction: Why are Small Software Organizations Different?****, IEEE Software, January 2007.*[550]

[550] DOI: 10.1109/MS.2007.12

“**Software process assessments** are typically the first step to commencing software process improvement. Small software companies find that many assessment methods are linked to **plan-driven improvement** models and can be **expensive** in terms of the resources required. “

Fergal Mc Caffery, Philip S. Taylor, Gerry Coleman, ***Adept: A Unified Assessment Method for Small Software Companies****, IEEE Software, January 2007.*[551]

[551] DOI: 10.1109/MS.2007.3

"There is (or should be) **more fun** in software engineering than you might think."

Robert L. Glass, ***Is Software Engineering Fun?***, *IEEE Software*, *January 2007.*[552]

[552] DOI: 10.1109/MS.2007.18

IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

Software

Stakeholders in Requirements Engineering

MARCH/APRIL 2007

IEEE computer society

www.computer.org/software

“Designing **incrementally**, keeping it **clean** as you go, can help you **avoid accidental complexity**. But doing this takes **discipline** and design **familiarity**.”

---

*Rebecca J. Wirfs-Brock, **Toward Design Simplicity**, IEEE Software, March 2007.*[553]

[553] DOI: 10.1109/MS.2007.57

"During an **architectural assessmen** … it's important to be **truthful** as well as **gentle**. … The development organization's unique task is to address all the essential concerns of all the important stakeholders and **avoid being blindsided** by unexpected problems and stakeholders."

Grady Booch, ***Speaking Truth to Power***, *IEEE Software*, *March 2007.*[554]

[554] DOI: 10.1109/MS.2007.53

"Accepting some of the testing team's responsibility by **writing your own tests** lets you trade the time you spend fixing defects for less time spent avoiding them in the first place."

---

*J.B. Rainsberger, **Avoiding Defects**, IEEE Software, March 2007.*[555]

[555] DOI: 10.1109/MS.2007.34

"The growing attention being paid to **stakeholders' needs and desires** reflects the growing importance of requirements engineering (RE) in software and systems development. … : identifying the stakeholders in a project, determining who and how important they are, prioritizing the **identified** stakeholder roles, and **selecting representative individuals** or groups from the identified and **prioritized** stakeholder roles with whom the development team can elicit and validate system requirements."

Roel J. Wieringa, Martin Glinz, ***Guest Editors' Introduction: Stakeholders in Requirements Engineering****, IEEE Software, March 2007.*[556]

[556] DOI: 10.1109/MS.2007.42

"**Requirements engineering** must manage the risks arising from project stakeholders. The **Outcome-Based Stakeholder Risk Assessment Model** (Obsram) provides guidance in stakeholder identification, identification of **stakeholder impacts and perceptions**, identification of potentially negative responses that pose risks to the project, and assessment and prioritization of those risks."

Richard W. Woolridge, Joanne E. Hale, Denise J. McManus, ***Stakeholder Risk Assessment: An Outcome-Based Approach****, IEEE Software, March 2007.*[557]

[557] DOI: 10.1109/MS.2007.54

"**Terminological interference** occurs in requirements engineering when stakeholders have **different interpretations** of the terms they use to describe their problem domain."

Nan Niu, Steve Easterbrook, ***So, You Think You Know Others' Goals? A Repertory Grid Study****, IEEE Software, March 2007.*[558]

[558] DOI: 10.1109/MS.2007.52

"The recommendations for **analyzing productivity** in the appendix to the ISO/IEC 15939 standard are inappropriate. … Problems with the ISO/IEC advice can be compounded if software engineers attempt to apply **statistical process-control** techniques to software productivity metrics."

Colin Connaughton, David Ross Jeffery, Barbara Kitchenham, ***Misleading Metrics and Unsound Analyses****, IEEE Software, March 2007.*[559]

[559] DOI: 10.1109/MS.2007.49

“**Eclipse** is an **open source** software project dedicated to providing a robust, full-featured, **and commercial-quality platform** for developing and supporting highly **integrated** software engineering **tools**.”

Michael Jiang, Zhihui Yang, ***Using Eclipse as a Tool-Integration Platform for Software Development****, IEEE Software, March 2007.*[560]

[560] DOI: 10.1109/MS.2007.58

IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

Software

Test-Driven Development

MAY | JUNE 2007

IEEE computer society

www.computer.org/software

"The **architecture** of a software-intensive system is largely **irrelevant to its end users**. Far more important to these stakeholders is the **system's behavior**, exhibited by raw, working source code. As long as a system provides the right answers at the right time with all the right other '-ilities' (maintainability, dependability, changeability, and so on), end users couldn't care less about what's behind the curtain making things work. To stakeholders other than end users, however, a system's architecture is intensely interesting. Moreover, software architecture has had a hand in **better project management**, greater use of **iterative development**, and leverage from the **Web's infrastructure**."

---

*Grady Booch, **The Irrelevance of Architecture**, IEEE Software, May 2007.*[561]

[561] DOI: 10.1109/MS.2007.93

"Knowing what tactic to take when someone **criticizes your design** is important. Designers need to recognize, accept, and **seek out valid criticism**, while **deflecting false criticisms** and **defusing aesthetic arguments**."

Rebecca J. Wirfs-Brock, ***Handling Design Criticism****, IEEE Software, May 2007.*[562]

[562] DOI: 10.1109/MS.2007.76

"**Test-driven development** is a discipline of design and programming where every line of new **code is written in response to a test** the programmer writes just before coding. … The ways TDD is being used in nontrivial situations (database development, embedded software development, GUI development, performance tuning), signifying an adoption level for the practice beyond the visionary phase and into the early mainstream."

Grigori Melnik, Ron Jeffries, ***Guest Editors' Introduction: TDD–The Art of Fearless Programming****, IEEE Software, May 2007.*[563]

[563] DOI: 10.1109/MS.2007.75

“A professional software developer ships **clean**, **flexible** code that **works-on time**. … **Test-driven development** is a discipline that helps developers behave in a more professional manner.”

---

*Robert C. Martin, **Professionalism and Test-Driven Development**, IEEE Software, May 2007.*[564]

[564] DOI: 10.1109/MS.2007.85

"Developers can use a **test-driven development** with **database schema** just as they use it with application code. Implementing test-driven database development (TDDD) involves three relatively simple steps: database **refactoring**, database **regression testing**, and **continuous** database **integration**."

Scott W. Ambler, ***Test-Driven Development of Relational Databases***, *IEEE Software, May 2007.*[565]

[565] DOI: 10.1109/MS.2007.91

“An explanation of test-driven development often begins by describing the **red-green-refactor** cycle. This slogan is so catchy and the description so simple that practitioners and tool developers tend to focus only on this **localized cycle**. Experience has shown that a successful functional test-driven development strategy must **span the entire application life cycle** and must be supported by effective tools. “

Jennitta Andrea, ***Envisioning the Next Generation of Functional Testing Tools****, IEEE Software, May 2007.*[566]

[566] DOI: 10.1109/MS.2007.73

IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

Software

Software Patterns

JULY | AUGUST 2007

IEEE computer society

www.computer.org/software

“Cognitive biases exist, and designers are remiss if we ignore them. By becoming aware of some common **cognitive biases in design discussions**, you can learn when it’s worthwhile to tweak your message to increase the likelihood of it being accepted.”

Rebecca J. Wirfs-Brock, ***Giving Design Advice***, *IEEE Software, July 2007.*[567]

[567] DOI: 10.1109/MS.2007.108

“**A domain-specific language** for building user interfaces offers a transparent way for programmers to specify interface elements. **Microsoft’s Extensible Application Markup Language** is an XML dialect for this purpose. However, **XAML** isn’t the only choice for programmers who wish to try a declarative approach, and some options are even open source.”

Panagiotis Louridas, ***Declarative GUI Programming in Microsoft Windows****, IEEE Software, July 2007.*[568]

[568] DOI: 10.1109/MS.2007.105

“Virtually all **well-structured music**, music that pleases the ear and moves the spirit, is **full of patterns**. By comparing musical and software patterns, the author helps clarify the purposes and forms of patterns. Architectural and design patterns make software-intensive systems easier to understand and adapt to because of their **regularity and simplicity**.”

---

*Grady Booch, **The Well-Tempered Architecture**, IEEE Software, July 2007.*[569]

[569] DOI: 10.1109/MS.2007.122

"**Patterns** have become part of the **software development mainstream**. They're available for all phases of the development process, including analysis, documentation, design, testing, and configuration management, to name a few."

Michael Kircher, Markus Völter, ***Guest Editors' Introduction: Software Patterns****, IEEE Software, July 2007.*[570]

[570] DOI: 10.1109/MS.2007.109

“For the past two decades, **software patterns** have significantly influenced how developers design and implement computing systems, well above and **beyond the most popular research** in the software field. “

Douglas C. Schmidt, Kevlin Henney, Frank Buschmann, ***Past, Present, and Future Trends in Software Patterns****, IEEE Software, July 2007.*[571]

[571] DOI: 10.1109/MS.2007.115

“All of software design involves developers **making decisions** and reifying those decisions in code. … However, architects often **fail to document** their decisions well. This leads to **architectural erosion**: decisions made during later development might conflict with the original architectural decisions and thus cause significant system disruptions. “

Neil B. Harrison, Uwe Zdun, Paris Avgeriou, ***Using Patterns to Capture Architectural Decisions****, IEEE Software, July 2007.*[572]

[572] DOI: 10.1109/MS.2007.124

"**Microsoft's patterns & practices** group conducted a survey that indicates a **significant gap** between the patterns **expert community** and the **software practitioners** attempting to use and leverage patterns in their daily work."

Jason Hogg, Dragos Manolescu, Wojtek Kozaczynski, Ade Miller, ***The Growing Divide in the Patterns World****, IEEE Software, July 2007.*[573]

[573] DOI: 10.1109/MS.2007.120

"**Researchers** have studied and created a wide range of techniques to **support software engineers** during development. … there's a high demand and acceptance for **unobtrusive**, **quickly** executable, and reactive **assistance** in core software engineering phases to help solve the problems at hand."

Eric Ras, Jörg Rech, Björn Decker, ***Intelligent Assistance in German Software Development: A Survey****, IEEE Software, July 2007.*[574]

[574] DOI: 10.1109/MS.2007.110

"**Asimov's Laws of Robotics** constrain robots to serve their human masters. Minor rewording shows that similar principles are very **relevant to software too**. These **laws of software** encompass a host of desiderata and trade-offs that software developers need to keep in mind. They also demonstrate that issues that are typically treated in a fragmented manner are actually strongly intertwined."

Dror G. Feitelson, ***Asimov's Laws of Robotics Applied to Software***, *IEEE Software, July 2007.*[575]

[575] DOI: 10.1109/MS.2007.100

IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

Software

Dynamically Typed Languages

SEPTEMBER/OCTOBER 2007

IEEE computer society

www.computer.org/software

"Building and acquiring software requires making **many decisions** and choosing between numerous solutions, yet the infrastructure to help people make decisions **based on good evidence** isn't well developed. The software engineering community needs better communication between researchers and practitioners to help make **useful bodies of evidence** available that can impact practice."

---

*Forrest Shull, **Who Needs Evidence, Anyway?**, IEEE Software, September 2007.*[576]

[576] DOI: 10.1109/MS.2007.152

“Since 2005, developers have used **Ajax** to let users interact with Web applications much as they do with desktop applications. But the **Ajax** features represented in applications such as **Google Maps**, **Netvibes**, and Zimbra Collaboration Suite demand detailed coding. **Ajax frameworks** are utility sets that make it easier to develop and maintain these applications.”

Juan Pablo Aroztegi, Nicolás Serrano, ***Ajax Frameworks in Interactive Web Apps****, IEEE Software, September 2007.*[577]

[577] DOI: 10.1109/MS.2007.132

"Building **clean abstractions** with **clearly defined extension points** is satisfying, but the best design choice isn't always obvious. How much access should you give a subclass to a class's inner workings? How much freedom should you give a subclass designer to 'bend' inherited behaviors to make a new abstraction fit in or to extend an existing one? These decisions involve nuanced reasoning. The contract between a class and its subclasses requires **thoughtful design**, **experimentation**, and **careful specification**."

---

*Rebecca J. Wirfs-Brock, **Designing Extensible Classes**, IEEE Software, September 2007.*[578]

[578] DOI: 10.1109/MS.2007.137

"**The architect**, either as an individual, a role, or a team, lovingly crafts, grows, and governs that architecture as it emerges from the **thousands of individual design decisions** of which it's composed. In this sense, an **architecture-first approach** appears to be a reflection of sound development practices."

---

*Grady Booch, **The Economics of Architecture-First**, IEEE Software, September 2007.*[579]

[579] DOI: 10.1109/MS.2007.146

"**Copy-pasting** code is a source of bugs. By employing in our programs **abstraction mechanisms** such as **functions**, **classes**, **types**, **decision tables**, **domain-specific languages**, and databases, we can abstract common elements into **parameterized reusable functionality**. However, **abstraction has its cost**. Its early gains are large, but eventually the benefits turn negative and the code becomes less comprehensible and maintainable. Deciding when abstracting is appropriate is what makes programming an art."

Diomidis Spinellis, ***Abstraction and Variation****, IEEE Software, September 2007.*[580]

[580] DOI: 10.1109/MS.2007.127

"**The languages** discussed … have a long history, which is perhaps why some have had **several different names** over the years. One such language is **Lisp**, the second-oldest programming language. For years, many somewhat dismissively described languages such as Lisp as '**scripting languages**.' Today, we more commonly refer to them as **dynamically typed languages**, typified by **Python and** Ruby, **and** their impact is arguably greater than ever."

Roel Wuyts, Laurence Tratt, ***Guest Editors' Introduction: Dynamically Typed Languages****, IEEE Software, September 2007.*[581]

[581] DOI: 10.1109/MS.2007.140

“Five years ago, the team at Reflexis ran into **a little language from Brazil**. Lua (pronounced loo-ah) changed the way they work profoundly. It lets them create hybrid solutions that combine the strengths of **statically typed software** with the flexibility of a **dynamically typed environment**. In short, with Lua, they get the best of both worlds. Lua can help you become more productive by extending your **C/C++** creations with the expressive power and flexibility of a dynamically typed language.”

---

*Ashwin Hirschi, **Traveling Light, the Lua Way**, IEEE Software, September 2007.*[582]

[582] DOI: 10.1109/MS.2007.150

"**Python**, a dynamically typed language, can implement these software frameworks for major OR operational research methodologies (mathematical programming and simulation) in the same programming environment. By doing so, **Python**, in effect, **glues software environments** that have been independent, thus improving the software development cycle for sophisticated applications requiring different model types."

Suleyman Karabuk, F. Hank Grant, ***A Common Medium for Programming Operations-Research Models****, IEEE Software, September 2007.*[583]

[583] DOI: 10.1109/MS.2007.125

"The emergence of the **model-driven development** paradigm has revitalized interest in **domain-specific languages**. **Embedding a DSL** in a dynamic language facilitates rapid development."

Jesús García Molina, Jesús Sánchez Cuadrado, ***Building Domain-Specific Languages for Model-Driven Development****, IEEE Software, September 2007.*[584]

[584] DOI: 10.1109/MS.2007.135

“By harnessing **Smalltalk’s dynamic nature** and reflective capabilities, Seaside is able to incorporate key features such as a component architecture that supports multiple, simultaneously active control flows; programmatical **XHTML** generation; and on-the-fly debugging, code editing, and recompilation.”

Stéphane Ducasse, Lukas Renggli, Adrian Lienhard, ***Seaside: A Flexible Environment for Building Dynamic Web Applications****, IEEE Software, September 2007.*[585]

[585] DOI: 10.1109/MS.2007.144

"Although the **Semantic Web** is **data oriented** and the **World Wide Web** is **document oriented**, both are fundamentally **decentralized**, **heterogeneous**, and **open**. The Semantic Web isn't a global database, centralized in one location with one agreed-upon schema and one meaning. Instead, anyone can make any statement at any location, using any vocabulary or structure."

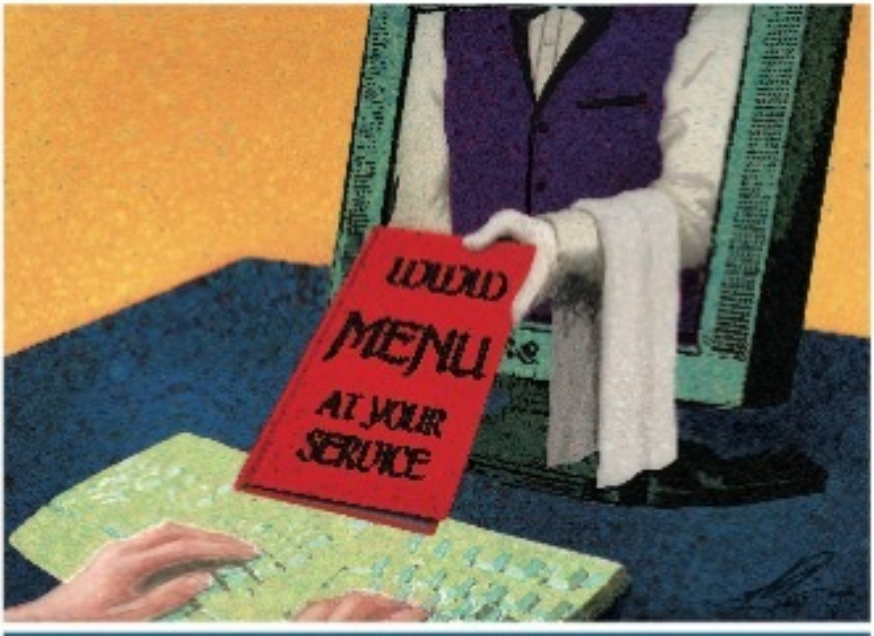
Eyal Oren, Armin Haller, Manfred Hauswirth, Benjamin Heitmann, Stefan Decker, Cédric Mesnage, ***A Flexible Integration Framework for Semantic Web 2.0 Applications****, IEEE Software, September 2007.*[586]

[586] DOI: 10.1109/MS.2007.126

IEEE
BUILDING THE COMMUNITY OF LEADING SOFTWARE PRACTITIONERS

Software

Service-Centric Software Systems



NOVEMBER | DECEMBER 2007

IEEE computer society

www.computer.org/software

"Software development teams need to **make decisions** effectively **as a team**, but most groups spend more time in **meetings and discussions** than they need. Sometimes, to **evaluate an idea**, we have to **try it** for a while. If you see your team bogged down in meetings, it's time to **run an experiment** and replace **opinions** with **facts**."

---

*J.B. Rainsberger, **Just Try It**, IEEE Software, November 2007.*[587]

[587] DOI: 10.1109/MS.2007.171

"**Yukijiro Matsumoto**, chief designer of the **Ruby** programming language, claims brevity is one of the most important contributors to **beautiful code**. Although brevity can contribute to code beauty, clarity of purpose, expressive use of the programming language, and design elegance also play a part. But is there more to **good design** than **beautiful code**?"

Rebecca J. Wirfs-Brock, ***Does Beautiful Code Imply Beautiful Design?****, IEEE Software, November 2007.*[588]

[588] DOI: 10.1109/MS.2007.163

"**Paper as a tool** for **expressing our thoughts** offers superb usability and versatility, letting us mix various notations and multiple levels of abstraction. This makes it easier to pour out our thoughts on it. In contrast to software tools, paper doesn't provide feedback and **won't interrupt us** with various notifications. This gives us a **chance to meditate** on our design in a state of flow."

---

*Diomidis Spinellis, **On Paper**, IEEE Software, November 2007.*[589]

[589] DOI: 10.1109/MS.2007.173

"A comparison of **building architecture** and **software architecture** reveals the differences, congruences, and commonalities between the two. There are differences in cost estimation, but there are similarities in divisions of labor or knowledge, degrees of formality, and the use of different viewpoints, use cases, an incremental design, and a particular style."

---

*Grady Booch, **Artifacts and Process**, IEEE Software, November 2007.*[590]

[590] DOI: 10.1109/MS.2007.159

"**Service-centric software system** is a multidisciplinary paradigm concerned with software systems that are constructed as **compositions of autonomous services**. These systems extend the service-oriented architecture paradigm by focusing on the design, development, and maintenance of software built under SOAs."

Olivier Nano, Andrea Zisman, ***Guest Editors' Introduction: Realizing Service-Centric Software Systems****, IEEE Software, November 2007.*[591]

[591] DOI: 10.1109/MS.2007.166

"**Composing software services** requires solving both **low-level technical problems** and **high-level semantic issues**."

Philippe Lalanda, Cristina Marin, ***A Domain-Configurable Development Environment for Service-Oriented Applications****, IEEE Software, November 2007.*[592]

[592] DOI: 10.1109/MS.2007.154

"Development of today's information systems must take a **service-centric** approach. The drive is toward designing **service-centric** systems involving **well-defined**, **loosely coupled** services that are **reusable**, **discoverable**, and **composable**."

Dieter Fensel, Maciej Zaremba, Michal Zaremba, Tomas Vitvar, Matthew Moran, ***SESA: Emerging Technology for Service-Centric Environments****, IEEE Software, November 2007.*[593]

[593] DOI: 10.1109/MS.2007.178

“Open source development is often regarded as a chaotic environment where new initiatives’ success or failure just happens by chance. However, **successful open source communities** are applying **incubation processes** for **managing the risks** associated with creating new projects. … e.g. **Apache** and **Eclipse** communities.”

---

*José L. Ruiz, Manuel Santillán, Hugo A. Parada G., Félix Cuadrado, Juan C. Dueñas, **Apache and Eclipse: Comparing Open Source Project Incubators**, IEEE Software, November 2007.*[594]

[594] DOI: 10.1109/MS.2007.157

"**Ruby on Rails** is a novel Web 2.0 application development framework that attempts to combine **PHP**'s simple immediacy with **Java**'s architecture, purity, and quality."

---

*Paul Kirchberg, Michael Bächle, **Ruby on Rails**, IEEE Software, November 2007.*[595]

[595] DOI: 10.1109/MS.2007.176

2008

IEEE
Software

Quality Requirements

MARCH/APRIL 2008

IEEE computer society
www.computer.org/software

“**Measuring** software development **productivity** is difficult, but it’s not impossible. It’s prone to misuse and misinterpretation, and highly portable, precise measures are elusive. But we can still implement meaningful measures, provided we understand why we’re doing it and provided we’re aware of their limitations.”

Hakan Erdogmus, ***Measurement Acquiescence****, IEEE Software, March 2008.*[596]

[596] DOI: 10.1109/MS.2008.40

"As the code written today becomes part of tomorrow's inexorably **growing legacy**, preserving these stories becomes increasingly important. It's costly to rely upon informal **storytelling** to **preserve** and **communicate important decisions**; it's incredibly costly to try to recreate those decisions and their rationale when the storytellers themselves are gone. Insofar as a software development organization can **preserve** its stories in a system's written architecture, it can make evolving that system materially easier."

---

*Grady Booch, **Tribal Memory**, IEEE Software, March 2008.*[597]

[597] DOI: 10.1109/MS.2008.52

"When dealing with **quality requirements**, you often end up in difficult **trade-off analysis**. You must take into account aspects such as **release** targets, **end-user experience**, and **business opportunities**. At the same time, you must consider what is **feasible** with the evolving system architecture and the available development resources. Our experience from the mobile-phone domain shows that much can be gained if development team members **share** a common **framework of quality indicators** and have a **simple**, easy-to-use model for reasoning about quality targets."

Richard Berntsson Svensson, Björn Regnell, Thomas Olsson, ***Supporting Roadmapping of Quality Requirements****, IEEE Software, March 2008.*[598]

[598] DOI: 10.1109/MS.2008.48

"**Test-driven development** (TDD) is first and foremost a design practice. The question is, 'How good are the resulting designs?' … TDD programmers tend to write software modules that are **smaller**, **less complex**, and more **highly tested** than modules produced by their test-last counterparts. However, the results didn't support claims for lower coupling and increased cohesion with TDD."

David Janzen, Hossein Saiedian, ***Does Test-Driven Development Really Improve Software Design Quality?****, IEEE Software, March 2008.*[599]

[599] DOI: 10.1109/MS.2008.34

“The **Business Process Execution Language** specifies Web services that work together to perform a business process. BPEL is an **orchestrating language**: it sets down exactly how the Web services will cooperate to carry out the overall business process. BPEL is an **XML-based programming language** - that is, you write BPEL programs in XML. Because XML wasn’t designed with programmers in mind, the programming results aren’t prime examples of elegance. Fortunately, you rarely need to write in BPEL by hand. Most BPEL programs are written using **special graphical editors** that let you describe the business process diagrammatically and then automatically generate the corresponding BPEL code.”

Panagiotis Louridas, ***Orchestrating Web Services with BPEL****, IEEE Software, March 2008.*[600]

[600] DOI: 10.1109/MS.2008.42

"**Google**'s position as a leading Web-based applications platform and its embrace of **rigorous incremental testing** might be the vanguard of a new definition of what software testing encompasses."

Greg Goth, ***'Googling' Test Practices? Web Giant's Culture Encourages Process Improvement****, IEEE Software, March 2008.*[601]

[601] DOI: 10.1109/MS.2008.28

IEEE
Software

Quantitative Project Management

MAY | JUNE 2008

IEEE computer society

www.computer.org/software

"**Model-based testing (MBT)** approaches help automatically **generate test cases** using models extracted from software artifacts, … certain **specialized domains** are applying MBT, but it does not yet seem to be a mainstream approach."

Arilo Dias Neto, Rajesh Subramanyan, Forrest Shull, Guilherme Horta Travassos, Marlon Vieira, ***Improving Evidence about Software Technologies: A Look at Model-Based Testing****, IEEE Software, May 2008.*[602]

[602] DOI: 10.1109/MS.2008.64

"Designers need to sharpen their **focus** and apply **design energy** where it will have the most impact. So, identifying **what's core** to our system's success is one of the most critical things we must do."

---

*Rebecca J. Wirfs-Brock, **Design Strategy**, IEEE Software, May 2008.*[603]

[603] DOI: 10.1109/MS.2008.58

"How to apply **user-centered design** methods to design interactive systems for the **elderly**. A case study (with its successes and weaknesses) showed that a need exists for more **creative and participatory** design approaches for this population."

Ulrike Pfeil, Helena Sustar, Panayiotis Zaphiris, ***Requirements Elicitation with and for Older Adults****, IEEE Software, May 2008.*[604]

[604] DOI: 10.1109/MS.2008.69

"What is the **optimal design** for a given system, a design that reasonably **balances all the forces** that weigh in on the problem? In turn, what is the **optimal organizational structure** for developing, deploying, and evolving that system? The challenge for every organization is to find the sweet spot that provides the **right balance** of **innovation**, **predictability**, **repeatability**, and **risk confrontation** for that project at every given moment."

---

*Grady Booch, **Architectural Organizational Patterns**, IEEE Software, May 2008.*[605]

[605] DOI: 10.1109/MS.2008.56

"**Salesforce.com** has used the RITE (**Rapid Iterative Testing and Evaluation**) method to quickly and iteratively improve its software design. RITE has helped the company retain high quality while increasing its rate of delivery using an agile development approach."

Jeff Patton, ***Getting Software RITE***, *IEEE Software*, *May 2008.*[606]

[606] DOI: 10.1109/MS.2008.62

"Software building's golden rule is that you should **automate all build tasks**. The most popular tool options for doing this are the facilities that your integrated development environment (IDE) provides, the various implementations of **Make**, and **Apache Ant** and **Maven**."

---

*Diomidis Spinellis, **Software Builders**, IEEE Software, May 2008.*[607]

[607] DOI: 10.1109/MS.2008.74

“A successful project effectively manages four cornerstones - **schedule**, **cost**, **scope**, and **quality** - to achieve its goals. Every project activity influences these four cornerstones. **Stochastic optimization modeling** factors in the uncertainties associated with project activities and provides insight into the expected project outputs as probability distributions rather than as deterministic approximations.”

Uma Sudhakar Rao, Chinmay Pradhan, Srikanth Kestur, ***Stochastic Optimization Modeling and Quantitative Project Management****, IEEE Software, May 2008.*[608]

[608] DOI: 10.1109/MS.2008.77

"**Open source** refers to software that you may freely use, modify, or distribute provided you observe certain restrictions with respect to copyright and protection of its open source status. A major difference between free and open source software (FOSS) and freeware or public-domain software is that FOSS generally has a copyright. FOSS isn't free software and often **requires substantial investment** before you can deploy it in the marketplace. "

Christof Ebert, ***Open Source Software in Industry***, *IEEE Software, May 2008.*[609]

[609] DOI: 10.1109/MS.2008.67

"Software development **effort estimates** are reported to be highly **inaccurate** and systematically **overly optimistic**. Empirical evidence suggests that this problem is caused to some extent by the influence of **irrelevant and misleading information** – for example, information about the client's budget. The only effective way to eliminate this influence is to **avoid exposure** to such information."

Stein Grimstad, Magne J?rgensen, ***Avoiding Irrelevant and Misleading Information When Estimating Development Effort****, IEEE Software, May 2008.*[610]

[610] DOI: 10.1109/MS.2008.57

"People might love to support **underdogs**, but they also **love to kick them** when they're down. And, at this point in time at least, software is the world's technological underdog!"

---

*Robert L. Glass, **Software: Hero or Zero?**, IEEE Software, May 2008.*[611]

[611] DOI: 10.1109/MS.2008.75

IEEE
Software

Developing Scientific Software

JULY | AUGUST 2008

IEEE computer society

www.computer.org/software

"**Process trends** can be placed inside a triangular map according to their emphasis on three aspects, represented by the vertices: **people**, **technology**, and **rigor**. **Plan-oriented**, engineering, and research-based approaches tend to view software as a rigid artifact, so they **stress technology** and **rigor** over **people**. **Evolutionary approaches** tend to view software development as an organic, skills-driven technical activity, so they **stress people** and **technology** over **rigor**. … A more complete scheme requires dissection in terms of seven essential, mutually reinforcing characteristics: **human-centricity**, **technical orientation**, **discipline**, **pragmatism**, **empiricism**, **experimentation**, and **value orientation**."

---

*Hakan Erdogmus, **Essentials of Software Process**, IEEE Software, July 2008.*[612]

[612] DOI: 10.1109/MS.2008.87

"There can be significant benefits in thinking through a design until you get it 'right enough' before launching into a major development effort. For such **up-front design** to be effective, you must develop **a design rhythm** that balances **thinking**, **learning**, **and** doing."

Rebecca J. Wirfs-Brock, ***Up-front Design***, *IEEE Software, July 2008.*[613]

[613] DOI: 10.1109/MS.2008.104

"**Without refactoring**, complex software-intensive **systems** become **increasingly irregular** and thus increasingly chaotic over time. We can **understand** complex software systems only when they're **nearly decomposable** and **hierarchic**. One measure … is **lines of source code**: the greater the SLOC, the **more inertia** to change the system will have, the more people it will take to keep it fed, the more stakeholders who will be crawling all over it. … the more complex measures … are tuned to Philippe Kruchten's 4+1 view model of architecture."

---

*Grady Booch, **Measuring Architectural Complexity**, IEEE Software, July 2008.*[614]

[614] DOI: 10.1109/MS.2008.91

“Not all **scientific computing** is high-performance computing - **the variety** of scientific software is huge. Such software might be complex **simulation** software developed and running on a **high-performance computer**, or software developed on a **PC** for embedding into **instruments**; for manipulating, analyzing or **visualizing** data or for orchestrating **workflows**.”

Judith Segal, Chris Morris, ***Developing Scientific Software****, IEEE Software, July 2008.*[615]

[615] DOI: 10.1109/MS.2008.85

"The development of **scientific software** involves **risk** in the **underlying theory**, its implementation, and its use. … If the software's purpose **shifts away** from just **showing the theory's viability**, risk shifts to **the implementation**. At this point, testing must assess **the implementation**, not the theory. Most **scientists miss** this shift."

Rebecca Sanders, Diane Kelly, ***Dealing with Risk in Scientific Software Development****, IEEE Software, July 2008.*[616]

[616] DOI: 10.1109/MS.2008.84

"Studies of computational **scientists** developing software for **high-performance computing systems** indicate that these scientists face **unique** software engineering **issues**. Previous failed attempts to transfer SE technologies to this domain haven't always taken these issues into account. To **support scientific-software development**, the SE community can disseminate appropriate practices and processes, develop educational materials specifically for computational scientists, and investigate the large-scale reuse of development frameworks."

Jeffrey C. Carver, Lorin M. Hochstein, Daniela Cruzes, Victor R. Basili, Jeffrey K. Hollingsworth, Marvin V. Zelkowitz, Forrest Shull, ***Understanding the High-Performance-Computing Community: A Software Engineer's Perspective****, IEEE Software, July 2008.*[617]

[617] DOI: 10.1109/MS.2008.103

"**Scientific workflows** - models of computation that capture the **orchestration** of scientific codes to conduct in silico research - are gaining recognition as an attractive **alternative** to **script-based orchestration**. Even so, researchers developing scientific workflow technologies still face fundamental challenges, including developing the **underlying science of scientific workflows**. You can classify scientific-workflow environments according to three major phases of in silico research: **discovery**, **production**, and **distribution**."

David Woollard, Nenad Medvidovic, Yolanda Gil, Chris A. Mattmann, ***Scientific Software as Workflows: From Discovery to Distribution****, IEEE Software, July 2008.*[618]

[618] DOI: 10.1109/MS.2008.92

“**Reactive systems** that service **multiple clients** or users are often **highly configurable** to provide **customized**, value-added services to individual users. A large **configuration space** is characteristic of such systems, resulting in a **large test state space**.”

Tony Savor, ***Testing Feature-Rich Reactive Systems***, *IEEE Software, July 2008.*[619]

[619] DOI: 10.1109/MS.2008.99

"**Checklists** are an important part of **code** and **design inspections**. Ideally, they aim to increase the number of faults found per inspection hour by highlighting known areas of previous failure. … The author subjects checklists' effectiveness to **formal statistical testing**, using data from **308 inspections** by industrial engineers over a three-year period. The results showed **no evidence** that checklists **significantly improved** these inspections."

Les Hatton, ***Testing the Value of Checklists in Code Inspections****, IEEE Software, July 2008.*[620]

[620] DOI: 10.1109/MS.2008.100

"**Software engineers** will do **better work and stay with a company** if they are **motivated** - as a result the success of software projects is likely to improve. … in-depth review of the **92 studies** published in the last **25 years** on software engineer motivation … give an overview of what **managers need to know** to improve motivation among their employees."

Nathan Baddoo, Tracy Hall, Sarah Beecham, Helen Sharp, Hugh Robinson, ***What Do We Know about Developer Motivation?****, IEEE Software, July 2008.*[621]

[621] DOI: 10.1109/MS.2008.105

"The software development process and the resulting product are so complex that no **error-detecting approach** will ever be able to produce **error-free software**."

Robert L. Glass, ***Two Mistakes and Error-Free Software: A Confession***, *IEEE Software, July 2008.*[622]

[622] DOI: 10.1109/MS.2008.102

IEEE
Software

Software Development Tools

SEPTEMBER/OCTOBER 2008

IEEE computer society

www.computer.org/software

"A **not-so-subtle divide** separates **empirical** and **constructionist software research**. **Constructionists** maintain that software research should be about creating technologies, devising **new methods**. **Empiricists** are interested in studying and **understanding existing approaches**. The antagonism between the two camps does not serve our industry well - it needs both modes of research."

Hakan Erdogmus, ***Must Software Research Stand Divided?****, IEEE Software, September 2008.*[623]

[623] DOI: 10.1109/MS.2008.120

"**Trust** is a **subjective**, **user-centric**, **context-dependent** concept, and is thus difficult to define universally. **On the Internet**, several factors make trust more difficult to build, explaining why some successful brick-and-mortar retail chains have been **unable** to **translate** their **reputation** to the **virtual platform** the Web offers."

Patricia Beatty, Ejike Ofuonye, Scott Dick, James Miller, Ian Reay, ***How Do We Build Trust into E-commerce Web Sites?****, IEEE Software, September 2008.*[624]

[624] DOI: 10.1109/MS.2008.136

"Some **software engineering ideas** have a **half-life**. … this **half-life** is **roughly five years**, … the need for software engineers to thus stay abreast of **new technologies**."

Philippe Kruchten, ***The Biological Half-Life of Software Engineering Ideas***, *IEEE Software, September 2008.*[625]

[625] DOI: 10.1109/MS.2008.127

"A **software agent** is defined as an **encapsulated** software entity with one or more **specified goals**. To fulfill these goals, an agent shows **autonomous behavior** and **interacts** continuously with its environment and other agents."

Hisham Mubarak, ***Developing Flexible Software Using Agent-Oriented Software Engineering****, IEEE Software, September 2008.*[626]

[626] DOI: 10.1109/MS.2008.135

"**Static analysis** examines code in the **absence of input data** and **without running the code**. It can detect potential **security violations** (SQL injection), **runtime errors** (dereferencing a null pointer) and **logical inconsistencies** (a conditional test that can't possibly be true). … **FindBugs**, an open source static-analysis tool for Java … evaluates what kinds of defects can be effectively detected with relatively simple techniques and helps developers understand how to incorporate such tools into software development."

Nathaniel Ayewah, John Penix, J. David Morgenthaler, William Pugh, David Hovemeyer, ***Using Static Analysis to Find Bugs****, IEEE Software, September 2008.*[627]

[627] DOI: 10.1109/MS.2008.130

"**Refactoring tools** can **improve** the **speed and accuracy** with which developers create and maintain software - but only **if they are used**. In practice, tools are not used as much as they could be; this seems to be because sometimes they do not align with the refactoring tactic preferred by most programmers, a tactic the authors call 'floss refactoring.' They propose five principles that characterize successful floss-refactoring tools—principles that can help programmers to choose the most appropriate refactoring tools and also help toolsmiths to design tools that fit the programmer's purpose."

Emerson Murphy-Hill, Andrew P. Black, ***Refactoring Tools: Fitness for Purpose****, IEEE Software, September 2008.*[628]

[628] DOI: 10.1109/MS.2008.123

“**Over time**, software systems suffer gradual **quality decay** and therefore costs can rise if organizations fail to take **proactive countermeasures**. Quality control is the first step to avoiding this cost trap. **Continuous quality assessments** help users identify quality problems early, when their removal is still inexpensive; they also aid decision making by providing an integrated view of a software system’s current status. As a **side effect**, continuous and timely feedback helps developers and maintenance personnel **improve** their **skills** and thereby decreases the likelihood of future quality defects. To make regular quality control **feasible**, it must be **highly automated**, and assessment results must be presented in an aggregated manner to **avoid overwhelming users** with data. “

Florian Deissenboeck, Stefan Wagner, Markus Pizka, Benjamin Hummel, Elmar Juergens, Benedikt Mas y Parareda, ***Tool Support for Continuous Quality Control****, IEEE Software, September 2008.*[629]

[629] DOI: 10.1109/MS.2008.129

"Design teams rarely **consider multiple solution ideas** before committing to one. They often forget that an even better idea could be just around the corner, and consider alternative ideas only when they don't like the current one. Using **sketchboarding**, design studio, or a combination of these two techniques can let teams quickly ideate over many solutions. They then have a chance to arrive at a solution that no one individual had thought of."

---

*Jeff Patton, **Consider Multiple Solutions**, IEEE Software, September 2008.*[630]

[630] DOI: 10.1109/MS.2008.134

“**Software resource estimation** methods and models have had a major impact on successful software engineering practice. They provide milestone budgets and schedules that help projects determine when they are making **satisfactory progress** and when they need **corrective action**. They help decision makers analyze **software cost-schedule-value trade-offs** and make decisions regarding investments, outsourcing, COTS products, and legacy software phaseouts. They help organizations **prioritize investments** in improving software productivity, quality, and time to market. “

Ricardo Valerdi, Barry W. Boehm, ***Achievements and Challenges in Cocomo-Based Software Resource Estimation****, IEEE Software, September 2008.*[631]

[631] DOI: 10.1109/MS.2008.133

"Despite various industry reports about the **failure rates** of software projects, there's still **uncertainty** about the **actual figures**. Researchers performed a global Web survey of IT departments in 2005 and 2007. The results suggest that the software **crisis is perhaps exaggerated** and that most software projects deliver. However, the overall project failure rate, including cancelled and completed but poorly performing projects, remains arguably high for an applied discipline."

Khaled El Emam, A. Günes Koru, ***A Replicated Survey of IT Software Project Failures****, IEEE Software, September 2008.*[632]

[632] DOI: 10.1109/MS.2008.107

“Software developers often need to understand a large body of **unfamiliar code** with little or **no documentation**, **no experts** to consult, and **little time** to do it. … The most common suggestions were to use a **code navigation tool**, use a **design recovery** tool, use a **debugger** to step through the code, create a **runtime trace**, use problem-based learning, ask people for help, study the code from top down, and **print out all** the code.”

Sukanya Ratanotayanon, Susan Elliott Sim, ***Inventive Tool Use to Comprehend Big Code****, IEEE Software, September 2008.*[633]

[633] DOI: 10.1109/MS.2008.118

“Every new line of code quickly becomes **legacy**. When that **legacy** mounts, it forms a significantly massive pile of software, which cannot be ignored. … what we can do intentionally with such piles, from **abandonment** to **evolution** and many **things in between**.”

---

*Grady Booch, **Nine Things You Can Do with Old Software**, IEEE Software, September 2008.*[634]

[634] DOI: 10.1109/MS.2008.139

"**One member** of a work team can **decrease** the whole **team's productivity**."

Robert L. Glass, ***Negative Productivity and What to Do about It***, *IEEE Software*, *September 2008.*[635]

[635] DOI: 10.1109/MS.2008.121

IEEE
Software

Opportunistic System Development

25 YEARS

NOVEMBER|DECEMBER 2008

Celebrating 25 Years of *IEEE Software*:
A 25-Year History of Software Milestones

IEEE computer society

www.computer.org/software

"In the world of **user-centered design** thinking, **Alan Cooper** is responsible for many of the tenets used in **interaction design practice** today. Most notably, he introduced the use of **personas** to distill and make relevant information about a system's users, information we subsequently use to drive interaction design."

Jeff Patton, ***A Conversation with Alan Cooper: The Origin of Interaction Design****, IEEE Software, November 2008.*[636]

[636] DOI: 10.1109/MS.2008.142

"Over the **past 25 years**, we've made **great advances** in tooling, technologies, and techniques that make software **design** more concrete. But **design** still **requires careful thought**."

---

*Grady Booch, **Back to the Future**, IEEE Software, November 2008.*[637]

[637] DOI: 10.1109/MS.2008.144

“Many major **technology breakthroughs** happened **before 1984**: Milestones such as the **IBM OS/360** and **the microprocessor**, and even many still-relevant software engineering practices, had been developed much earlier. So what makes the recent 25 years unique? First, software **moved** from a few company desks to the lives of practically **everyone on the planet**. The **PC**, the **Internet**, and **mobile phones** showcase this tremendous evolution. Second, empirical evaluations overcame opinions. Mary Shaw described the eighties by stating, ‘Software engineering is not yet a true discipline, but it has the potential to become one.’ In those early days, a lot of technologies were just assembled and delivered, but from the ’80s onward, engineers evaluated and empirically assessed new technologies to **judge their impact**.”

Christof Ebert, ***A Brief History of Software Technology***, *IEEE Software, November 2008.*[638]

[638] DOI: 10.1109/MS.2008.141

“Today’s **power tools** enable us to **cut code** and **test** our **design ideas** much more quickly. This is a significant improvement. Yet the **more code** we create, the more opportunity we have for it to **grow** unwieldy, inconsistent, and **unmaintainable**. “

---

*Rebecca J. Wirfs-Brock, **Designing Then and Now**, IEEE Software, November 2008.*[639]

[639] DOI: 10.1109/MS.2008.146

"The increasing sophistication and use of **software measurement** over the past 25 years … highlights **four obstacles** to more effective use of measurement: dealing with **uncertainty**, anticipating **change**, measuring '**soft**' characteristics, and developing **heuristics**."

Shari Lawrence Pfleeger, ***Software Metrics: Progress after 25 Years?***, *IEEE Software, November 2008.*[640]

[640] DOI: 10.1109/MS.2008.160

"**Opportunistic software systems development** (OSSD) is an approach in which developers **meld together** software pieces that they have found. Most often they find unrelated software **components** and systems that **weren't designed to work together** but that provide functionality they want to include in a new system. Typically, in opportunistic development, developers spend less effort developing software functionality to meet particular requirements and more time developing '**glue code**' and using other techniques for integrating the various software pieces."

Anatol W. Kark, Cornelius Ncube, Patricia Oberndorf, ***Opportunistic Software Systems Development: Making Systems from What's Available****, IEEE Software, November 2008.*[641]

[641] DOI: 10.1109/MS.2008.153

"Developing products and services pragmatically places requirements on the **relationship** between the software **developer** and the **third-party functionality** provider."

Sjaak Brinkkemper, Slinger Jansen, Cetin Demir, Ivo Hunink, ***Pragmatic and Opportunistic Reuse in Innovative Start-up Companies****, IEEE Software, November 2008.*[642]

[642] DOI: 10.1109/MS.2008.155

"**Situated software**, a type of opportunistic software, is created by a small subset of users to fulfill a specific purpose. For example, business users have been creating situated software through mashups, which combine data from multiple sources on internal systems or the Internet."

Grace A. Lewis, Soumya Simanta, Dennis B. Smith, Sriram Balasubramaniam, ***Situated Software: Concepts, Motivation, Technology, and the Future****, IEEE Software, November 2008.*[643]

[643] DOI: 10.1109/MS.2008.159

"Using **opportunistic software development principles** in **computer engineering education** encourages students to be creative and to develop solutions that cross the boundaries of diverse technologies."

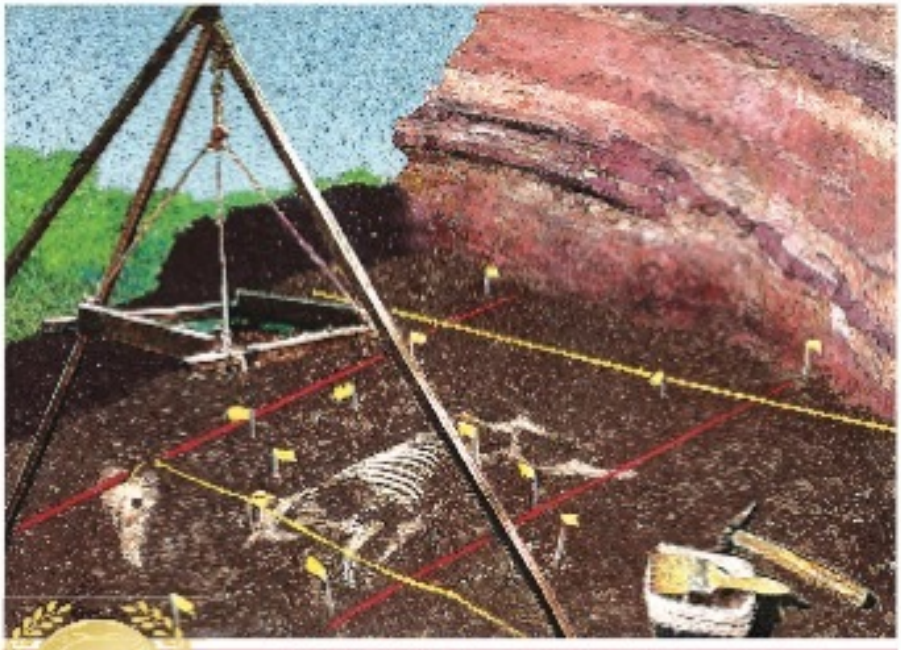
Dragan Gaševic, Željko Obrenovic, Anton Eliëns, ***Stimulating Creativity through Opportunistic Software Development****, IEEE Software, November 2008.*[644]

[644] DOI: 10.1109/MS.2008.162

2009

IEEE
Software

Mining Software Archives



JANUARY | FEBRUARY 2009

IEEE
Celebrating 125 Years of Engineering the Future

IEEE computer society

www.computer.org/software

"Few software practices are as important as testing, and **testing techniques** are amenable to measurement and reasoning about their effectiveness. Because they're aimed at **removing faults**, measuring the number and type of such removed faults seems like a natural part of applying these techniques."

Ana Moreno, Sira Vegas, Natalia Juristo, Forrest Shull, ***A Look at 25 Years of Data****, IEEE Software, January 2009.*[645]

[645] DOI: 10.1109/MS.2009.2

“Becoming a **better designer** means getting better at what we do now while not getting lulled into accepting the status quo. To stay effective as designers, we need to **continue** to **learn**, **adapt**, keep an **open mind**, and work to **perfect our craft**.”

---

*Rebecca J. Wirfs-Brock, **Designing in the Future**, IEEE Software, January 2009.*[646]

[646] DOI: 10.1109/MS.2009.7

"Today, many software projects are **geographically distributed**, so software managers must know how to manage **distributed teams**. For example, they need to know how to build teams across sites, how to break down and distribute tasks, how to share knowledge across time, space, and cultural differences, and how to coordinate work to produce coherent outcomes."

Lars Mathiassen, John Stouby Persson, ***A Process for Managing Risks in Distributed Teams****, IEEE Software, January 2009.*[647]

[647] DOI: 10.1109/MS.2009.157

"Modern programming environments automatically collect lots of data on software development, notably changes and defects. The field of **mining software archives** is concerned with the **automated extraction**, **collection**, and **abstraction** of information from this data."

Andreas Zeller, Nachiappan Nagappan, Thomas Zimmermann, ***Guest Editors' Introduction: Mining Software Archives****, IEEE Software, January 2009.*[648]

[648] DOI: 10.1109/MS.2009.14

"**Software archives** such as source code **version-control systems** and **issue-tracking systems** (for bugs and change requests) are rich sources to examine what changes have what impact on the software."

Harald C. Gall, Martin Pinzger, Beat Fluri, ***Change Analysis with Evolizer and ChangeDistiller****, IEEE Software, January 2009.*[649]

[649] DOI: 10.1109/MS.2009.6

"In 1994, **Standish** published **the Chaos report** that showed a shocking 16 percent project success. This and renewed figures by Standish are often used to indicate that project management of application software development is in trouble. However, **Standish's definitions** have four **major problems**. First, they're **misleading** because they're based solely on estimation accuracy of cost, time, and functionality. Second, their estimation accuracy measure is **one-sided**, leading to unrealistic success rates. Third, **steering** on their definitions **perverts good estimation** practice. Fourth, the resulting figures are **meaningless** because they **average** numbers with an **unknown bias**, numbers that are introduced by **different underlying estimation** processes."

Chris Verhoef, J. Laurenz Eveleens, ***The Rise and Fall of the Chaos Report Figures****, IEEE Software, January 2009.*[650]

[650] DOI: 10.1109/MS.2009.154

"As **software's impact** and influence grows, so do the **possibilities for innovation** and increasing the competitive advantage through software."

Steven Kunsman, Samuel Fricker, Kenneth Palm, Tony Gorschek, ***A Lightweight Innovation Process for Software-Intensive Product Development****, IEEE Software, January 2009.*[651]

[651] DOI: 10.1109/MS.2009.164

"Mining software repositories using **analytics-driven dashboards** provides a **unifying mechanism** for **understanding**, **evaluating**, and **predicting** the development, management, and economics of large-scale systems and processes. Dashboards enable **measurement** and **interactive graphical displays** of complex information and support flexible analytic capabilities for user customizability and extensibility."

Richard W. Selby, ***Analytics-Driven Dashboards Enable Leading Indicators for Requirements and Designs of Large-Scale Systems****, IEEE Software, January 2009.*[652]

[652] DOI: 10.1109/MS.2009.4

"Developers should **factor rework into sizing** and productivity calculations when estimating software effort. **Reworked code** is software created during development that **doesn't exist** in the **final build**. Using lines of code as a sizing metric is helpful when estimating projects with similar domains, platforms, processes, development teams, and coding constraints."

Edmund Morozoff, ***Using a Line-of-Code Metric to Understand Software Rework****, IEEE Software, January 2009.*[653]

[653] DOI: 10.1109/MS.2009.160

“**Software-intensive systems**, like bridges and societies, are subject to **collapse**. Collapse isn’t necessarily inevitable, however, but avoiding it requires active, vigorous, and intentional intervention by the system’s architects.”

---

*Grady Booch, **Not with a Bang**, IEEE Software, January 2009.*[654]

[654] DOI: 10.1109/MS.2009.18

"**Usability** is a growing issue for developers of **scientific software**. Scientists seeking software to support scientific discovery and funding bodies seeking better **return on investment** increase the pressure to produce scientific software that has an impact beyond a limited set of users (that is, scientists in a single lab)."

Jason R. Swedlow, David Sloan, Peter Gregor, Xinyi Jiang, Catriona Macaulay, Paula Forbes, Scott Loynton, ***Usability and User-Centered Design in Scientific Software Development****, IEEE Software, January 2009.*[655]

[655] DOI: 10.1109/MS.2009.27

IEEE
Software

Capturing Design Knowledge

MARCH | APRIL 2009

IEEE Celebrating 125 Years of Engineering the Future
IEEE computer society
www.computer.org/software

"At the core of **cloud computing** is a simple concept: **software as a service**, or SaaS. Whether the underlying software is an **application**, application **component**, **platform**, **framework**, **environment**, or some other soft infrastructure for composing applications to be delivered as a service on the Web, it's all software in the end. But the **simplicity ends** there. Just a step away from that core, a **complex concoction** of paradigms, concepts, and technologies envelop cloud computing."

Hakan Erdogmus, ***Cloud Computing: Does Nirvana Hide behind the Nebula?****, IEEE Software, March 2009.*[656]

[656] DOI: 10.1109/MS.2009.31

“Given that useful knowledge in software engineering has a half-life of about five years, **reading** remains **an excellent way** to replenish this vanishing resource for the diligent software engineer.”

---

*Philippe Kruchten, **You Are What You Read**, IEEE Software, March 2009.*[657]

[657] DOI: 10.1109/MS.2009.55

"**Successful** software-intensive systems are generally quite **innovative**, as evidenced by their success. Although their architectures will, over time, converge to a stable point, this does not mean that innovation stops. Rather, for any such system to remain vibrant and relevant, **innovation must proceed** simultaneously at many levels."

---

*Grady Booch, **The Resting Place of Innovation**, IEEE Software, March 2009.*[658]

[658] DOI: 10.1109/MS.2009.53

"**Capturing** software **design knowledge** is important because it tends to evaporate as software systems evolve. This has severe consequences for many software projects. To counteract this phenomenon, effective, systematic **documentation** of design knowledge is important."

Uwe Zdun, ***Guest Editor's Introduction: Capturing Design Knowledge****, IEEE Software, March 2009.*[659]

[659] DOI: 10.1109/MS.2009.37

"**Large software systems**, developed over several years, are the backbone of industries such as banking, retail, transportation, and telecommunications. With multiple bug fixes and feature enhancements, these systems gradually deviate from the intended architecture and deteriorate into **unmanageable monoliths**."

Madhu K. Iyengar, Saravanan Sivagnanam, K. Rangarajan, Shubha Ramachandran, G. Sathish Kumar, Santonu Sarkar, ***Modularization of a Large-Scale Business Application: A Case Study****, IEEE Software, March 2009.*[660]

[660] DOI: 10.1109/MS.2009.42

"**A decision view** provides a useful complement to the traditional sets of architectural views and viewpoints. It gives an explanatory perspective that illuminates the **reasoning process** itself and not solely its results. The decision view documents **aspects** of the architecture that are **hard to reverse-engineer** from the software itself and that are often left tacit."

Philippe Kruchten, Juan Carlos Dueñas, Rafael Capilla, ***The Decision View's Role in Software Architecture Practice****, IEEE Software, March 2009.*[661]

[661] DOI: 10.1109/MS.2009.52

"We rarely see the **traditional way** of software development in which one company handles design, production, sales, delivery, and service. Business models, engineering life cycles, distribution channels, and services have changed dramatically. A key driver in these new value networks is **open source software** (OSS). Worldwide companies in various industries have invested in open source. Market leaders such as Google, IBM, Microsoft, SAP, and Siemens as well as many small companies turn to OSS for multiple reasons."

Christof Ebert, ***Guest Editor's Introduction: How Open Source Tools Can Benefit Industry****, IEEE Software, March 2009.*[662]

[662] DOI: 10.1109/MS.2009.38

"A **software forge** is a tool platform for **collaborative software development**, similar to integrated CASE environments. Unlike CASE tools, however, software forges have been designed for the software development practices of the **open source community**."

Dirk Riehle, Tamir Menahem, Barak Naveh, John Ellenberger, Boris Mikhailovski, Yuri Natchetoi, Thomas Odenwald, ***Open Collaboration within Corporations Using Software Forges****, IEEE Software, March 2009.*[663]

[663] DOI: 10.1109/MS.2009.44

“Developing **complex software** can be difficult no matter how good designers get at architecture, tooling, or technology. Although agile techniques and practices vary, successful agile designers I know are **passionate about** producing **high-quality** incremental design solutions. “

---

*Rebecca J. Wirfs-Brock, **Designing with an Agile Attitude**, IEEE Software, March 2009.*[664]

[664] DOI: 10.1109/MS.2009.32

"One way to combine **rigor** and **relevance** in research might be to perform applicability checks, in which focus groups provide feedback on research projects."

Robert L. Glass, ***Making Research More Relevant While Not Diminishing Its Rigor***, *IEEE Software*, *March 2009.*[665]

[665] DOI: 10.1109/MS.2009.40


IEEE
Software
Domain-Specific Modeling
JULY | AUGUST 2009
4 | The Seven Traits of Superprofessionals
62 | Meyer's Seven Test Principles Debated
66 | Two Classic Articles Revisited
IEEE
Celebrating 125 Years of Engineering the Future
IEEE computer society
www.computer.org/software

"**Superprofessionalism** is a mode of conduct characterized by seven central traits: focus on **individual responsibility**, acute **awareness**, commitment to **facts**, **resilience** under pressure, sense of **fairness**, attention to detail in **perspective**, and **pragmatism** first."

Hakan Erdogmus, ***The Seven Traits of Superprofessionals****, IEEE Software, July 2009.*[666]

[666] DOI: 10.1109/MS.2009.107

"**Simple architectures** have **conceptual integrity** and are better than more complex ones. Continuous architectural refactoring helps to converge a system to its practical and optimal simplicity."

Grady Booch, ***The Defenestration of Superfluous Architectural Accoutrements***, *IEEE Software, July 2009.*[667]

[667] DOI: 10.1109/MS.2009.105

“Compliance to a professional society’s **code of ethics** carries obligations beyond minimum standards of behavior. Members of software engineering professional societies should also serve the **public interest** and promote the common good.”

Duncan Hall, ***The Ethical Software Engineer***, *IEEE Software*, *July 2009.*[668]

[668] DOI: 10.1109/MS.2009.106

"A **framework** for **thinking about** domain-specific languages (**DSLs**) divides them into **internal DSLs**, **external DSLs**, and **language workbenches**. In all cases, it's important to have an explicit semantic model so that they form a veneer over an underlying library. DSLs are valuable for increasing programmer **productivity** and improving **communication** with domain experts."

Martin Fowler, ***A Pedagogical Framework for Domain-Specific Languages****, IEEE Software, July 2009.*[669]

[669] DOI: 10.1109/MS.2009.85

"**Domain-specific techniques** provide a high-level specification for software systems. The technology's foundations have been developed over the last few years. However, domain-specific techniques **aren't a panacea**, and deciding whether investment in them is merited is an important step in understanding their benefits."

Jonathan Sprinkle, Diomidis Spinellis, Juha-Pekka Tolvanen, Marjan Mernik, ***Guest Editors' Introduction: What Kinds of Nails Need a Domain-Specific Hammer?****, IEEE Software, July 2009.*[670]

[670] DOI: 10.1109/MS.2009.92

"Maintenance in software-intensive systems is critical because software often continuously evolves not only during development but also after delivery, to meet users' ever-changing needs. So, maintenance performance significantly impacts software development productivity."

Lan Cao, Matti Rossi, Balasubramaniam Ramesh, ***Are Domain-Specific Models Easier to Maintain Than UML Models?****, IEEE Software, July 2009.*[671]

[671] DOI: 10.1109/MS.2009.87

"The single largest factor that led to a **language not being** used was when organizations gave the language design task to someone with **insufficient experience** in the **problem domain**."

Steven Kelly, Risto Pohjonen, ***Worst Practices for Domain-Specific Modeling****, IEEE Software, July 2009.*[672]

[672] DOI: 10.1109/MS.2009.109

"**Reusing DSLs** is **hard**, however, because they're often designed to precisely describe a single domain or concern. A new approach uses techniques from **software product lines** (SPLs) to improve the reusability of a DSL, DSL composition, or supporting tool by providing traceability of language concepts to DSL design."

Sumant Tambe, Jules White, Jeff Gray, James H. Hill, Aniruddha S. Gokhale, Douglas C. Schmidt, ***Improving Domain-Specific Language Reuse with Software Product Line Techniques****, IEEE Software, July 2009.*[673]

[673] DOI: 10.1109/MS.2009.95

"Although **lessons-learned activities** aid software process improvement, **few** IT industry **organizations** regularly and adequately **conduct them**."

Anders Baaz, Anna Sandberg, Agneta Nilsson, Lena Holmberg, Helena Olsson, ***Appreciating Lessons Learned****, IEEE Software, July 2009.*[674]

[674] DOI: 10.1109/MS.2009.198

"For the past 40 years… we've **tortured ourselves** over our **inability** to finish a software project **on time** and **on budget**. But … this **never** should have been the **supreme goal**. The **more important** goal is transformation, creating **software that changes the world** or that transforms a company or how it does business. We've been rather **successful at transformation**, often while operating outside our control envelope. Software development is and always will be somewhat **experimental**. The actual software construction isn't necessarily experimental, but its **conception is**. And this is where our focus ought to be. It's where our focus always ought to have been."

Tom DeMarco, ***Software Engineering: An Idea Whose Time Has Come and Gone?****, IEEE Software, July 2009.*[675]

[675] DOI: 10.1109/MS.2009.101

IEEE
Software

Software Development Tools

SEPTEMBER/OCTOBER 2008

IEEE computer society

www.computer.org/software

"The review results suggest that despite some limitations, **agile development** can improve **job satisfaction**, project **productivity**, and **customer satisfaction**."

Tore Dybâ, Torgeir Dingsøyr, ***What Do We Know about Agile Software Development?****, IEEE Software, September 2009.*[676]

[676] DOI: 10.1109/MS.2009.145

“**Millions of people** program to support their work but don’t call themselves programmers. The field of **end-user software engineering** is concerned with helping these people create reliable, dependable, and reusable programs, without distracting them from their primary tasks. “

Brad A. Myers, Robin Abraham, Andrew J. Ko, Margaret M. Burnett, ***Guest Editors’ Introduction: End-User Software Engineering****, IEEE Software, September 2009.*[677]

[677] DOI: 10.1109/MS.2009.129

"People often **write code** to **prototype**, **ideate**, and **discover**. To do this, they **work opportunistically**, emphasizing speed and ease of development over code robustness and maintainability. Quickly hacking a program together can provide both practical and **learning benefits** for novices and experts: professional programmers and designers prototype to **explore and communicate** ideas, scientists program laboratory instruments, and entrepreneurs assemble complex spreadsheets to **better understand** their business."

Joel Brandt, Philip J. Guo, Joel Lewenstein, and Scott R. Klemmer, Mira Dontcheva, ***Writing Code to Prototype, Ideate, and Discover****, IEEE Software, September 2009.*[678]

[678] DOI: 10.1109/MS.2009.147

"**Spreadsheets** are popular end-user programming tools. Many people use spreadsheet-computed values to make **critical decisions**, so spreadsheets must be correct. Proven software engineering principles can assist the construction and maintenance of dependable spreadsheets."

Martin Erwig, ***Software Engineering for Spreadsheets****, IEEE Software, September 2009.*[679]

[679] DOI: 10.1109/MS.2009.140

“**Spreadsheet technology** is central to the functioning of the **financial sector**, but professionally created spreadsheets have a **high level of error**, which highlights the need for innovative supporting processes and tools. The current global financial crisis will likely accelerate this need because anticipated regulation will require novel, innovative risk management methods and technologies that cover development, risk assessment, review, and other spreadsheet activities. “

Alan Rust, Kevin McDaid, ***Test-Driven Development for Spreadsheet Risk Management****, IEEE Software, September 2009.*[680]

[680] DOI: 10.1109/MS.2009.143

“Using **Selenium**, Web **acceptance-test designers** have a tool that programmatically reflects **business structure** much better than **protocol-level tools would**, while avoiding the pitfalls of capture-and-replay tools.”

Andreas Bruns, Dennis Wichmann, Andreas Kornstädt, ***Web Application Tests with Selenium****, IEEE Software, September 2009.*[681]

[681] DOI: 10.1109/MS.2009.144

IEEE
Software

Human Aspects of Software Engineering

NOVEMBER | DECEMBER 2009

IEEE
Celebrating 125 Years of Engineering the Future

IEEE computer society

www.computer.org/software

“Software developers are notorious for **skimping** on **design documentation**, often eschewing it altogether. This trend has led to claims that it is merely an **impediment** in the **fast-paced** and highly pliable world of software development - a useless vestige of **old-style engineering** that should be eliminated altogether. … because of the **complexity** of modern software systems and the **cryptic nature** of current programming **languages**, **good design documentation** is not only useful but vital. However, we must seek ways of adapting it to suit the **medium** as well as the **exceptionally dynamic development** process.”

---

*Bran Selic, **Agile Documentation, Anyone?**, IEEE Software, November 2009.*[682]

[682] DOI: 10.1109/MS.2009.167

"Software is developed by people, used by people, and supports interaction among people. As such, **human characteristics** and **cooperation** are central to modern practical software construction."

Janice Singer, Cleidson R. B. de Souza, Helen Sharp, Gina Venolia, Li-Te Cheng, ***Guest Editors' Introduction: Cooperative and Human Aspects of Software Engineering****, IEEE Software, November 2009.*[683]

[683] DOI: 10.1109/MS.2009.176

“In the transformation from **traditional command-and-control** management to **collaborative self-managing** teams, the main challenges were the absence of **redundancy and conflict** between team- and individual-level autonomy.”

Tore Dybâ, Torgeir Dingsøyr, Nils Brede Moe, ***Overcoming Barriers to Self-Management in Software Teams****, IEEE Software, November 2009.*[684]

[684] DOI: 10.1109/MS.2009.182

“Data showed that **learning resources** for **APIs** are critically important and shed light on three issues: the need to **discover** the **design** and **rationale** of the API when needed, the challenge of finding **credible** usage API **examples** at the right level of complexity, and the challenge of understanding **inexplicable** API **behavior**.”

Martin P. Robillard, ***What Makes APIs Hard to Learn? Answers from Developers****, IEEE Software, November 2009.*[685]

[685] DOI: 10.1109/MS.2009.193

"**Small to medium enterprises** (SMEs) increasingly participate in **offshore software development**. Key competitive SME abilities include **detecting market niches** and deploying highly flexible software development approaches. Therefore, learning how offshoring affects such capabilities, which are closely related to organizational learning, is crucial."

Volker Wulf, Alexander Boden, Bernhard Nett, ***Operational and Strategic Learning in Global Software Development****, IEEE Software, November 2009.*[686]

[686] DOI: 10.1109/MS.2009.113

"Our aspirations grow faster than our capabilities, so **I don't expect software development to 'get solved.'**"

Mary Shaw, ***Continuing Prospects for an Engineering Discipline of Software****, IEEE Software, November 2009.*[687]

[687] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2009.172

"Advice on **courteousness** and **politeness** in **technical communication** is in short supply, yet this is needed when developers communicate with other people. When discussing technical problems, aim to encourage rather than complain, focusing on technology issues rather than the people behind them. Every **email** should tackle **one topic** and that topic should be **the subject line**."

Diomidis Spinellis, ***Basic Etiquette of Technical Communication****, IEEE Software, November 2009.*[688]

[688] DOI: 10.1109/MS.2009.170

“**Process metrics** can ignite **strong opinions** because they represent an area where **technical considerations** bump up against **human aspects** of software development. “

Forrest Shull, Medha Umarji, ***Measuring Developers: Aligning Perspectives and Other Best Practices****, IEEE Software, November 2009.*[689]

[689] DOI: 10.1109/MS.2009.180

"In his final **Loyal Opposition** column for IEEE Software, **Robert Glass** points out the continued need for **testing academic theories** in practice and for practitioners to **discuss lessons learned**. He also states that **software estimation** is a **deeply flawed** activity and that software practitioners should always remain open-minded."

---

*Robert L. Glass, **Goodbye!**, IEEE Software, November 2009.*[690]

[690] DOI: 10.1109/MS.2009.175

2010

IEEE
Software

Project Management

JANUARY | FEBRUARY 2010

IEEE computer society

www.computer.org/software

"The **software project management** body of knowledge is gradually being renewed across the entire lifecycle. In the **conception phase**, the focus is on **fostering innovation** through new approaches such as business analysis and crowdsourcing techniques. In the **construction phase**, the rise of **global software development** has shrunk the world and led to new approaches to risk management that take into account factors such as cultural diversity. In the **project conclusion** phase, new approaches to the evaluation of project success are being introduced, such as project **retrospectives** and **intellectual capital reporting**."

John Favaro, ***Guest Editor's Introduction: Renewing the Software Project Management Life Cycle****, IEEE Software, January 2010.*[691]

[691] DOI: 10.1109/MS.2010.9

"Developing parallel applications is notoriously difficult, but it's even more complex for desktop applications. The added difficulties primarily come from their interactive nature, where users largely perceive their performance. Desktop applications are typically developed with **graphical toolkits** that in turn have **limitations** in regards to **multithreading**."

Oliver Sinnen, Nasser Giacaman, ***Object-Oriented Parallelisation of Java Desktop Programs****, IEEE Software, January 2010.*[692]

[692] DOI: 10.1109/MS.2010.135

"If you're **successful**, stakeholders will **trust you**. However, **too much trust** can be **dangerous**. Stakeholders can become **overdependent** on your guidance, especially if they **lack the knowledge** to specify requirements for the new system. "

Neil Maiden, ***Trust Me, I'm an Analyst****, IEEE Software, January 2010.*[693]

[693] DOI: 10.1109/MS.2010.22

"Desktop software developers' interest in **graphics hardware** is increasing as a result of modern graphics cards' capabilities to act as compute devices that augment the **main processor**. This capability means parallel computing is no longer a dedicated task for the CPU. A trend toward heterogeneous computing combines the **main processor** and **graphics processing unit** (GPU)."

Frank Feinbube, Peter Troger, Andreas Polze, ***Joint Forces: From Multithreaded Programming to GPU Computing****, IEEE Software, January 2010.*[694]

[694] DOI: 10.1109/MS.2010.134

“The drive to rapidly develop layered, interconnected, and flexible systems has eclipsed consideration of resource costs. Consequently, **large Java applications** suffer from **runtime bloat**: a large and pervasive **infrastructure tax**, where simple transactions require a few hundred thousand method calls, and a server with 1 Gbyte of memory sometimes can only support a few hundred users.”

Edith Schonberg, Nick Mitchell, Gary Sevitsky, ***Four Trends Leading to Java Runtime Bloat****, IEEE Software, January 2010.*[695]

[695] DOI: 10.1109/MS.2010.7

"**Mobile devices** are increasingly accepted as suitable media for multimedia-rich applications. … the most popular development platform options … **Java ME**, **.NET Compact Framework**, **Flash Lite**, and **Android**."

Damianos Gavalas, Daphne Economou, ***Development Platforms for Mobile Applications: Status and Trends****, IEEE Software, January 2010.*[696]

[696] DOI: 10.1109/MS.2010.155

"Modern distributed software systems involve **dynamic operating conditions** that pose engineering challenges to traditional offline design. **Multiagent systems** engineering can solve some of these problems by offering self-adaptive features such as **loose coupling**, **context sensitivity**, and **robustness to failure**."

Michael Georgeff, Danny Weyns, ***Self-Adaptation Using Multiagent Systems****, IEEE Software, January 2010.*[697]

[697] DOI: 10.1109/MS.2010.18

"**Architecture** is just a collective hunch, **a shared hallucination**, an assertion by a set of stakeholders on the nature of their observable world, be it a world that is or a world as they wish it to be."

---

*Grady Booch, **Architecture as a Shared Hallucination**, IEEE Software, January 2010.*[698]

[698] DOI: 10.1109/MS.2010.4

IEEE
Software

Agility and Architecture

MARCH | APRIL 2010

IEEE
IEEE computer society

www.computer.org/software

“Railroad tracks offer guidance and support. There are various tools that can give our software the same handling. The main tool for guiding the code’s direction is the **language’s type system**. For values, the type system can help us by establishing a separate type for each distinct class; for code, interfaces and abstract classes ensure that we won’t forget some crucial methods when we add functionality through a new class. With **domain-specific languages** or even suitably initialized data structures we can efficiently express exactly what the designer intended and nothing more. At a higher level, **architectures** that enforce a particular open-ended but well-defined interface will also guide a software’s progress. Finally, the most flexible track-laying approach is a **tool-supported** software development process.”

Diomidis Spinellis, ***Software Tracks***, *IEEE Software*, *March 2010.*[699]

[699] DOI: 10.1109/MS.2010.56

"In a typical **client-server scenario**, a server provides valuable services to client applications that run remotely on **untrusted client computers**. Typical examples are **video on demand**, online games, **voice-over-IP** communications, and many others. However, **client-side users** often hold **administrative privileges** on their machines and could tamper with the client application to fulfill the service in violation of the service usage conditions or service agreements. **Guaranteeing client-code security** is one of the most difficult security problem to address."

Paolo Tonella, Mariano Ceccato, ***CodeBender: Remote Software Protection Using Orthogonal Replacement****, IEEE Software, March 2010.*[700]

[700] DOI: 10.1109/MS.2010.158

"**Providing architecture** as a **service to application developers**. The approach is an effective way to implement the architecture process especially, but not only, in the context of agile development. In their role as stakeholders of nonfunctional system qualities, architects prepare and support developers by **participating** in **coding activities** and play a key role in **communicating** the architecture throughout the project's lifetime."

Roland Faber, ***Architects as Service Providers****, IEEE Software, March 2010.*[701]

[701] DOI: 10.1109/MS.2010.25

"**Agile development** delivers value quickly, using a series of short-term goals based on **immediate priorities**. **Architecture** grows value carefully, using a set of long-term objectives based on fundamental principles. The two seem at odds, but the architect can bring them together at four well-defined points in agile projects: during **project initiation** by setting architectural direction, through **storyboarding** by introducing specific architectural tasks, within sprints by close **collaboration on challenging issues**, and as working software gets delivered by performing **direct inspection**."

James Madison, ***Agile Architecture Interactions****, IEEE Software, March 2010.*[702]

[702] DOI: 10.1109/MS.2010.27

"**Unintentionally violating** open source software **(OSS) licenses** by reusing OSS code is a serious problem for both software companies and OSS developers. The simplest intuitive way to identify such reuse is to **measure code clones** - duplicated code fragments - between a suspected program and an existing OSS program."

Yuki Manabe, Satoshi Okahara, Kenichi Matsumoto, Akito Monden, ***Guilty or Not Guilty: Using Clone Metrics to Determine Open Source Licensing Violations****, IEEE Software, March 2010.*[703]

[703] DOI: 10.1109/MS.2010.159

"The evolution of **software into services** imposes certain concerns in the form of expressing and accessing services. The seamless proliferation of services demands a **new kind** of **software protection** with respect to **copyrights** and **moral rights** of service-based software to enable services' widespread use."

G.R. Gangadharan, Vincenzo D'Andrea, ***Managing Copyrights and Moral Rights of Service-Based Software****, IEEE Software, March 2010.*[704]

[704] DOI: 10.1109/MS.2010.161

"**Design tactics** are a methodology architects can use to master this challenge: choosing design solutions that are **simple**, **economic**, and **appropriate** for resolving the problems at hand."

Frank Buschmann, ***Learning from Failure, Part III: On Hammers and Nails, and Falling in Love with Technology and Design****, IEEE Software, March 2010.*[705]

[705] DOI: 10.1109/MS.2010.47

"How can an **organization transition** from several **functionally overlapping systems** to just one? This scenario is common after, for example, **company acquisitions** and mergers or as a result of different units in an organization growing to the point at which the two independent efforts must be synchronized to continue."

Ivica Crnkovi, Rikard Land, ***Oh Dear, We Bought Our Competitor: Integrating Similar Software Systems****, IEEE Software, March 2010.*[706]

[706] DOI: 10.1109/MS.2010.86

"**Enterprise architecture** and **technical architecture** are related yet **different**: whereas EA focuses on the architecture of a **business** that uses **software-intensive systems**, TA focuses on the architecture of the **software-intensive systems** that are used by a business to makes its mission manifest."

Grady Booch, ***Enterprise Architecture and Technical Architecture***, *IEEE Software, March 2010.*[707]

[707] DOI: 10.1109/MS.2010.42

IEEE
Software

Software Product Lines

MAY | JUNE 2010

IEEE computer society

www.computer.org/software

"**Web 2.0** is less a **new technology** than a **new way of using technology**."

José Manuel Torres, Nicolás Serrano, ***Web 2.0 for Practitioners****, IEEE Software, May 2010.*[708]

[708] DOI: 10.1109/MS.2010.84

"Using **software product lines** to create a shared set of features can increase productivity and reduce costs for organizations. Successful software product lines share certain **commonalities** but also differ in certain ways, depending on diverse aspects of the products and the product lines themselves."

Kentaro Yoshimura, Paul Jensen, Dirk Muthig, John D. McGregor, ***Guest Editors' Introduction: Successful Software Product Line Practices****, IEEE Software, May 2010.*[709]

[709] DOI: 10.1109/MS.2010.74

"We characterize two **strategic pitfalls** that repeatedly occur: failure to recognize that a software product line approach is **a business and technical strategy**, and failure to **manage the unique aspects** of governance for a product line and roll it out appropriately."

Linda M. Northrop, Lawrence G. Jones, ***Clearing the Way for Software Product Line Success****, IEEE Software, May 2010.*[710]

[710] DOI: 10.1109/MS.2010.71

"**Successful product lines** suffer over time from **increasing dependencies** between the software assets that make up the product line and, consequently, the teams associated with these assets. This results in high **coordination cost**, **slow release** cycles, and high system-level **error density**."

Jan Bosch, ***Toward Compositional Software Product Lines****, IEEE Software, May 2010.*[711]

[711] DOI: 10.1109/MS.2010.32

"**Product line scoping** is the process of determining which of an organization's products, features, and domains would find **systematic reuse** economically useful."

Isabel John, ***Using Documentation for Product Line Scoping****, IEEE Software, May 2010.*[712]

[712] DOI: 10.1109/MS.2010.34

"**JavaScript** is often seen as a **toy language**. Yet, it offers a **powerful mix** of interesting **language features** based on **functional programming**, **prototyping**, and **mutable objects**. **Web 2.0** apps use JavaScript extensively to realize sophisticated client-side functionality. Taken this into account, it isn't surprising that JavaScript made it to the **top 10** in a survey on the most popular programming languages. "

Holger M. Kienle, ***It's About Time to Take JavaScript (More) Seriously****, IEEE Software, May 2010.*[713]

[713] DOI: 10.1109/MS.2010.76

“**Refactoring** was originally conceived as a technique for enhancing the design of an existing code base by applying small **behavior-preserving transformations** to the code. … discuss how to **improve the usability** of a Web application by applying **refactoring** on its **design structure**. “

Gustavo Rossi, Alejandra Garrido, Damiano Distante, ***Refactoring for Usability in Web Applications****, IEEE Software, May 2010.*[714]

[714] DOI: 10.1109/MS.2010.114

"What are the top five properties that make a **software design elegant**? … we explore … five properties leading architects have found useful: **economy**, **visibility**, **spacing**, **symmetry**, and **emergence**."

Frank Buschmann, Kevlin Henney, ***Five Considerations for Software Architecture, Part 1****, IEEE Software, May 2010.*[715]

[715] DOI: 10.1109/MS.2010.72

“**Developing concurrent software** is hard. **Testing** concurrent software is **harder**. “

Sebastian Burckhardt, Madan Musuvathi, Shaz Qadeer, Peli de Halleux, Thomas Ball, ***Predictable and Progressive Testing of Multithreaded Code****, IEEE Software, May 2010.*[716]

[716] DOI: 10.1109/MS.2010.64

"An **architectural review** serves several purposes: to gain confidence in the design, to reason about alternatives, to attend to architectural rot. The process of such a review involves the interplay of **design decisions**, **scenari**os, and **forces** on the system."

---

*Grady Booch, **Architecture Reviews**, IEEE Software, May 2010.*[717]

[717] DOI: 10.1109/MS.2010.68

IEEE
Software

Software Evolution

JULY | AUGUST 2010

IEEE computer society

www.computer.org/software

"The **cloud-computing paradigm** is characterized by **transactional** resource **acquisition** … **nonfederated** resource **provisioning** … a **metered resource**."

Panos Louridas, ***Up in the Air: Moving Your Applications to the Cloud***, *IEEE Software, July 2010.*[718]

[718] DOI: 10.1109/MS.2010.109

"**Evolving** and **maintaining** software-intensive systems is critical, and, consequently, **most developers** are involved in maintaining, incrementally enhancing, and adapting existing systems."

Yann-Gaël Guehénéuc, Maja D'Hondt, Juan Fernández-Ramil, Tom Mens, ***Guest Editors' Introduction: Software Evolution****, IEEE Software, July 2010.*[719]

[719] DOI: 10.1109/MS.2010.100

"Despite growth in the popularity of desktop systems, Web applications, and mobile computing, **mainframe systems** remain the **dominant force** in large-scale enterprise computing. Although they're sometimes referred to as '**the dinosaurs of computing**,' even mainframe systems must adapt to changing circumstances to survive."

Serge Demeyer, Joris Van Geet, ***Reverse Engineering on the Mainframe: Lessons Learned from 'In Vivo' Research****, IEEE Software, July 2010.*[720]

[720] DOI: 10.1109/MS.2010.65

"**Architecture evaluations** offer many benefits, including the early detection of problems and a better understanding of a system's possibilities. … the **lightweight sanity check** for implemented architectures (LiSCIA) evaluation method … can be used out of the box to perform a first architectural evaluation of a system. … By **periodically performing** this check, developers and project managers can **control** the implemented **architecture's erosion** as the system (and its requirements) evolves over time."

Eric Bouwers, Arie van Deursen, ***A Lightweight Sanity Check for Implemented Architectures****, IEEE Software, July 2010.*[721]

[721] DOI: 10.1109/MS.2010.60

"**Software** is key to commercial **magnetic resonance imaging (MRI)** scanners, the medical devices that make images of the living human body for clinical purposes."

Joop van der Linden, Lennart Hofland, ***Software in MRI Scanners****, IEEE Software, July 2010.*[722]

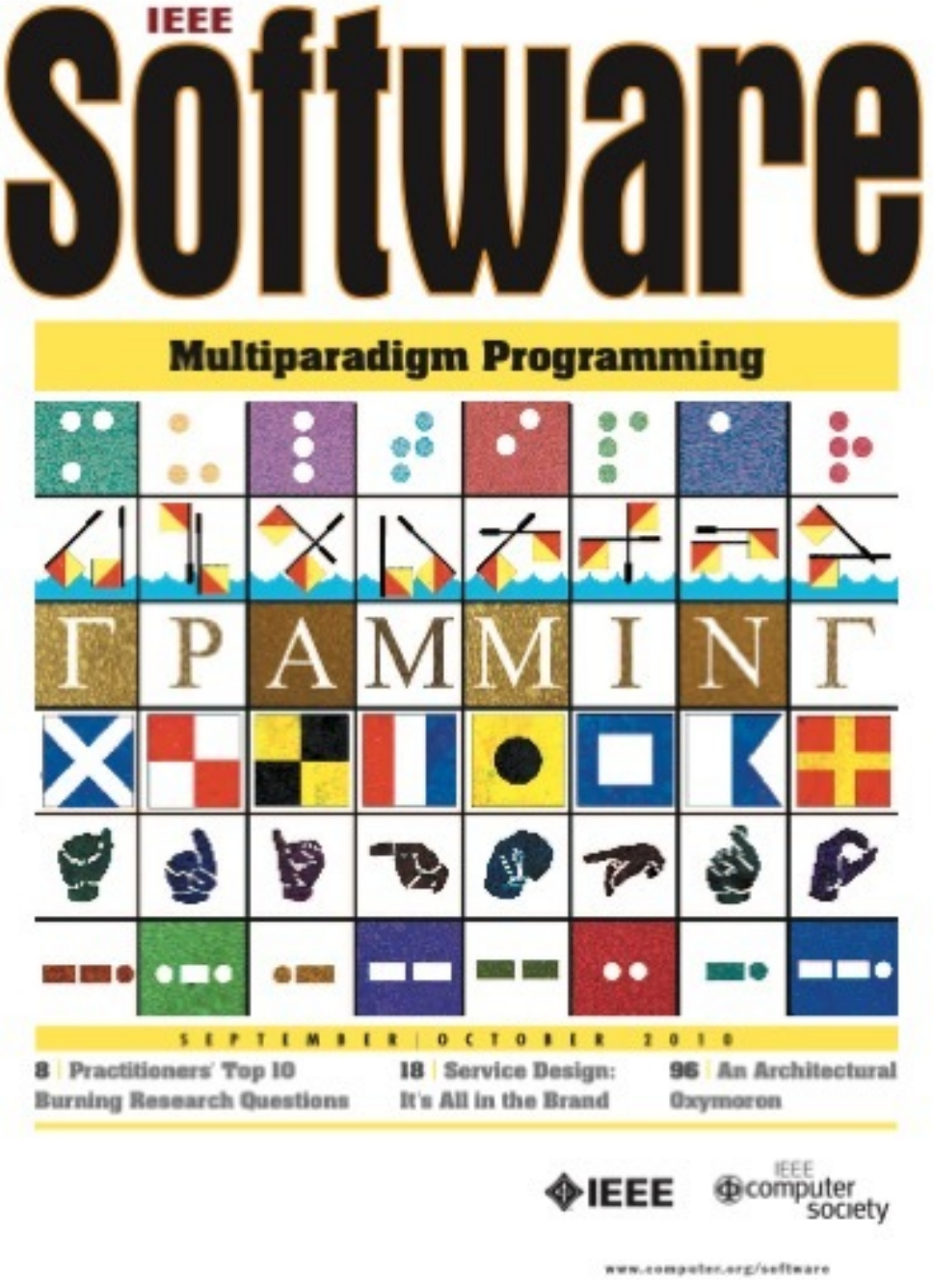
[722] DOI: 10.1109/MS.2010.106

"All complex **systems fail**, by some measure of the word 'fail,' with consequences ranging from **benign** to **catastrophic**."

---

*Grady Booch, **Systems Architecture**, IEEE Software, July 2010.*[723]

[723] DOI: 10.1109/MS.2010.107


IEEE
Software
Multiparadigm Programming
ΓΡΑΜΜΙΝΓ
SEPTEMBER | OCTOBER 2010
8 | Practitioners' Top 10 Burning Research Questions
18 | Service Design: It's All in the Brand
96 | An Architectural Oxymoron
IEEE
IEEE computer society
www.computer.org/software

"The top questions were Agile and **large projects**. What factors can **break self-organization**? Do teams really need to always be **collocated** to collaborate effectively? **Architecture and agile**—how much design is enough for different classes of problem? Hard facts on **costs of distribution** (in $, £, €, and so on). The correlation between **release length** and **success rate**. **What metrics** can we use with minimal **side-effects**? Distributed agile and **trust**—what happens around 8–12 weeks? Statistics and data about how much **money/time is saved** by agile. **Sociological studies**—what were the personalities in successful/failed agile teams?"

Helen Sharp, Sallyann Freudenberg, ***The Top 10 Burning Research Questions from Practitioners****, IEEE Software, September 2010.*[724]

[724] DOI: 10.1109/MS.2010.129

“A collection of **coherent**, often **ideologically** or **theoretically** based **abstractions** constitutes a **programming paradigm**. … Well-known examples include **object-oriented**, **relational**, **functional**, **constraint-based**, **theorem-proving**, **concurrent**, **imperative**, and **declarative**. Less well-known (or perhaps less well-defined) examples include **graphical**, **reflective**, **context-aware**, **rule-based**, and **agent-oriented**. … using just **one language** technology and paradigm is becoming much **less common**, replaced by **multiparadigm programming** in which the heterogeneous application consists of several subcomponents, each implemented with an appropriate paradigm and able to communicate with other subcomponents implemented with a different paradigm. “

Dean Wampler, Tony Clark, ***Guest Editors' Introduction: Multiparadigm Programming****, IEEE Software, September 2010.*[725]

[725] DOI: 10.1109/MS.2010.119

"**Combining paradigms** offers important benefits—for example, OOP minimizes the conceptual gap between the problem domain and the implementation in software, and functional programming (FP) brings mathematical rigor and robustness to computing, especially for concurrent applications."

Dean Wampler, Tony Clark, ***Guest Editors' Introduction: Multiparadigm Programming****, IEEE Software, September 2010.*[726]

[726] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2010.119

"**Domain-specific languages** (DSLs) are becoming a mature application development tool that developers use to express concerns. **Multi-DSL** applications comprise DSLs and host language code. Exploiting the **Ruby** programming language's **built-in support** for the **imperative**, **functional**, and **object-oriented** paradigm, extended with **feature-oriented** programming …"

Sebastian Günther, ***Multi-DSL Applications with Ruby****, IEEE Software, September 2010.*[727]

[727] DOI: 10.1109/MS.2010.91

"**Constraint programming** (CP) is a young but rapidly developing technology that supports the modeling and solution of a wide range of **planning**, **scheduling**, **search**, and **optimization** problems. The integration of CP concepts into languages from other paradigms yields constraint-based multiparadigm programming."

Petra Hofstedt, ***Constraint-Based Object-Oriented Programming****, IEEE Software, September 2010.*[728]

[728] DOI: 10.1109/MS.2010.89

"**Storing data** the same **way it's used** in the application would simplify the programming model, making it easier to decentralize data processing and, in turn, enable horizontal scaling. Emerging **NoSQL** data-storage engines support this strategy. Just like the application layer, the data-storage layer can use multiple paradigms and store data in a way that's **semantically closer** to the corresponding domain models."

Debasish Ghosh, ***Multiparadigm Data Storage for Enterprise Applications****, IEEE Software, September 2010.*[729]

[729] DOI: 10.1109/MS.2010.87

“To some, the phrase ‘**agile architecture**’ is an **oxymoron**. “

---

*Grady Booch, **An Architectural Oxymoron**, IEEE Software, September 2010.*[730]

[730] DOI: 10.1109/MS.2010.117

IEEE
Software

Framing Stakeholders' Concerns

NOVEMBER | DECEMBER 2010

IEEE computer society

www.computer.org/software

"Techniques for **requirements acquisition** must find new ways to gather information about **brands** and **emotional responses** to them. Consumers will also likely have new types of service requirements that must be captured, documented, and easily traceable via new multidisciplinary techniques. … use **storyboards** to capture the interplay between **human interaction** and **service design** and so improve the quality of **service design** delivery."

Malcolm Sutherland, Neil Maiden, ***Storyboarding Requirements****, IEEE Software, November 2010.*[731]

[731] DOI: 10.1109/MS.2010.147

"It has long been recognized that one of the key **benefits of architecting** our systems is **managing their complexity**. This complexity arises from **many factors**: the needs and constraints of the multitude of **system stakeholders** … the **political**, **social**, and other factors from the **environment** in which the system is embedded; the realities and constraints of the system's **development**, **implementation**, **maintenance**, and **operation** in relation to available resources; and, of course, the intended properties of the system itself. Taken together, these diverse interests are a system's **stakeholder concerns**. "

Rich Hilliard, Paris Avgeriou, Patricia Lago, ***Guest Editors' Introduction: Software Architecture: Framing Stakeholders' Concerns****, IEEE Software, November 2010.*[732]

[732] DOI: 10.1109/MS.2010.142

"Architectures come about through forces and needs other than those captured in traditional requirements documents. **A business goal** expresses **why** a system is being developed and what stakeholders in the developing organization, the customer organization, and beyond aspire to achieve through its production and use."

Len Bass, Paul Clements, ***The Business Goals Viewpoint****, IEEE Software, November 2010.*[733]

[733] DOI: 10.1109/MS.2010.116

"The **initial presentation** of **new open source software projects** plays a potentially critical role in **attracting developers**."

Namjoo Choi, Indushobha Chengalur-Smith, Andrew Whitmore, ***Managing First Impressions of New Open Source Software Projects****, IEEE Software, November 2010.*[734]

[734] DOI: 10.1109/MS.2010.26

"The **architecture** of a software-intensive system is best **reasoned** about through multiple, nearly **independent views.**"

Grady Booch, ***The Elephant and the Blind Programmers****, IEEE Software, November 2010.*[735]

[735] DOI: 10.1109/MS.2010.149

2011

JANUARY/FEBRUARY 2011 WWW.COMPUTER.ORG/SOFTWARE

IEEE Software

Parallelism on the Desktop

IEEE computer society

"**Watts Humphrey** had a truly remarkable career, during which he developed or contributed to the **Personal Software Process**, **Team Software Process**, and **Capability Maturity Model Integration** (CMMI) framework, among many other contributions."

Forrest Shull, ***Watts Humphrey: 4 July 1927 - 28 October 2010****, IEEE Software, January 2011.*[736]

[736] DOI: 10.1109/MS.2011.21

"**Developers**, for the most part, **don't draw diagrams** because diagrams all too often don't offer any fundamental value that advances essential work. Yet, the problem remains that **we must visualize** ultra-large complex systems that have no directly observable physical manifestation."

---

*Grady Booch, **Draw Me a Picture**, IEEE Software, January 2011.*[737]

[737] DOI: 10.1109/MS.2011.4

"The computer industry is experiencing a major shift: **improved single processor performance** via **higher clock rates** has reached its technical limits due to overheating. … exploiting the **full potential** of these processors requires **parallel programming**."

Kurt Keutzer, Wolfram Schulte, Victor Pankratius, ***Guest Editors' Introduction: Parallelism on the Desktop****, IEEE Software, January 2011.*[738]

[738] DOI: 10.1109/MS.2011.8

“Writing a correct parallel program is difficult; writing a **highly modular parallel program** that performs well in a multiprogrammed environment is even more so. **Intel Threading Building Blocks** (Intel TBB), a key component of Intel Parallel Building Blocks , is a widely used C++ template library that helps developers achieve this goal.”

Michael Voss, Wooyoung Kim, ***Multicore Desktop Programming with Intel Threading Building Blocks****, IEEE Software, January 2011.*[739]

[739] DOI: 10.1109/MS.2011.12

"Looking at **software** from a **design perspective**, understanding software as a designed artifact, and considering how design reaches into the whole software life cycle can bring significant benefits both to our understanding of what works in software design and to our approach to tools and practices."

Andre van der Hoek, Alex Baker, Harold Ossher, Marian Petre, ***Guest Editors' Introduction: Studying Professional Software Design****, IEEE Software, January 2011.*[740]

[740] DOI: 10.1109/MS.2011.155

“**Software designers** make decisions covering a wide variety of aspects of the software to be designed through nested, intertwined processes. Some of these dependencies among design decisions might not be obvious, especially for people who didn’t start with the project at the beginning of the design process. Extending or **altering an existing design decision** without fully understanding its dependencies might result in a deterioration of the quality of the software design. “

Nobuto Matsubara, Kumiyo Nakakoji, Yoshinari Shirai, Yasuhiro Yamamoto, ***Toward Unweaving Streams of Thought for Reflection in Professional Software Design****, IEEE Software, January 2011.*[741]

[741] DOI: 10.1109/MS.2011.125

"**Collaboration** can enhance the output of **early-stage design**. When software designers or architects work together to define a problem and explore potential solutions, they find and address design problems earlier and arrive at more innovative and effective solutions than when they work alone. Nonetheless, collaboration can fail without **proper planning**."

Ania Dilmaghani, Jim Dibble, ***Strategies for Early-Stage Collaborative Design****, IEEE Software, January 2011.*[742]

[742] DOI: 10.1109/MS.2011.124

"A central task in design is deciding what artifact will best satisfy the client's needs, whether that requires creating an artifact or choosing from existing alternatives. **A design space** identifies and **organizes the decisions** that must be made, together with the alternatives for those decisions, thereby **providing guidance** for creating artifacts or a framework for comparing them."

---

*Mary Shaw, **The Role of Design Spaces**, IEEE Software, January 2011.*[743]

[743] DOI: 10.1109/MS.2011.121

"**Software design** is about **a sequence of steps** taken to achieve a goal. Designers must plan their approach to carrying out these steps. In studying designers at work, the authors observed **breadth- versus depth-first** approaches to design-space exploration and problem- versus solution-driven approaches during the actual design."

Hans van Vliet, Antony Tang, ***Design Strategy and Software Design Effectiveness****, IEEE Software, January 2011.*[744]

[744] DOI: 10.1109/MS.2011.130

"**Architectural decisions** are design decisions that are **hard to make** or **costly to change**."

Olaf Zimmermann, ***Architectural Decisions as Reusable Design Assets***, *IEEE Software, January 2011.*[745]

[745] DOI: 10.1109/MS.2011.3

"**Usability** has a significant impact on the success of software-centric systems and products. It relates to the actual usage of a system, but also to its effective design and development. Ultimately, failing to **build usable software** may degrade a project's ability to deliver in time, budget, functionality, and quality."

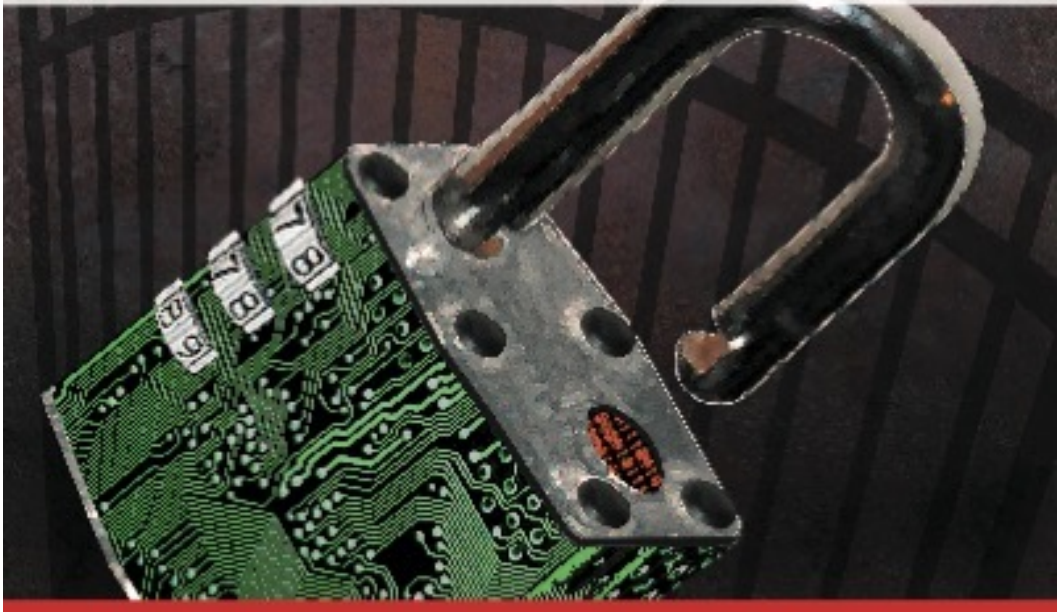
Frank Buschmann, ***Unusable Software Is Useless, Part 1****, IEEE Software, January 2011.*[746]

[746] DOI: 10.1109/MS.2011.19

MARCH/APRIL 2011 WWW.COMPUTER.ORG/SOFTWARE

IEEE Software

Software Protection



IEEE
IEEE computer society

"The metaphor of '**technical debt**' is useful for reasoning about trading off software development activities: An exclusive focus on implementing functionality can lead to **code decay**. Since this deterioration of the system usually reflects a **lack of** activity spent on **refactoring**, **documentation**, and other aspects of the **project infrastructure**, it can be viewed as a kind of debt that the developers owe the system."

Forrest Shull, ***Perfectionists in a World of Finite Resources****, IEEE Software, March 2011.*[747]

[747] DOI: 10.1109/MS.2011.38

"**Software protection** is increasingly becoming an important requirement for industrial software development, especially when building systems for **military defense**, **national infrastructure**, and **medical informatics**. Every software vendor should be **aware** of the **potential for attacks** against its products and the techniques available to mitigate these attacks. Employing **software protection techniques** can mean the difference between business survival and failure. "

Christian Collberg, Mikhail Atallah, Mariusz Jakubowski, Paolo Falcarin, ***Guest Editors' Introduction: Software Protection****, IEEE Software, March 2011.*[748]

[748] DOI: 10.1109/MS.2011.34

“Platforms such as **Windows Azure** let applications conduct data-intensive cloud computing. **Unit testing** can help ensure high-quality development of such applications, but the results **depend on** test inputs and the cloud **environment’s state**. Manually providing various test inputs and **cloud states** is laborious and time-consuming. However, automated test generation must **simulate** various **cloud states** to achieve effective testing.”

Jian Lu, Linghao Zhang, Tao Xie, Nikolai Tillmann, Xiaoxing Ma, Peli de Halleux, ***Environmental Modeling for Automated Cloud Application Testing****, IEEE Software, March 2011.*[749]

[749] DOI: 10.1109/MS.2011.158

"The large-scale, dynamic, and heterogeneous nature of cloud computing poses numerous security challenges. But the **cloud's** main challenge is to provide a **robust authorization mechanism** that incorporates **multitenancy** and **virtualization** aspects of resources."

Walid G. Aref, Arif Ghafoor, Saleh Basalamah, Abdulrahman A. Almutairi, Muhammad I. Sarfraz, ***A Distributed Access Control Architecture for Cloud Computing****, IEEE Software, March 2011.*[750]

[750] DOI: 10.1109/MS.2011.153

"As applications and services migrate to the cloud, testing will follow the same trend. Therefore, organizations must understand the **dynamics of cloud-based testing**. … cloud computing can make testing faster and enhance the delivery of testing services. Cloud computing also highlights important aspects of testing that require attention, such as **integration and interoperability**."

Ossi Taipale, Kari Smolander, Leah Riungu-Kalliosaari, ***Testing in the Cloud: Exploring the Practice****, IEEE Software, March 2011.*[751]

[751] DOI: 10.1109/MS.2011.132

"An experimental approach employs the **Google App Engine** (GAE) for high-performance parallel computing. A generic **master-slave framework** enables fast prototyping and integration of parallel algorithms that are transparently scheduled and executed on the Google cloud infrastructure. Compared to **Amazon Elastic Compute Cloud** (EC2), GAE offers lower resource-provisioning overhead and is cheaper for jobs shorter than one hour."

Radu Prodan, Simon Ostermann, Michael Sperk, ***Evaluating High-Performance Computing on Google App Engine****, IEEE Software, March 2011.*[752]

[752] DOI: 10.1109/MS.2011.131

MAY/JUNE 2011 WWW.COMPUTER.ORG/SOFTWARE

IEEE Software

Software Components: Beyond Programming

IEEE
IEEE computer society

"**Keeping up to date** with new software engineering methods, practices, and tools is challenging in the best of times, and made even more urgent by today's tough economic climate. … One way to save time is to take a look at **only the best content**. Because we don't always know ahead of time which that will be, people find these shortcuts useful: finding content from the **established thought leaders** in the field, rather than from unknown voices with unknown quality; and reading content where someone has already spent time **aggregating or summarizing** the best stuff from other raw materials."

Forrest Shull, ***How Do You Keep Up to Date?****, IEEE Software, May 2011.*[753]

[753] DOI: 10.1109/MS.2011.57

"**Architecting** a software-intensive system encompasses **technical elements**, **social considerations**, and a **technical core**. Most interesting systems start small and focus on technical concerns, but once they grow to the point of **economic significance**, **social issues** begin to loom large."

---

*Grady Booch, **The Architect's Journey**, IEEE Software, May 2011.*[754]

[754] DOI: 10.1109/MS.2011.66

"In the last decade, **software components** have been of an increased interest in software engineering community. The appealing concepts of **building systems from existing components** and reusing components, as well as the appearance of new technologies that enabled the separation of component development from system development, attracted researchers and industry to develop and apply principles of component-based software engineering."

Ivica Crnkovic, Clemens Szyperski, Judith Stafford, ***Software Components beyond Programming: From Routines to Services****, IEEE Software, May 2011.*[755]

[755] DOI: 10.1109/MS.2011.62

"Scientists commonly describe their data-processing systems metaphorically as **software pipelines**. These pipelines input one or more data sources and apply steps to transform the data and create useful results. Although **conceptually simple**, pipelines often adopt **complex topologies** and must meet stringent **quality-of-service** requirements that stress the software infrastructure used to construct the pipeline."

Yan Liu, Adam Wynne, Jian Yin, Ian Gorton, ***Components in the Pipeline****, IEEE Software, May 2011.*[756]

[756] DOI: 10.1109/MS.2011.23

JULY/AUGUST 2011 WWW.COMPUTER.ORG/SOFTWARE

IEEE Software

Software Business

IEEE
IEEE computer society

"**Software** plays an increasingly important role in **most aspects of business**. Many **new business models** for software-intensive enterprises have arisen in the last decade, ranging from selling software as a service to offshoring and crowdsourcing. Governments and **standards bodies** have also intervened to influence business models for stimulating growth in the industry. The software business has also had ancillary effects including the creation of new sectors such as **innovation management**. The management of **intellectual property rights** has become a more critical issue as software is embedded in more and more products."

Shari Lawrence Pfleeger, John Favaro, ***Guest Editors' Introduction: Software as a Business****, IEEE Software, July 2011.*[757]

[757] DOI: 10.1109/MS.2011.77

"**Open innovation** and the recent emphasis on **client involvement** imply the emergence of **hybrid software licensing models** combining the limited openness of source code with traditional value appropriation logic. … The central idea is that the vendor of commoditized products also **licenses source code to select clients**, who become participants in and subscribers to an ongoing closed development community."

---

*Mikko Riepula, **Sharing Source Code with Clients: A Hybrid Business and Development Model**, IEEE Software, July 2011.*[758]

[758] DOI: 10.1109/MS.2011.53

"**Cloud computing** offers new ways for firms to operate in the **global market** so that even **small firms can compete** in markets traditionally dominated by multinational corporations. "

Arto Ojala, Pasi Tyrväinen, ***Developing Cloud Business Models: A Case Study on Cloud Gaming****, IEEE Software, July 2011.*[759]

[759] DOI: 10.1109/MS.2011.51

"**Refactoring** is limited in what qualities it can help improve. It can also do **more harm than good** when practiced **informally** or ad hoc or when it's used as a **synonym** for **any form of change** in a system."

Frank Buschmann, ***Gardening Your Architecture, Part 1: Refactoring****, IEEE Software, July 2011.*[760]

[760] DOI: 10.1109/MS.2011.76

SEPTEMBER/OCTOBER 2011 WWW.COMPUTER.ORG/SOFTWARE

IEEE Software

Engineering Fun

IEEE
IEEE computer society

"**Security and privacy** are **interdependent** concepts. Each impacts the other, but to say that they are alternatives is a false dichotomy. Both are issues of **human concern**; their policies and their risks may be made manifest in software-intensive systems. Architecting a system that attends to the needs of security and privacy is possible and desirable, yet there are often unintended and **unexpected consequences** in so doing."

Grady Booch, ***Unintentional and Unbalanced Transparency****, IEEE Software, September 2011.*[761]

[761] DOI: 10.1109/MS.2011.112

"Successful development of **video games** hinges on understanding the difficulties of ensuring the resulting product is **fun**. Addressing this **soft requirement**, incorporating nontrivial multimedia, and other domain-specific concerns bring novel challenges to software development. "

Paul Kruszewski, Clark Verbrugge, ***Guest Editors' Introduction: Engineering Fun****, IEEE Software, September 2011.*[762]

[762] DOI: 10.1109/MS.2011.98

"Introducing reuse and **software product line** (SPL) concepts into digital **game-development processes** isn't a straightforward task. This work presents a systematic process for bridging SPLs to game development, culminating with **domain-specific languages** and generators streamlined for game subdomains."

Geber L. Ramalho, Andre W.B. Furtado, Eduardo Santana de Almeida, Andre L.M. Santos, ***Improving Digital Game Development with Software Product Lines****, IEEE Software, September 2011.*[763]

[763] DOI: 10.1109/MS.2011.101

"The design of **massively multiplayer online games** (MMOGs) is challenging because scalability, consistency, reliability, and fairness must be achieved while providing good performance and enjoyable gameplay."

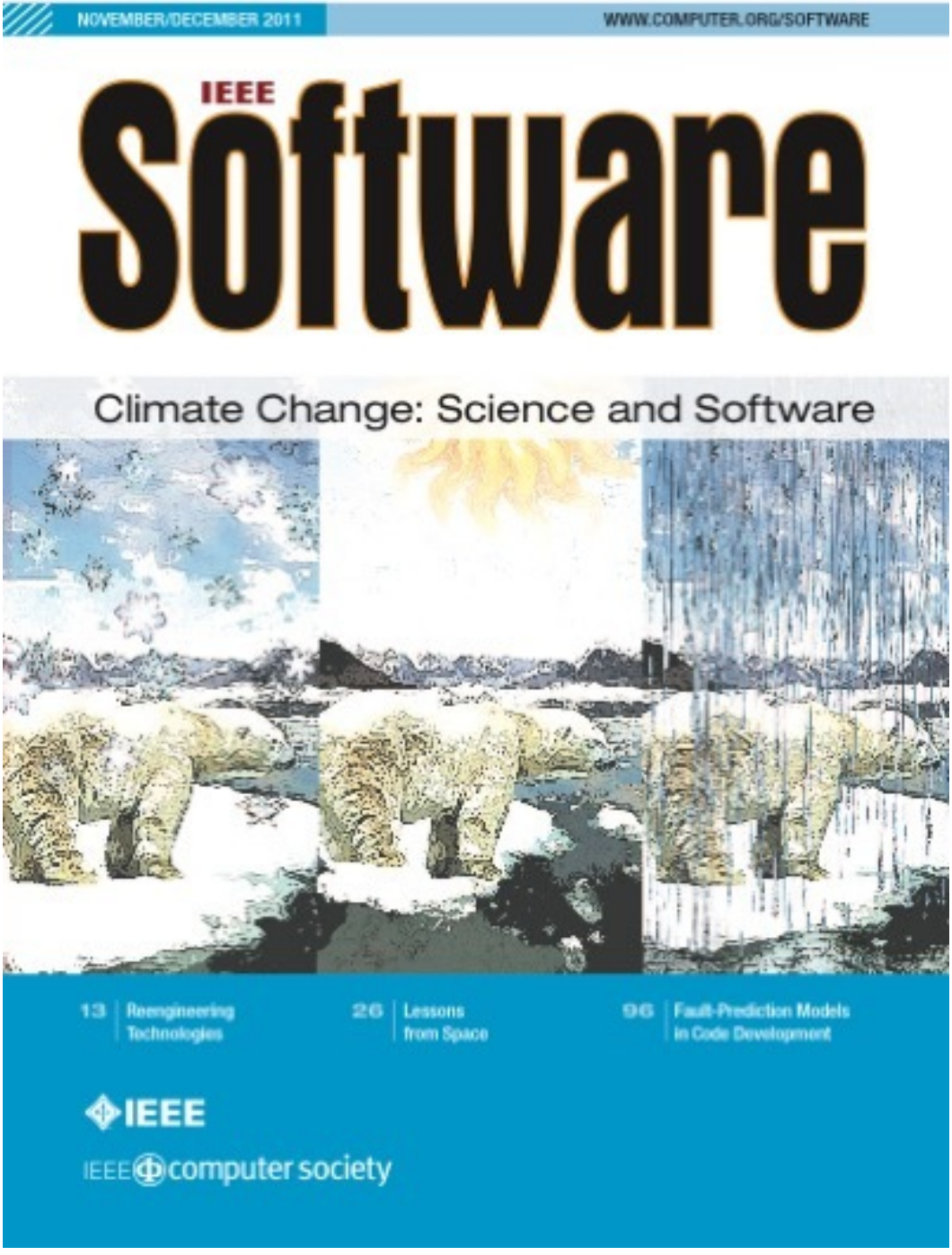
Jörg Kienzle, Alexandre Denault, ***Journey: A Massively Multiplayer Online Game Middleware****, IEEE Software, September 2011.*[764]

[764] DOI: 10.1109/MS.2011.88

"**Games** must be **emergent**, constantly surprising players by the possibilities they offer. However, emergence **creates unpredictability**, preventing developers from verifying that their games won't lead to undesirable states. Worse still, even when **a bug** is found, finding out how it occurred can be a significant challenge."

Chris Lewis, Jim Whitehead, ***Repairing Games at Runtime or, How We Learned to Stop Worrying and Love Emergence****, IEEE Software, September 2011.*[765]

[765] DOI: 10.1109/MS.2011.87


NOVEMBER/DECEMBER 2011
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
Climate Change: Science and Software
13 Reengineering Technologies
26 Lessons from Space
96 Fault-Prediction Models in Code Development
IEEE
IEEE computer society

“If you give a **hurricane meteorologist** a giant pile of data about a storm spinning in the middle of the Atlantic Ocean and ask her to **determine exactly** where it will come ashore, she can analyze the data, construct a detailed and accurate model of the atmospheric conditions and weather patterns, run some simulations, and come up with a forecast—**two months after** the storm hits. It’ll probably be wrong, but not by much, a moot point for the people in the storm’s path. The **problem isn’t** with the **forecast’s accuracy** but with **the time** needed to prepare it. On the other extreme, if given a satellite image and a few points of other data—and a few minutes—a hurricane meteorologist can prepare a forecast so uncertain it might as well not even exist. There is a point where forecast **accuracy and timeliness overlap**. The model favored by hurricane meteorologists is to **do just enough** data acquisition and analysis to **be reasonably certain** what the storm will do and then **start telling people** to get ready.”

Eric Richardson, ***What an Agile Architect Can Learn from a Hurricane Meteorologist****, IEEE Software, November 2011.*[766]

[766] DOI: 10.1109/MS.2011.152

"Software systems must **continually evolve** to meet ever changing needs. However, such systems often become **legacy systems** as a consequence of **uncontrolled maintenance** combined with **obsolete technology**. To control maintenance costs and preserve complex embedded business rules, companies must **evolve their legacy systems**."

Mario Piattini, Ignacio García-Rodriguez de Guzmán, Christof Ebert, Ricardo Pérez-Castillo, ***Reengineering Technologies****, IEEE Software, November 2011.*[767]

[767] DOI: 10.1109/MS.2011.145

"There is complexity, and then there is organized complexity. **Pure complexity** is **chaotic**; **organized complexity** is full of **patterns**. Naming these **patterns** and respecting their intention is the essence of architecture."

---

*Grady Booch, **The Architecture of Small Things**, IEEE Software, November 2011.*[768]

[768] DOI: 10.1109/MS.2011.148

"**Climate change** is likely to be one of the defining global issues of the 21st century. The past decade - the hottest in recorded history - has witnessed countries around the world struggling to deal with drought, heat waves, and extreme weather. The sheer **scale of the problem** also makes it hard to understand, predict, and solve. **Climate science** journals regularly publish special issues on specific climate models, typically timed to present results from a major new release of a given model. However, these tend to focus on the **new science** that the model enables, rather than to describe the **software** and its development."

Steve M. Easterbrook, Venkatramani Balaji, Reinhard Budich, Paul N. Edwards, ***Guest Editors' Introduction: Climate Change - Science and Software****, IEEE Software, November 2011.*[769]

[769] DOI: 10.1109/MS.2011.141

"The **Clear Climate Code** project rewrote GISTEMP, a legacy software system used to produce an important global surface temperature dataset. The **focus** of the project is **on clarity**: making the source code as clear as possible to interested people, to **improve public understanding**."

David Jones, Nicholas Barnes, ***Clear Climate Code: Rewriting Legacy Science Software for Clarity****, IEEE Software, November 2011.*[770]

[770] DOI: 10.1109/MS.2011.113

"**Coupled climate models** exhibit scientific, numerical, and architectural **variability.** This variability introduces requirements that give rise to **complexity**."

Spencer Rugaber, Sameer Ansari, Leo Mark, Rocky Dunlap, ***Managing Software Complexity and Variability in Coupled Climate Models****, IEEE Software, November 2011.*[771]

[771] DOI: 10.1109/MS.2011.114

"**Models** play a central role for **climate change policy-makers**, but they're often so **complex and computationally demanding** that experts must run them and interpret their results. This reduces stakeholders' ability to **explore alternative scenarios**, increases perceptions of model complexity and opacity, and can ultimately reduce public confidence ."

Joshua Introne, Robert Laubacher, Thomas Malone, ***Enabling Open Development Methodologies in Climate Change Assessment Modeling****, IEEE Software, November 2011.*[772]

[772] DOI: 10.1109/MS.2011.115

"**Lateness** is the **most common** form of software project **failure**."

---

*Tom DeMarco, **All Late Projects Are the Same**, IEEE Software, November 2011.*[773]

[773] DOI: 10.1109/MS.2011.134

2012


JANUARY/FEBRUARY 2012
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
Professional Design / Algorithms for Today's Practitioner
Model-Based Testing // 14
Conway's Law Revisited // 90
IEEE
IEEE computer society

"In **model-based testing** (MBT), manually selected algorithms automatically and systematically **generate test cases** from a set of models of the system under test or its environment. Whereas test automation replaces manual test execution with automated test scripts, MBT replaces manual test designs with automated test designs and test generation. "

Ina Schieferdecker, ***Model-Based Testing****, IEEE Software, January 2012.*[774]

[774] DOI: 10.1109/MS.2012.13

"**Architecture mastery** is more than professional expertise in modern software engineering methods and techniques. It is mainly in how architects approach design. Particularly, the '**things between things**' require the architect's full attention: domain concepts hidden between the lines of code; interactions and interfaces residing between components; and even choices between design options. This is the **architect's territory**, and successful architecture **uncovers the things 'in-between'** as early as possible, make them explicit, and decide about them!"

Frank Buschmann, ***To Boldly Go Where No One Has Gone Before****, IEEE Software, January 2012.*[775]

[775] DOI: 10.1109/MS.2012.18

"Enormous advances in computing power and programming environments have obscured the **importance of algorithms**, one of the **foundational pillars** of software engineering. Today, even university curricula too often pay only lip service to the teaching of algorithmic fundamentals, reinforcing the popular belief that their place at the core of a software engineer's **education is past**. Yet even today, the importance of algorithms in software engineering has not diminished, and the effects of neglect are evident everywhere in needlessly **inefficient industrial applications**."

Giuseppe Prencipe, John Favaro, Cesare Zavattari, Alessandro Tommasi, ***Guest Editors' Introduction: Algorithms and Today's Practitioner****, IEEE Software, January 2012.*[776]

[776] DOI: 10.1109/MS.2012.9

"Formally speaking, **mastering complexity** requires a proof of the asymptotic computation, storage, and communication needs of a system. While we don't always do formal specifications and proofs of the properties of our algorithms, the underlying **behavior of the algorithms** factors into our **capacity modeling**—and therefore our capital and operational expense planning—in a fundamental way."

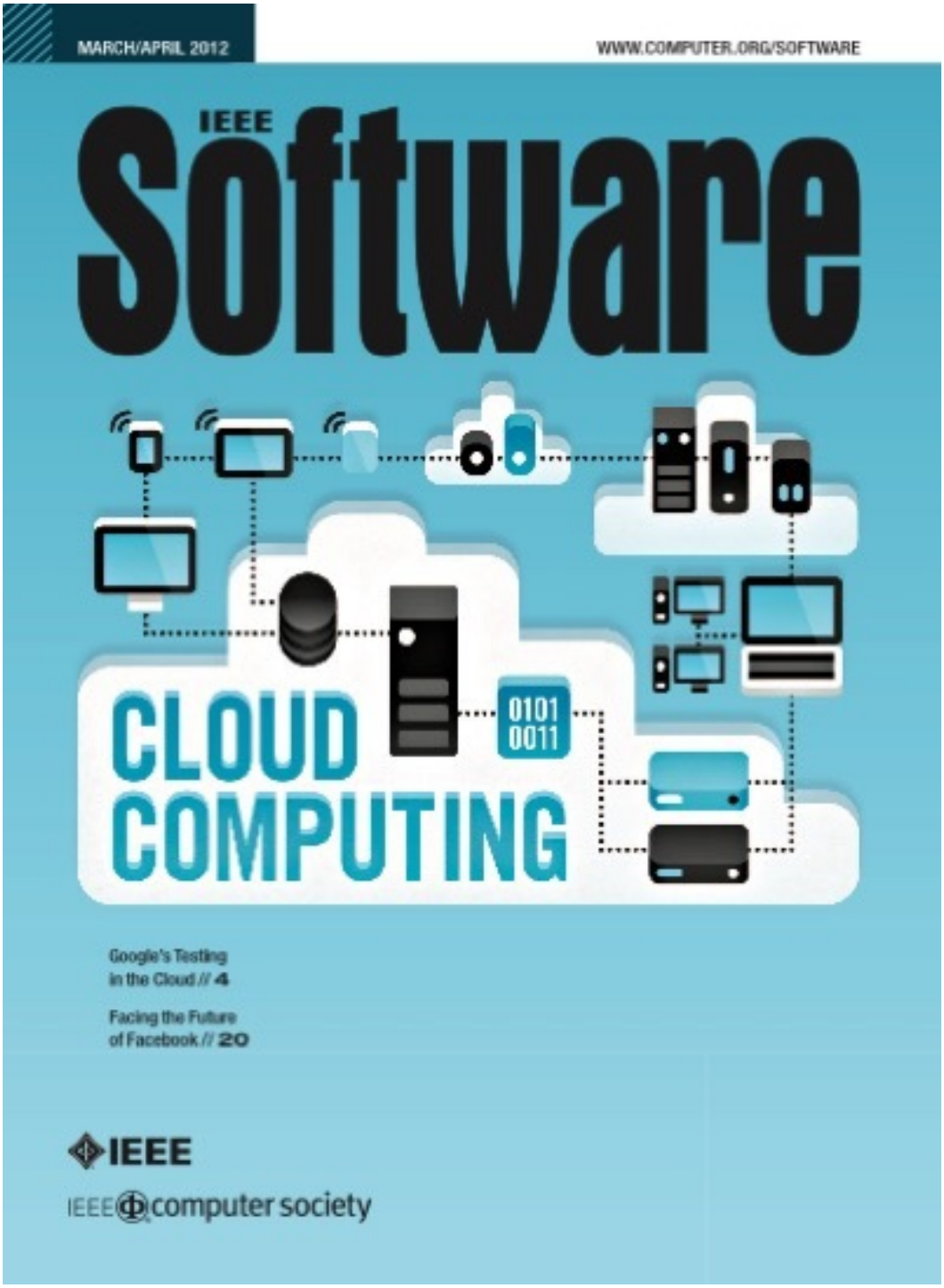
John Favaro, ***Excellence in Search: An Interview with David Chaiken****, IEEE Software, January 2012.*[777]

[777] DOI: 10.1109/MS.2012.7

"**Conway's law**, also called the **mirroring hypothesis**, predicts that a development **organization** will inevitably design **systems** that **mirror** its **organizational communication** structure."

Marcelo Cataldo, Irwin Kwan, Daniela Damian, ***Conway's Law Revisited: The Evidence for a Task-Based Perspective****, IEEE Software, January 2012.*[778]

[778] DOI: 10.1109/MS.2012.3


MARCH/APRIL 2012
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
CLOUD
COMPUTING
0101
0011
Google's Testing
in the Cloud // 4
Facing the Future
of Facebook // 20
IEEE
IEEE computer society

"The increasing pervasiveness of **cloud computing** is changing the state of the practice in software testing. … interview with **James Whittaker** … covers key technology changes, such as more pervasive access to **monitoring frameworks**, the ability to **aggregate and act on feedback** directly from massive user communities (the '**crowdsourcing**' of quality assurance), and the ability to know the **exact machine configuration** when **bugs** are discovered."

---

*Forrest Shull, **A Brave New World of Testing? An Interview with Google's James Whittaker**, IEEE Software, March 2012.*[779]

[779] DOI: 10.1109/MS.2012.23

"There comes **a point of no return** in the life of every successful software-intensive system, a point where you can no longer place a pile of your best developers at one end of a lever and expect them to move the world. Rather, you must come to realize that **putting piles of developers** at the end of even the longest lever is no longer the right tool to use. Crossing that point while still preserving the values and the **tribal memory** of your organization's development culture requires some serious adult supervision."

---

*Grady Booch, **Facing Future**, IEEE Software, March 2012.*[780]

[780] DOI: 10.1109/MS.2012.29

"**Cloud computing** is a new paradigm for software systems where applications are divided into sets of **composite services** hosted on leased, **highly distributed platforms**. There are many new software engineering challenges in building effective cloud-based software applications."

Jacky Keung, Gerald Kaefer, Anna Liu, John Grundy, ***Guest Editors' Introduction: Software Engineering for the Cloud****, IEEE Software, March 2012.*[781]

[781] DOI: 10.1109/MS.2012.31

"There is broad consensus that **architects should code**. Yet the challenging question is: how can architects program without being lost in myriads of local code details? … **Agile practices** help architects to balance their coding activities with other duties, allowing them to be in control of the amount of time they spend on programming and the concerns and system parts on which they program."

---

*Frank Buschmann, Jörg Bartholdt, **Code Matters!**, IEEE Software, March 2012.*[782]

[782] DOI: 10.1109/MS.2012.27

"A **package management system** organizes and simplifies the installation and maintenance of software by standardizing and organizing the production and consumption of software collections. As a software developer, you can benefit from package managers in two ways: through a rich and stable development environment and through **friction-free reuse**. Promisingly, the structure that package managers bring both to the tools we use in our development process and the libraries we reuse in our products ties nicely with the recent move emphasizing **DevOps** (development operations) as an integration between software development and IT operations."

Diomidis Spinellis, ***Package Management Systems***, *IEEE Software, March 2012.*[783]

[783] DOI: 10.1109/MS.2012.38

“Almost everyone agrees that there is **a gender gap** in computer science, where there are far too few females participating in the field. But does that gap occur in the whole of the field of computing? This sounding board explores the notion that the gap is **unique to CS**, and that any solution to the problem must occur within that field and not the broader field of computing.”

Robert L. Glass, ***The Gender Gap: Is It a Computing Problem or Simply a Computer Science Problem?****, IEEE Software, March 2012.*[784]

[784] DOI: 10.1109/MS.2012.44


MAY/JUNE 2012
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
SE FOR
COMPLIANCE
?
Disbanding the "Process Police" // 3
Managing Privacy-Enhancing
Technologies // 45
IEEE
IEEE computer society

“Even the fanciest videoconferencing or 3G holography can’t overcome **the fundamental time-zone problem** — that it’s sleep time on the other side of the world. Consequently, no miracle technology will overcome time-zone differences.”

Christof Ebert, Rafael Prikladnicki, Sabrina Marczak, Erran Carmel, ***Technologies to Support Collaboration across Time Zones****, IEEE Software, May 2012.*[785]

[785] DOI: 10.1109/MS.2012.68

"Typically, organizations face conflicting objectives, with **compliance policies** possibly **hindering innovation**, slowing down the product development process, or making the whole process **most costly**. The goal of software engineering for compliance is to bridge **the gap** between the **software engineering community** and the **compliance community**."

Ayse Bener, Uwe Zdun, Erlinda L. Olalia-Carin, ***Guest Editors' Introduction: Software Engineering for Compliance****, IEEE Software, May 2012.*[786]

[786] DOI: 10.1109/MS.2012.63

"**Ensuring compliance** to laws, regulations, and standards in a **constantly changing** business environment is a major challenge for companies. So, organizations have an increasing need for **systematic approaches** to manage compliance throughout the business process (BP) life cycle."

Michael P. Papazoglou, Amal Elgammal, Willem-Jan van den Heuvel, Oktay Turetken, ***Capturing Compliance Requirements: A Pattern-Based Approach****, IEEE Software, May 2012.*[787]

[787] DOI: 10.1109/MS.2012.45

"The changing **global business environment** and continued introduction of new technologies are significantly affecting organizations' privacy practices. In this environment, **privacy-enhancing technology (PET)** often becomes a key to protecting personal information."

David Pelkola, ***A Framework for Managing Privacy-Enhancing Technology****, IEEE Software, May 2012.*[788]

[788] DOI: 10.1109/MS.2012.47

"**Leadership** is the key for architects to balance all their activities and duties with the interests of different **stakeholders** without losing control of the architecture under development. They must have a **clear vision** and strict focus on key aspects of success. All their activities should be **goal-driven** and in direct cooperation and interaction with the relevant stakeholder groups."

---

*Frank Buschmann, **A Week in the Life of an Architect**, IEEE Software, May 2012.*[789]

[789] DOI: 10.1109/MS.2012.55

“**Git** is a distributed revision control system available on all mainstream development platforms through a free software license. An important difference of git over its older ancestors is that it elevates the **software’s revisions** to **first-class citizens**. “

Diomidis Spinellis, ***Git***, *IEEE Software*, *May 2012.*[790]

[790] DOI: 10.1109/MS.2012.61


JULY/AUGUST 2012
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
MOBILE SOFTWARE DEVELOPMENT
SOA: Myth or Reality? // 46
Social and Community Intelligence // 88
IEEE
IEEE computer society

"**Smart mobile devices** have had a **huge impact** on the world today with new apps being produced at a prodigious rate. How we got to this point has a lot to do with the **ease of use** that manufacturers and app developers have achieved, which includes aspects such as **quick response time**, **intuitive interfaces**, and well-designed functionality."

Forrest Shull, ***Designing a World at Your Fingertips: A Look at Mobile User Interfaces****, IEEE Software, July 2012.*[791]

[791] DOI: 10.1109/MS.2012.81

"**Computing has transformed humanity** in ways that we have only begun to metabolize. Computing amplifies what we celebrate most about being human, but it also has the capacity to magnify that which we mourn. **Exploring the story of computing** has value, for an **educated populace** is far more able to reconcile its past, reason about its present, and be intentional about its future."

---

*Grady Booch, **The Human Experience**, IEEE Software, July 2012.*[792]

[792] DOI: 10.1109/MS.2012.103

"**Smartphones** aren't very 'smart' without the **software apps** that give them their usability and versatility. Apps, like all software, need some degree of guidance, regulation, and measurement to ensure a user is receiving proper functionality and quality of service."

Jeffrey Voas, J. Bret Michael, Michiel van Genuchten, ***The Mobile Software App Takeover****, IEEE Software, July 2012.*[793]

[793] DOI: 10.1109/MS.2012.104

"**Mobile devices** have become a commodity: we use several devices for various purposes. Although we carry only some of our devices with us, we still want to access content originating from any device. To overcome this issue, device users often upload content into a hosting service available in the cloud. However, **cloud-based** hosting can **alienate the control** and ownership of the content."

Niko Mäkitalo, Varvara Myllärniemi, Tommi Mikkonen, Mikko Raatikainen, Tomi Männistö, Juha Savolainen, ***Mobile Content as a Service A Blueprint for a Vendor-Neutral Cloud of Mobile Devices****, IEEE Software, July 2012.*[794]

[794] DOI: 10.1109/MS.2012.54

"**Service-oriented architecture (SOA)** has gained significant attention as a means of developing flexible and modular systems. .. not all stated benefits are realised due to, among other things, a failure of service-oriented **thinking at an organisational level**, problems allocating **financial responsibility** for services within and between organisations, and a lack of **mature tool chains**."

Fethi A. Rabhi, Haresh Luthria, ***Service-Oriented Architectures: Myth or Reality?****, IEEE Software, July 2012.*[795]

[795] DOI: 10.1109/MS.2011.156

"**Codification** and **testing of business rules** in application programs has historically been a challenge in software engineering. Many organizations have adopted the business rules approach to formalize and compartmentalize business rules as a separate component from application code."

Euntae T. Lee, Chen Zhang, Thomas O. Meservy, Jasbir Dhaliwal, ***The Business Rules Approach and Its Effect on Software Testing****, IEEE Software, July 2012.*[796]

[796] DOI: 10.1109/MS.2011.120


SEPTEMBER/OCTOBER 2012
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
LEAN
SOFTWARE
DEVELOPMENT
The Software behind
the Higgs Boson Discovery // 11
Where's the Theory
for Software Engineering? // 96
IEEE
IEEE computer society

"A close look at the **evidence** underpinning the original concept of **lean production** and its popular interpretation reveals the inherent challenges of measuring and interpreting evidence for performance differences."

Helen Sharp, Tore Dybå, ***What's the Evidence for Lean?****, IEEE Software, September 2012.*[797]

[797] DOI: 10.1109/MS.2012.126

"Although some claim that **principles from other fields** can't apply to a creative and design-oriented discipline such as software development, many studies have proven **the simple wisdom** that we all benefit from **empowered and motivated teams**, we build our products faster and with better quality if **market strategy is understood** and **requirements changes managed**, we **learn from previous defects**, we can emphasize **repeatable processes**, and we should build from high-quality components. The software industry is now poised to transform to **customer-centric development**, which translates into both reducing the total life-cycle cost and increasing efficiency and effectiveness by **eliminating waste** and **adding value**."

Pekka Abrahamsson, Christof Ebert, Nilay Oza, ***Lean Software Development****, IEEE Software, September 2012.*[798]

[798] DOI: 10.1109/MS.2012.116

"The term '**lean**' … describes any efficient management practice that **minimized waste**, including in product development … In lean terms, '**waste**' is anything that doesn't either **add customer value directly** or **add knowledge** about how to deliver that value more effectively."

Mary Poppendieck, Michael A. Cusumano, ***Lean Software Development: A Tutorial****, IEEE Software, September 2012.*[799]

[799] DOI: 10.1109/MS.2012.107

"**Lean practices** use the principle of **Little's law** to improve the flow of value to the end user by eliminating sources of waste from a software development process. **Little's law** defines throughput as a ratio of **work in process** and **cycle time**. Increasing throughput (or productivity) requires continuously improving (that is, decreasing) cycle time while ensuring that the work-in-process limit doesn't exceed the capacity available to process the work."

Robert L. Nord, Ipek Ozkaya, Raghvinder S. Sangwan, ***Making Architecture Visible to Improve Flow Management in Lean Software Development****, IEEE Software, September 2012.*[800]

[800] DOI: 10.1109/MS.2012.109

“Modern **virtualization technology** allows us to run operating systems in a virtual machine that can be hosted on facilities ranging from our laptop to a datacenter in the cloud. It’s thus possible to create **a virtualized development environment** that contains all the tools, applications, and libraries that a programmer requires. This speeds up developer setup time, brings economies of scale, introduces parity between development and production environments, allows the use of platform-specific tools, and simplifies embedded-system development. Using VMs, testers can ensure a **pristine environment** and access to diverse (virtual) platforms. Deployment is also simplified by packaging all the system’s components and setup into a **VM appliance**. Finally, on the operations side, VMs make it easier for a system to support application **provisioning**, **maintenance windows**, **high availability**, and **disaster recovery**.”

Diomidis Spinellis, ***Virtualize Me***, *IEEE Software, September 2012.*[801]

[801] DOI: 10.1109/MS.2012.125


NOVEMBER/DECEMBER 2012
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
TECHNICAL DEBT
Cherishing Ambiguity // 9
The 10-Minute Test Plan // 70
IEEE
IEEE computer society

"**Computing** was once a companion to conflict; computing is now an instrument of war; computing is becoming a **theater of war**. Along the way, conflict has shaped computing, and computing has changed the nature of warfare."

---

*Grady Booch, **Woven on the Loom of Sorrow**, IEEE Software, November 2012.*[802]

[802] DOI: 10.1109/MS.2012.168

"The metaphor of **technical debt** in software development was introduced two decades ago to explain to **nontechnical stakeholders** the need for what we call now '**refactoring**.'"

Ipek Ozkaya, Robert L. Nord, Philippe Kruchten, ***Technical Debt: From Metaphor to Theory and Practice****, IEEE Software, November 2012.*[803]

[803] DOI: 10.1109/MS.2012.167

"**Agile teams** create **business value** by responding to changing business environments and delivering working software at regular intervals. While doing so, they make **design tradeoffs** to satisfy business needs such as meeting a release schedule. **Technical debt** is the result of such decisions or tradeoffs. When this happens, agile teams must pay off the **accumulated debt** by improving designs during subsequent iterations in order to improve maintainability."

Raja Bavani, ***Distributed Agile, Agile Testing, and Technical Debt***, *IEEE Software, November 2012.*[804]

[804] DOI: 10.1109/MS.2012.155

"**Technical debt** is more than a metaphor: applying **finance and accounting practices** typical of other business obligations to technical debt can, in addition to meeting ethical and legal governance requirements, generate real, sustained financial benefits."

Patrick Conroy, ***Technical Debt: Where Are the Shareholders' Interests?****, IEEE Software, November 2012.*[805]

[805] DOI: 10.1109/MS.2012.166

2013


JANUARY/FEBRUARY 2013
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
BRIDGING
SOFTWARE
COMMUNITIES
THROUGH
SOCIAL
NETWORKING
Cyber Dumpster Diving // 9
The Airbus A380's Cabin Software // 21
Programming with Ghosts // 74
IEEE
IEEE computer society

"Computing is transforming every aspect of the human experience. As creators of this technology, what obligations do we have to the **general public**, for whom we make the **complex machinery** of computing increasingly invisible? … it's important for the general public to know something about the **technology behind the curtain** of computing."

---

*Grady Booch, **The Great and Terrible Oz**, IEEE Software, January 2013.*[806]

[806] DOI: 10.1109/MS.2013.16

"Most people think of **requirements** as things to manipulate at the **start of a project**. Others, more enlightened, recognize that requirements also have a role toward the end of projects to **test compliance**. But few people have recognized an active role for requirements during their **system's use** - to **monitor** whether the system continues to **comply with its requirements** during its lifetime."

---

*Neil Maiden, **Monitoring Our Requirements**, IEEE Software, January 2013.*[807]

[807] DOI: 10.1109/MS.2013.10

"'**Innovation**' and '**innovative architecture**' are topics of broad popularity in software engineering. Yet, the two terms appear to mean different things to different people – with interpretations of both driven more by **personal interests** than by their **true meanings**. It's therefore essential for architects to have a clear understanding of what 'innovation' means in the context of their projects if they are to make the **right design decisions** and communicate the **intended messages** to project stakeholders."

---

*Frank Buschmann, **Innovation Reconsidered**, IEEE Software, January 2013.*[808]

[808] DOI: 10.1109/MS.2013.9

"Over the past decade, the advent of **social networking** has fundamentally altered the landscape of how software is used, designed, and developed. It has expanded how **communities of software stakeholders** communicate, **collaborate**, **learn** from, and **coordinate** with one another."

Andrew Begel, Jan Bosch, Margaret-Anne Storey, ***Bridging Software Communities through Social Networking****, IEEE Software, January 2013.*[809]

[809] DOI: 10.1109/MS.2013.3

"**Software development** is increasingly carried out by **developer communities** in a **global setting**. One way to prepare for development success is to uncover and **harmonize** these **communities** to exploit their collective, collaborative potential."

Hans van Vliet, Patricia Lago, Damian A. Tamburri, ***Uncovering Latent Social Communities in Software Development****, IEEE Software, January 2013.*[810]

[810] DOI: 10.1109/MS.2012.170

"A new generation of **development environments** takes a radical approach to communication and coordination by **fusing social networking** functionality with flexible, **distributed version control**. Through these transparent work environments, people, repositories, development activities, and their histories are immediately and easily visible to all users …. **transparency** helps developers on **GitHub** manage their projects, handle dependencies more effectively, reduce communication needs, and figure out what requires their attention."

James Herbsleb, Jason Tsay, Colleen Stuart, Laura Dabbish, ***Leveraging Transparency****, IEEE Software, January 2013.*[811]

[811] DOI: 10.1109/MS.2012.172

"The **Social Web** provides comprehensive and publicly available **information about software developers**, identifying them as **contributors** to open source projects, **experts** at maintaining ties on social network sites, or **active participants** on knowledge-sharing sites. These **signals**, when aggregated and summarized, could be used to define potential **candidates' individual profiles**: potential employers could qualitatively evaluate job seekers, even those lacking a formal degree or changing their career path, by assessing candidates' online contributions."

Andrea Capiluppi, Alexander Serebrenik, Leif Singer, ***Assessing Technical Candidates on the Social Web****, IEEE Software, January 2013.*[812]

[812] DOI: 10.1109/MS.2012.169

"Many successful software companies use **social networking** as a way to **improve the services** or products they provide. … semistructured interviews with leaders from four successful companies: **Brian Doll**, an engineer who manages GitHub's marketing; **Doug Laundry**, a principal group program manager at Microsoft; **David Fullerton**, vice president of engineering at Stack Exchange; and **Robert Hughes**, the president and chief operating officer of TopCoder."

Andrew Begel, Jan Bosch, Margaret-Anne Storey, ***Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder****, IEEE Software, January 2013.*[813]

[813] DOI: 10.1109/MS.2013.13

"**Mutation testing** improves a system's bug-detection capability. It also helps improve coverage by exposing software or code areas that other types of testing might not expose."

Izzat Mahmoud Alsmadi, ***Using Mutation to Enhance GUI Testing Coverage***, *IEEE Software, January 2013.*[814]

[814] DOI: 10.1109/MS.2012.22

"**What works** for **whom**, **where**, **when**, and **why** is the ultimate question of **evidence-based software engineering**. Still, the empirical research seems mostly concerned with identifying **universal relationships** that are independent of how work settings and other contexts interact with the processes important to software practice. Questions of '**What is best?**' seem to prevail. For example, 'Which is better: pair or solo programming? test-first or test-last?' However, just as the question of **whether a helicopter is better than a bicycle** is meaningless, so are these questions because the answers depend on the settings and goals of the projects studied."

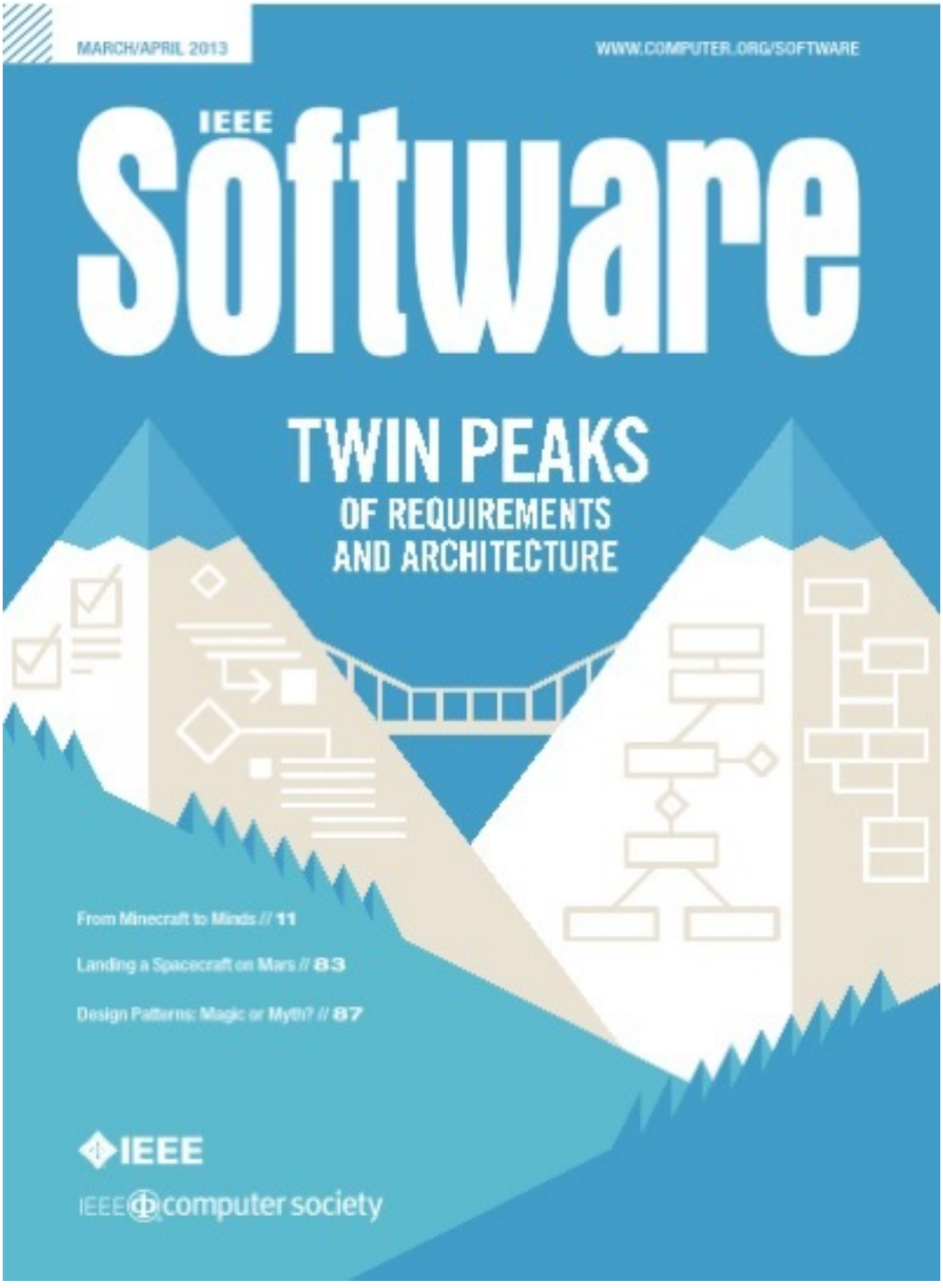
Tore Dyba, ***Contextualizing empirical evidence****, IEEE Software, January 2013.*[815]

[815] DOI: 10.1109/MS.2013.4

"**Testing** is a **destructive task** in which the goal is to find relevant defects as early as possible. It **requires automation** to reduce cost and ensure high regression, thus delivering determined quality. … In practice, **XUnit** frameworks are the most used technology to automate tests. In such frameworks, test cases are written in an **executable language** and can be executed automatically. They also provide specific operations to implement the **test case oracles**."

Macario Polo, Pedro Reales, Mario Piattini, Christof Ebert, ***Test Automation****, IEEE Software, January 2013.*[816]

[816] DOI: 10.1109/MS.2013.15


MARCH/APRIL 2013
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
TWIN PEAKS
OF REQUIREMENTS
AND ARCHITECTURE
From Minecraft to Minds // 11
Landing a Spacecraft on Mars // 83
Design Patterns: Magic or Myth? // 87
IEEE
IEEE computer society

"The subject of the **computability of the mind** introduces complex philosophical, ethical, and technical issues. That aside, this topic draws us in to the **nature of algorithms**. We are **surrounded by algorithms**; much of the history of computing is also the history of the advance of algorithms. For the public, algorithms are part of **computing's self-made mystery**, but to understand their nature is an important part of computational thinking."

---

*Grady Booch, **From Minecraft to Minds**, IEEE Software, March 2013.*[817]

[817] DOI: 10.1109/MS.2013.28

"**Requirements work** is really about **problem solving**. Its primary function is to locate and scope problems, then create and describe solutions for them."

Neil Maiden, ***So, What Is Requirements Work?****, IEEE Software, March 2013.*[818]

[818] DOI: 10.1109/MS.2013.35

"**Quality concerns**, often referred to as **nonfunctional requirements**, **service-level agreements**, **quality attributes**, **performance constraints**, or **architecturally significant requirements**, describe system-level attributes such as **security**, **performance**, **reliability**, and **maintainability**. In conjunction with functional requirements, these quality concerns drive and constrain a **system's architectural design** and often introduce significant **trade-offs** that must be carefully considered and balanced. The dependencies that exist between requirements and architecture have been referred to as the **twin peaks** of requirements and architecture."

Jane Cleland-Huang, Robert S. Hanmer, Sam Supakkul, Mehdi Mirakhorli, ***The Twin Peaks of Requirements and Architecture****, IEEE Software, March 2013.*[819]

[819] DOI: 10.1109/MS.2013.39

"The most useful forms of **documentation** are **views of the software** that can be **automatically generated**."

Mehdi Mirakhorli, Jane Cleland-Huang, ***Traversing the Twin Peaks****, IEEE Software, March 2013.*[820]

[820] DOI: 10.1109/MS.2013.40

"In the past decade, researchers have devised many methods to support and **codify architecture design**. However, what hampers such methods' adoption is that these methods employ abstract concepts such as **views**, **tactics**, and **patterns**, whereas practicing software architects choose technical design primitives from the services offered in **commercial frameworks**. … systematically links both **top-down concepts**, such as **patterns** and **tactics**, and **implementation artifacts**, such as frameworks, which are bottom-up concepts."

Rick Kazman, Perla Velasco-Elizondo, Humberto Cervantes, ***A Principled Way to Use Frameworks in Architecture Design****, IEEE Software, March 2013.*[821]

[821] DOI: 10.1109/MS.2012.175

“Systems are naturally constructed in **hierarchies**, in which **design choices** made at higher levels of abstraction levy requirements on system components at the lower levels. Thus, whether an aspect of a system is a **design choice** or **a requirement** largely depends on your **vantage poin**t within the system components’ hierarchy. “

Sanjai Rayadurgam, Mats P.E. Heimdahl, Anitha Murugesan, Darren Cofer, Andrew Gacek, Michael W. Whalen, ***Your ”What” Is My ”How”: Iteration and Hierarchy in System Design****, IEEE Software, March 2013.*[822]

[822] DOI: 10.1109/MS.2012.173

"Software architects often must work with **incomplete or ill-specified non-functional requirements** (NFRs) and use them to make decisions. Through this process, existing NFRs are **refined or modified** and new ones emerge. … The survey revealed that architects usually elicit NFRs themselves in an iterative process; they usually **don't document** the NFRs and only **partially validate** them."

Claudia Ayala, David Ameller, Jordi Cabot, Xavier Franch, ***Non-functional Requirements in Architectural Decision Making****, IEEE Software, March 2013.*[823]

[823] DOI: 10.1109/MS.2012.176

"There's clearly **no single magic tool** or technique that can be used to secure the reliability of any large and complex software application; rather, it takes good tools, workmanship, and a carefully managed process. The three main control points in this process are **prevention**, **detection**, and **containment**."

Gerard J. Holzmann, ***Landing a Spacecraft on Mars****, IEEE Software, March 2013.*[824]

[824] DOI: 10.1109/MS.2013.32

"Software development teams no longer live - or want to live - in a world of **command and control**. They want to be **self-organizing** and have adaptive, supportive, and **collaborative leadership** guiding them. This new age of management requires managers to build a **culture of trust**, encourage **participation** of their teams in decision making, and **sponsor innovation**. Simply put, managers need to do away with the traditional (micro) management and **share power** with self-organizing teams."

---

*Rashina Hoda, **Power to the People**, IEEE Software, March 2013.*[825]

[825] DOI: 10.1109/MS.2013.34


MAY/JUNE 2013
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
SAFETY-CRITICAL SOFTWARE
Storytelling for Software Professionals // 9
In Defense of Boring // 16
Beyond Data Mining // 92
IEEE
IEEE computer society

"On the one hand, we seek to build software-intensive systems that are **innovative**, **elegant**, and **supremely useful**. On the other hand, computing technology as a thing unto itself is not the place of enduring value, and therefore, as computing fills the spaces of our world, it becomes **boring**. And that's a very good and desirable thing."

---

*Grady Booch, **In Defense of Boring**, IEEE Software, May 2013.*[826]

[826] DOI: 10.1109/MS.2013.54

"**Agent orientation** is moving from its origins in computer science into applied **automation systems engineering**. The main benefit of using software agents in industrial automation is the combined application of agent-oriented software engineering with growing fields such as **semantic technologies**. Software agents also provide **flexibility**, which is often the key requirement for creating software system architectures that can **evolve at runtime**."

Stephan Pech, ***Software Agents in Industrial Automation Systems****, IEEE Software, May 2013.*[827]

[827] DOI: 10.1109/MS.2013.57

"We live in a world in which our **safety depends** on **software-intensive systems**. This is the case for the **aeronautic**, **automotive**, **medical**, **nuclear**, and **railway** sectors as well as many more. Organizations everywhere are struggling to find **cost-effective methods** to deal with the enormous increase in size and complexity of these systems, while simultaneously respecting the need to ensure their safety. Consequently, we're witnessing the ad hoc emergence of a **renewed discipline** of **safety-critical software systems** development as a broad range of software engineering methods, tools, and frameworks are revisited from a safety-related perspective."

Annie Combelles, Xabier Larrucea, John Favaro, ***Safety-Critical Software [Guest editors' introduction]****, IEEE Software, May 2013.*[828]

[828] DOI: 10.1109/MS.2013.55

"The transition from a **code-based process** to a **model-based process** isn't easy. This is particularly true for a company that operates in a **safety-critical sector**, where the products must be developed according to **international standards**, with certified tools and controlled processes."

Stefania Gnesi, Gianluca Magnani, Alessandro Fantechi, Alessio Ferrari, ***Model-Based Development and Formal Methods in the Railway Industry****, IEEE Software, May 2013.*[829]

[829] DOI: 10.1109/MS.2013.44

"Conventional software reliability assessment validates a system's reliability only at the end of development, resulting in costly **defect correction**. A … **statistical model checking** (SMC) … validate reliability at an **early stage**. SMC computes the **probability** that a target system will satisfy **functional-safety** requirements."

Tai-Hyo Kim, Jongmoon Baik, Moonzoo Kim, Okjoo Choi, Youngjoo Kim, ***Validating Software Reliability Early through Statistical Model Checking****, IEEE Software, May 2013.*[830]

[830] DOI: 10.1109/MS.2013.24

"Testing software in **air traffic control systems** costs much more than building them. This is basically true in every domain producing software-intensive critical systems. Software engineers strive to find methodological and process-level solutions to **balance these costs** and to better distribute verification efforts among all development phases."

Stefano Russo, Francesco Fucci, Roberto Pietrantuono, Mauro Faella, Gabriella Carrozza, ***Engineering Air Traffic Control Systems with a Model-Driven Approach****, IEEE Software, May 2013.*[831]

[831] DOI: 10.1109/MS.2013.20

"Software for **commercial aircraft** is subject to the stringent **certification processes** described in the **DO-178B standard**, 'Software Considerations in Airborne Systems and Equipment Certification.' Issued in 1992, this document focuses strongly on the verification process, with a major emphasis on testing. In 2005, the avionics industry initiated an effort to update DO-178B, in large part to accommodate development practices (including formal verification techniques) that had matured since its publication. A revised standard, DO-178C, was issued in late 2011, incorporating new guidance that allows **formal verification** to **replace** certain forms of **testing**."

Benjamin Monate, Emmanuel Ledinot, Herve Delseny, Virginie Wiels, Yannick Moy, ***Testing or Formal Verification: DO-178C Alternatives and Industrial Experience****, IEEE Software, May 2013.*[832]

[832] DOI: 10.1109/MS.2013.43

"To **support any claim** that a **product is safe** for its intended use, manufacturers must **establish traceability** within that product's development life cycle. "

Jane Cleland-Huang, Yi Zhang, Paul L. Jones, Patrick Mader, ***Strategic Traceability for Safety-Critical Projects****, IEEE Software, May 2013.*[833]

[833] DOI: 10.1109/MS.2013.60

“Aerospace or **flight control systems** software development follows a rigorous process according to the **RTCA DO-178B standard**, yet **software errors** still occur.”

Yogananda Jeppu, ***Flight Control Software: Mistakes Made and Lessons Learned***, *IEEE Software, May 2013.*[834]

[834] DOI: 10.1109/MS.2013.42

“Inspired by general ideas about how the **automotive industry** brings **innovation** into lean manufacturing, the author proposes introducing an activity called **software sketchifying** into software product development. Sketchifying aims to stimulate software stakeholders to spend more time generating and considering **alternative ideas** before making a decision to proceed with engineering.”

Zeljko Obrenović, ***Software Sketchifying: Bringing Innovation into Software Development****, IEEE Software, May 2013.*[835]

[835] DOI: 10.1109/MS.2012.71


JULY/AUGUST 2013
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
SOFTWARE ANALYTICS: SO WHAT?
Sustainable Embedded Software // 72
Emerging Metrics for Assessing Software // 99
IEEE
IEEE computer society

"**The fast-changing nature** of our field is one of the things that make working in software so much **fun** — and so **challenging**."

---

*Forrest Shull, **The Only Constant Is Change**, IEEE Software, July 2013.*[836]

[836] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2013.115

"For those on the outside of the **curtain of computing**, there is **much mystery** behind the matter of software-intensive systems. To some, it looks like magic; to most, its inner workings are irrelevant insofar that it simply works. To those of us behind the curtain, however, we know that such systems are filled with **chaos, regularity, and beauty**."

---

*Grady Booch, **The Wonder Years**, IEEE Software, July 2013.*[837]

[837] DOI: 10.1109/MS.2013.89

"Deciding whether to **write portable code** or not should be the outcome of a **cost-benefit analysis**. The key reason to favor portable code is that it opens up the **selection of resources** available to our project. Diverse technology choices free us from vendor lock-in, allowing us to select the best technology in each area based on quality and price, and minimize technology risks. However, portable code can **degrade functionality**, expressiveness, and efficiency."

Diomidis Spinellis, ***Portability: Goodies vs. the Hair Shirt****, IEEE Software, July 2013.*[838]

[838] DOI: 10.1109/MS.2013.82

"Many practitioners and researchers have turned to **analytics**—that is, the use of analysis, data, and systematic reasoning for making decisions. We can define software analytics as follows: '**Software analytics** is **analytics on software data** for managers and software engineers with the aim of empowering software development individuals and teams to **gain and share insight** from their data to make better decisions.'"

Tim Menzies, Thomas Zimmermann, ***Software Analytics: So What?****, IEEE Software, July 2013.*[839]

[839] DOI: 10.1109/MS.2013.86

"**Performance** is a critical component of **customer satisfaction** with network-based applications. Unfortunately, accurately evaluating the **performance of collaborative software** that operates in extremely heterogeneous environments is difficult with traditional techniques such as modeling workloads or testing in controlled environments."

Sandipan Ganguly, Brian Bussone, Christian Bird, Danyel Fisher, Jacqueline Richards, Robert Musson, ***Leveraging the Crowd: How 48,000 Users Helped Improve Lync Performance****, IEEE Software, July 2013.*[840]

[840] DOI: 10.1109/MS.2013.67

"Prominent technology companies including IBM, Microsoft, and Google have embraced an **analytics-driven culture** to help improve their **decision making**. Analytics aim to help practitioners answer questions critical to their projects, such as 'Are we on track to deliver the next release on schedule?' and 'Of the recent features added, which are the most prone to defects?' by providing **fact-based views** about projects. Analytic results are often quantitative in nature, presenting data as **graphical dashboards** with reports and charts."

Olga Baysal, Michael W. Godfrey, Reid Holmes, ***Developer Dashboards: The Need for Qualitative Analytics****, IEEE Software, July 2013.*[841]

[841] DOI: 10.1109/MS.2013.66

"**Defect density** is the ratio between the **number of defects** and **software size**. Properly assessing defect density in evolutionary product development requires a strong tool and rigid process support that enables defects to be traced to the offending source code. In addition, it requires waiting for **field defects** after the product is deployed."

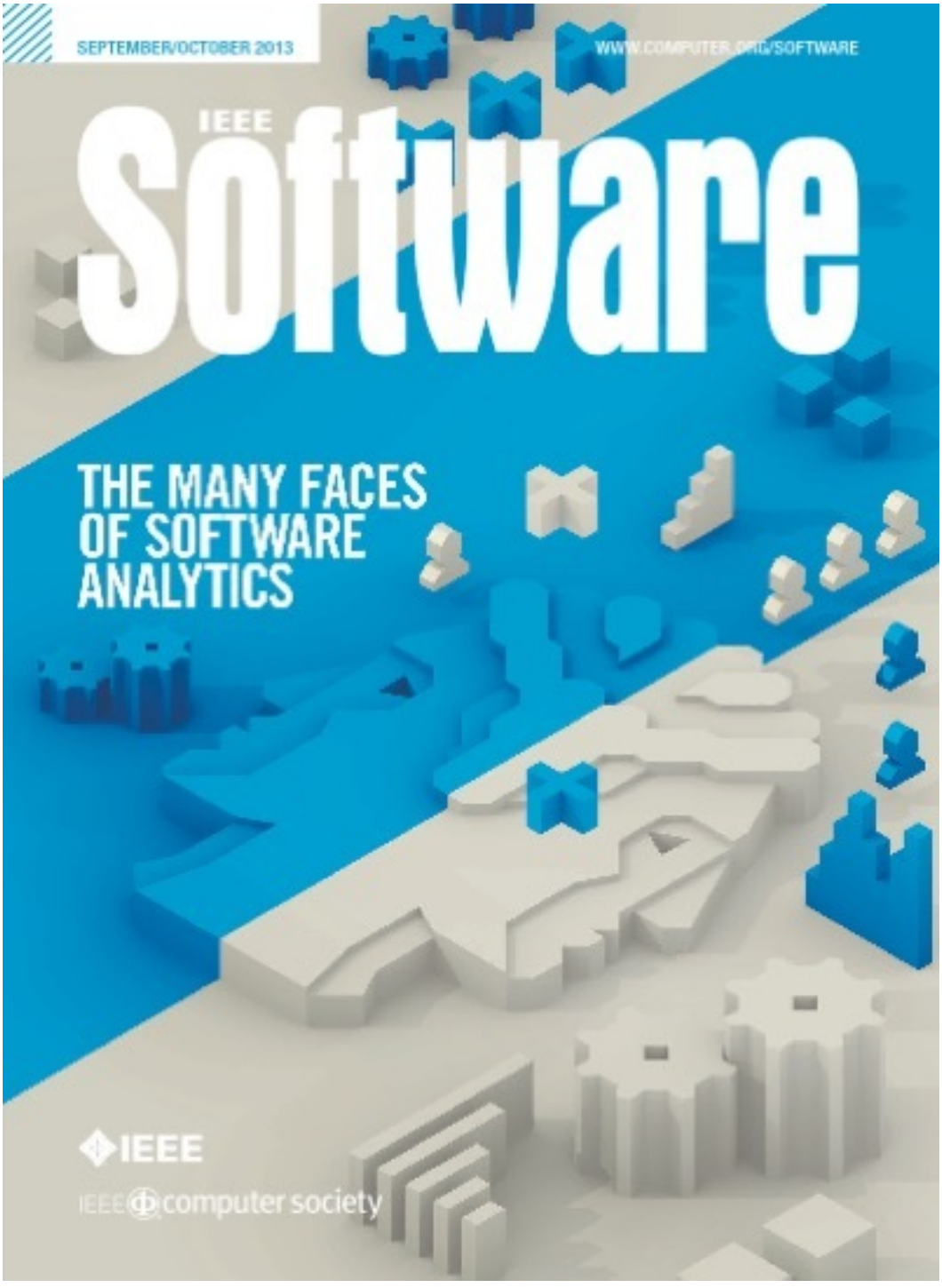
David Faller, Yang-Ming Zhu, ***Defect-Density Assessment in Evolutionary Product Development: A Case Study in Medical Imaging****, IEEE Software, July 2013.*[842]

[842] DOI: 10.1109/MS.2012.111

"Today's software development challenges require **learning teams** that can continuously apply new engineering and management practices, new and complex **technical skills**, **cross-functional skills**, and **experiential lessons** learned. The pressure of delivering working software often forces software teams to **sacrifice learning**-focused practices. Effective **learning under pressure** involves conscious efforts to implement original agile practices such as retrospectives and adapted strategies such as learning spikes. Teams, their management, and customers must all recognize the importance of creating **learning teams** as the key to braving the erratic climates and uncharted territories of future software development."

Jeffry Babb, Rashina Hoda, Jacob Norbjerg, ***Toward Learning Teams****, IEEE Software, July 2013.*[843]

[843] DOI: 10.1109/MS.2013.90


SEPTEMBER/OCTOBER 2013
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
THE MANY FACES OF SOFTWARE ANALYTICS
IEEE
IEEE computer society

"If estimating the time needed for implementing some software is difficult, coming up with a figure for **the time required to debug** it is nigh on impossible."

---

*Diomidis Spinellis, **Differential Debugging**, IEEE Software, September 2013.*[844]

[844] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2013.103

“With **smartphones** being the primary handheld device for more than **a billion people**, mobile Web apps are a necessity in both technical and commercial fields. There are several approaches to developing mobile Web apps, but given the **fast speed** of mobile software evolution, in which the leading companies become marginal in months and new gadgets continually appear, it’s crucial to **understand the basic technologies**. “

---

*Nicolás Serrano, Josune Hernantes, Gorka Gallardo, **Mobile Web Apps**, IEEE Software, September 2013.*[845]

[845] DOI: 10.1109/MS.2013.111

"With **software analytics**, software practitioners explore and analyze data to obtain insightful, actionable information for tasks regarding software development, systems, and users. The **StackMine** project produced a software analytics system for **Microsoft product teams**."

Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, Tao Xie, ***Software Analytics in Practice****, IEEE Software, September 2013.*[846]

[846] DOI: 10.1109/MS.2013.94

"When free, open source software development communities work with companies that use their output, it's especially important for both parties to understand how this collaboration is performing. The use of data analytics techniques on software development repositories can improve **factual knowledge** about **performance metrics**."

Jesus M. Gonzalez-Barahona, Daniel Izquierdo-Cortazar, Stefano Maffulli, Gregorio Robles, ***Understanding How Companies Interact with Free Software Communities****, IEEE Software, September 2013.*[847]

[847] DOI: 10.1109/MS.2013.95

"Amisoft, a **Chilean software company** with **43 employees**, successfully uses **software analytics** in its projects. These support a variety of **strategic and tactical decisions**, resulting in **less overwork** of employees. However, the analytics done at Amisoft are **very different** from the ones used in larger companies."

Romain Robbes, René Vidal, María Cecilia Bastarrica, ***Are Software Analytics Efforts Worthwhile for Small Companies? The Case of Amisoft****, IEEE Software, September 2013.*[848]

[848] DOI: 10.1109/MS.2013.92

"**Software analytics** guide practitioners in **decision making** throughout the software development process. In this context, **prediction models** help managers efficiently organize their resources and identify problems by **analyzing patterns** on existing project data in an intelligent and meaningful manner."

Ayse Tosun Misirli, Ayse Tosun Misirli, Ayse Bener, Burak Turhan, ***A Retrospective Study of Software Analytics Projects: In-Depth Interviews with Practitioners****, IEEE Software, September 2013.*[849]

[849] DOI: 10.1109/MS.2013.93

"As the last standardization effort was done in 2004, the software engineering curriculum is currently being revised. Haven't we reached the point where **agile development** should be **part of all software engineering curricula**? And if so, shouldn't new curriculum standards ensure that it is?"

Armando Fox, David Patterson, ***Is the New Software Engineering Curriculum Agile?****, IEEE Software, September 2013.*[850]

[850] DOI: 10.1109/MS.2013.109


NOVEMBER/DECEMBER 2013
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
ARCHITECTURE
SUSTAINABILITY
Computing and Spirituality // 14
20 Years of Patterns // 88
IEEE
IEEE computer society

"No matter your individual position on the matter, **faith is a powerful element** of the human experience. Therefore, it comes as no surprise that **computing intersects** with the **story of belief** in many ways … computing as a **medium for faith**, as a **ritual space**, and as a technology that itself raises certain **metaphysical issues**."

---

*Grady Booch, **Deus ex Machina**, IEEE Software, November 2013.*[851]

[851] DOI: 10.1109/MS.2013.122

"**Embedded analytics** and statistics for **big data** have emerged as an important topic across industries. As the volumes of data have increased, software engineers are called to support data analysis and applying some kind of statistics to them."

Panos Louridas, Christof Ebert, ***Embedded Analytics and Statistics for Big Data****, IEEE Software, November 2013.*[852]

[852] DOI: 10.1109/MS.2013.125

"**Software architecture** is the foundation of software system development, encompassing a system's architects' and stakeholders' **strategic decisions**."

Paris Avgeriou, Michael Stal, Rich Hilliard, ***Architecture Sustainability [Guest editors' introduction]****, IEEE Software, November 2013.*[853]

[853] DOI: 10.1109/MS.2013.120

"Software architects must **sustain design decisions** to endure throughout software evolution. Several criteria can help them assess decisions' sustainability … **Strategic** … **Measurable and Manageable** … **Achievable and Realistic** … **Rooted in Requirements** … **Timeless** … "

Uwe Zdun, Rafael Capilla, Huy Tran, Olaf Zimmermann, ***Sustainable Architectural Design Decisions****, IEEE Software, November 2013.*[854]

[854] DOI: 10.1109/MS.2013.97

"It's difficult to express a **software architecture's sustainability** in a single metric: relevant information is spread across requirements, architecture design documents, **technology choices**, source code, system context, and software architects' implicit knowledge. Many aspects influence economic sustainability, including design decisions facilitating **evolutionary changes**, adherence to good **modularization practices**, and **technology choices**."

Heiko Koziolek, Dominik Domis, Thomas Goldschmidt, Philipp Vorst, ***Measuring Architecture Sustainability****, IEEE Software, November 2013.*[855]

[855] DOI: 10.1109/MS.2013.101

"**Software product lines (SPLs)** are **long-living systems** that enable **systematic reuse** in application engineering. Product-specific changes over time can result in **architecture drift**, which requires updating assumptions made in the SPL's reuse infrastructure."

Juha Savolainen, Nan Niu, Tommi Mikkonen, Thomas Fogdal, ***Long-Term Product Line Sustainability with Planned Staged Investments****, IEEE Software, November 2013.*[856]

[856] DOI: 10.1109/MS.2013.96

“One of the most notable categories of successful UI development is **form-oriented frameworks** tightly coupled with **relational database** management systems. Essentially, this approach builds a UI for relational database applications by organizing that **interface into forms**, which present values of database fields in the corresponding form controls, such as text boxes, list boxes, check boxes, grids, and so on. Tools and runtime engines support generic navigation through these forms and direct coupling of controls with the back-end data. The developer doesn’t need to take care of data locking, transfer, transformation, and updates: when the user switches to another record in the master part of a master-details form, for example, the mechanism incorporated in the generic form **automatically refreshes** the values in the details part.”

Zarko Mijailovic, Dragan Milicev, ***A Retrospective on User Interface Development Technology****, IEEE Software, November 2013.*[857]

[857] DOI: 10.1109/MS.2013.45

"A **conservative estimate** puts today's number of **published patterns** at more than **7,500**, and growing. "

Gregor Hohpe, Rebecca Wirfs-Brock, Joseph W. Yoder, Olaf Zimmermann K1 architectural knowledge, ***Twenty Years of Patterns' Impact****, IEEE Software, November 2013.*[858]

[858] DOI: 10.1109/MS.2013.135

2014


JANUARY/FEBRUARY 2014
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
NEW PERSPECTIVES ON SOFTWARE QUALITY
Lessons from Healthcare.gov // 4
Voice-Based Machine Translation // 26
IEEE
IEEE computer society

"**Every line of code** represents a **moral decision**; every bit of data collected, analyzed, and visualized has moral implications."

---

*Grady Booch, **The Human and Ethical Aspects of Big Data**, IEEE Software, January 2014.*[859]

[859] DOI: 10.1109/MS.2014.16

“In practice, it’s neither **possible nor meaningful** to characterize software products or projects by a **single quality metric**. “

Ruth Breu, Annie Kuntzmann-Combelles, Michael Felderer, ***New Perspectives on Software Quality [Guest editors’ introduction]****, IEEE Software, January 2014.*[860]

[860] DOI: 10.1109/MS.2014.9

“**Automated GUIs** test application user interfaces and verify their functionalities. However, due to the uncertainty of runtime execution environments, the **device under test (DUT)** might not reproduce GUI operations on time, resulting in test failures.”

Ying-Dar Lin, Edward T.-H. Chu, Shang-Che Yu, Yuan-Cheng Lai, ***Improving the Accuracy of Automated GUI Testing for Embedded Systems****, IEEE Software, January 2014.*[861]

[861] DOI: 10.1109/MS.2013.100

"**Healthcare social networking sites** (HSNSs) provide users with tools and services to easily establish contact with each other around **shared problems** and utilize the **wisdom of crowds** to **attack disease**. The increasing popularity of HSNSs has led to concern over the **privacy of health-related data** published through these websites. The open philosophy of contemporary HSNSs can result in **unauthorized use** and disclosure of sensitive personal health data."

Jingquan Li, ***Data Protection in Healthcare Social Networks****, IEEE Software, January 2014.*[862]

[862] DOI: 10.1109/MS.2013.99

"**Agility** without **objective governance** cannot scale, and governance without agility cannot compete. **Agile methods** are **mainstream**, and software enterprises are adopting these practices in diverse delivery contexts and at enterprise scale. IBM's broad industry experience with agile transformations and deep internal know-how point to **two key principles** to deliver sustained improvements in software business outcomes with higher confidence: **measure and streamline** change **costs**, and **steer with economic governance** and **Bayesian analytics**."

Murray Cantor, Walker Royce, ***Economic Governance of Software Delivery****, IEEE Software, January 2014.*[863]

[863] DOI: 10.1109/MS.2013.102

"**ISO 26262**, a functional-safety standard, uses **Automotive Safety Integrity Levels (ASILs)** to assign safety requirements to automotive-system elements. System designers initially assign ASILs to **system-level hazards** and then allocate them to elements of the refined system architecture. Through ASIL decomposition, designers can divide a function's safety requirements among **multiple components**. However, in practice, manual ASIL decomposition is difficult and produces varying results."

Luis da Silva Azevedo, David Parker, Martin Walker, Yiannis Papadopoulos, Rui Esteves Araujo, ***Assisted Assignment of Automotive Safety Requirements****, IEEE Software, January 2014.*[864]

[864] DOI: 10.1109/MS.2013.118

"The rhetorical question '**do we practice what we preach?**' still seems to be relevant, even a decade after it appeared on the requirements engineering research landscape."

Smita Ghaisas, ***Practicing What We Preach***, *IEEE Software*, *January 2014.*[865]

[865] DOI: 10.1109/MS.2014.10


MARCH/APRIL 2014
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
NEXT-GENERATION
MOBILE
COMPUTING
Hollywood's View of Software // 19
The State of Crowdsourcing // 30
IEEE
IEEE computer society

"The **Healthcare.gov debacle** of 2013 leads many to wonder if a better understanding of the project's requirements could have lessened the impact of the failed launch."

Jane Cleland-Huang, ***Don't Fire the Architect! Where Were the Requirements?***, *IEEE Software, March 2014.*[866]

[866] DOI: 10.1109/MS.2014.34

"Many affordable **cloud-based offerings** that cover software development needs, like **version control**, **issue tracking**, **remote application monitoring**, **localization**, **deployment**, **payment processing**, and **continuous integration**, do away with the setup, maintenance, and user support costs and complexity associated with running such systems in-house. The most **important risks** of cloud-based tools concern **control of the data** stored and the services an organization uses."

Diomidis Spinellis, ***Developing in the Cloud****, IEEE Software, March 2014.*[867]

[867] DOI: 10.1109/MS.2014.33

"Within the **last decade**, **laptop sales** have surpassed those of **desktop computers** in many world markets, and the worldwide popularity of **smartphones** has surpassed them both … But smartphones are not the likely end of the mobile computing innovation vector. We believe **mobile computing is in its infancy**, and the next generations of mobile technology are going to be even more pervasive, smaller, and maybe even a bit weirder and more integral to our lives, jobs, and future."

James Edmondson, William Anderson, Jeff Gray, Joseph P. Loyall, Klaus Schmid, Jules White, ***Next-Generation Mobile Computing****, IEEE Software, March 2014.*[868]

[868] DOI: 10.1109/MS.2014.39

"Researchers from **sociological disciplines** could greatly benefit from **collective information** from the many people who use **mobile devices** to communicate via various social apps and services. However, processing that information is difficult because it's **scattered** among numerous social platforms. Furthermore, **users** are becoming **increasingly concerned** about how and by whom their information is being accessed."

Joaquin Guillen, Javier Miranda, Javier Berrocal, Jose Garcia-Alonso, Juan Manuel Murillo, Carlos Canal, ***People as a Service: A Mobile-centric Model for Providing Collective Sociological Profiles****, IEEE Software, March 2014.*[869]

[869] DOI: 10.1109/MS.2013.140

“Mobile cloud computing infrastructures supporting the vision of the **Internet of Things** (IoT) provide services **beneficial to our society**. For example, a **cloud of smartphones** can run software that shares the sensing and computing resources of nearby devices, providing increased **situational awareness** in a **disaster zone**. “

Tihamer Levendovszky, Abhishek Dubey, William R. Otte, Daniel Balasubramanian, Alessandro Coglio, Sandor Nyako, William Emfinger, Pranav Kumar, Aniruddha Gokhale, Gabor Karsai, ***Distributed Real-Time Managed Systems: A Model-Driven Distributed Secure Information Architecture Platform for Managed Embedded Systems****, IEEE Software, March 2014.*[870]

[870] DOI: 10.1109/MS.2013.143

"Newer models for interacting with wireless sensors such as **Internet of Things** and **Sensor Cloud** aim to overcome restricted resources and efficiency. The Missouri S&T (science and technology) sensor cloud enables different networks, spread in a huge geographical area, to connect together and be employed simultaneously by multiple users on demand. **Virtual sensors**, which are at the core of this sensor cloud architecture, assist in creating a multiuser environment on top of resource-constrained physical wireless sensors and can help in supporting multiple applications."

Sanjay Madria, Vimal Kumar, Rashmi Dalvi, ***Sensor Cloud: A Cloud of Virtual Sensors****, IEEE Software, March 2014.*[871]

[871] DOI: 10.1109/MS.2013.141

“Developers of **embedded systems** are driven to constantly improve product quality, reduce cost, and rapidly deliver reliable working code. The embedded software domain applies constraints which can **hinder agile methodologies** commonly used to achieve such benefits. **Simulation-based software development** is one proven method that addresses these constraints.”

Jason Ard, Kristine Davidsen, Terril Hurst, ***Simulation-Based Embedded Agile Development****, IEEE Software, March 2014.*[872]

[872] DOI: 10.1109/MS.2014.42


MAY/JUNE 2014
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
GREEN SOFTWARE
Return of the Pragmatic Architect // 10
The Incredible Lightness of Software // 88
IEEE
IEEE computer society

“Many **types of architects** work in the software industry, but when we consider the breadth of their work and their primary expertise, we find that they can be organized into three major groups: **enterprise architects**, **application architects**, and **infrastructure architects**. Knowing which group an architect falls into helps in understanding their expertise and what to expect of them.”

---

*Eoin Woods, **Return of the Pragmatic ArchitectK1 infrastructure**, IEEE Software, May 2014.*[873]

[873] DOI: 10.1109/MS.2014.69

“**Continuous integration** has been around for a while now, but the habits it suggests are **far from common practice**. **Automated builds**, a **thorough test suite**, and committing to **the mainline branch** every day sound simple at first, but they require a **responsible team** to implement and constant care. What starts with improved tooling can be **a catalyst** for long-lasting change in your company’s shipping culture. Continuous integration is more than a set of practices, it’s a mindset that has one thing in mind: **increasing customer value**.”

Mathias Meyer, ***Continuous Integration and Its ToolsK1 testing****, IEEE Software, May 2014.*[874]

[874] DOI: 10.1109/MS.2014.58

"**Quality goals** for security, business agility, maintainability and other such attributes are often achieved through implementing best practices. To know which **stakeholder goals** are attainable and how they can best be achieved, we must **empirically evaluate** software development beliefs and practices."

Mamoun Hirzalla, Peter Bahrs, Jane Cleland-Huang K1 quality goals, ***Beyond Anecdotal Thinking: Deepening Our Understanding for Achieving Quality Goals****, IEEE Software, May 2014.*[875]

[875] DOI: 10.1109/MS.2014.57

“Most studies and regulatory controls focus on **hardware-related measurement**, analysis, and control for **energy consumption**. However, all forms of hardware include **significant software components**. Although software systems don’t consume energy directly, they affect **hardware utilization**, leading to **indirect energy consumption**. Therefore, it’s important to engineer software to optimize its energy consumption. “

---

*Ayse Basar Bener, Maurizio Morisio, Andriy Miranskyy K1 cloud, **Green Software**, IEEE Software, May 2014.*[876]

[876] DOI: 10.1109/MS.2014.62

“Many software systems today control **large-scale sociotechnical systems**. These systems aren’t just entangled with the environment but also with our dwindling resources and mostly unsustainable way of living, while the planet’s population continues to grow. Dealing with **sustainability requirements** and systematically supporting their elicitation, analysis, and realization is a problem that has yet to be solved.”

Birgit Penzenstadler, Ankita Raturi, Debra Richardson, Bill Tomlinson, ***Safety, Security, Now Sustainability: The Nonfunctional Requirement for the 21st Century****, IEEE Software, May 2014.*[877]

[877] DOI: 10.1109/MS.2014.22

“**Energy efficiency** and other **sustainability issues** are common concerns in the material production industries but rarely addressed in software development efforts. Instead, traditional software development life cycles and methodologies place an emphasis on maintainability and other intrinsic software quality features. One standard practice is to **improve maintainability** by detecting **bad smells** in a system’s architecture and then applying **refactoring transformations** to deal with those smells. The refactoring research area is sufficiently mature for most techniques to achieve more maintainable system architectures, but … they can also lead to both **decreased sustainability** and **increased power consumption**.”

Ricardo Perez-Castillo, Mario Piattini, ***Analyzing the Harmful Effect of God Class Refactoring on Power Consumption****, IEEE Software, May 2014.*[878]

[878] DOI: 10.1109/MS.2014.23

"To develop more powerful, service-specific strategies for **reducing IT's carbon footprint**, we need more complete and widely **understandable specifications** that describe exactly a service's functionality, the level of quality it achieves, and its **environmental consequences**. Such green specifications will allow more stakeholders involved in the delivery and consumption of IT services to understand their detailed functionality and the **tradeoffs** between **quality of service** and **environmental impact** entailed in their use."

Colin Atkinson, Thomas Schulze, Sonja Klingert, ***Facilitating Greener IT through Green Specifications****, IEEE Software, May 2014.*[879]

[879] DOI: 10.1109/MS.2014.19

"In applications in which **embedded devices** cooperate with **ICT (information and communication technology)** systems to make industrial processes more efficient, reduce waste or raw materials, and save the environment, the concept of **green software** becomes increasingly complex. To deal with this issue, the green-software community has introduced the concepts of **greening ICT** or **greening through ICT**."

Krzysztof Sierszecki, Tommi Mikkonen, Michaela Steffens, Thomas Fogdal, Juha Savolainen K1 software engineering, ***Green Software: Greening What and How Much?****, IEEE Software, May 2014.*[880]

[880] DOI: 10.1109/MS.2014.63

“Hardware and software engineers are instrumental in developing **energy-efficient mobile systems**. Unfortunately, the **last mile of energy** efficiency relies on end users’ choices and requirements. Imagine a user who has no power outlet access and must remain productive on the laptop’s battery. How does this person maximize battery life, yet remain productive? What does the user have to give up to keep working?”

Chenlei Zhang, Abram Hindle, Daniel M. German, ***The Impact of User Choice on Energy Consumption****, IEEE Software, May 2014.*[881]

[881] DOI: 10.1109/MS.2014.27


JULY/AUGUST 2014
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
Moods, Emotions, and Performance // 24
"REFLECTIVE SOFTWARE ENGINEER"
The Case of Drupal and Twitter // 72
IEEE
computer society

"Studies show that software **developers' happiness** pays off when it comes to **productivity**."

Daniel Graziotin, Xiaofeng Wang, Pekka Abrahamsson,

***Software Developers, Moods, Emotions, and Performance**,*

IEEE Software, July 2014.[882]

[882] DOI: 10.1109/MS.2014.94

"The **capacity to reflect** on past practice is important for **continuous learning** in software development. Reflection often takes place **in cycles** of **experience** followed by **conscious application** of learning from that experience, during which a software developer might explore comparisons, ponder alternatives, take diverse perspectives, and draw inferences, especially in new and/or complex situations. Such reflective practice has been shown in different disciplines to be an **effective developmental practice** for **organizations**, for **teams**, and for **individuals**."

Tore Dyba, Neil Maiden, Robert Glass K1 practitioners, ***The Reflective Software Engineer: Reflective Practice****, IEEE Software, July 2014.*[883]

[883] DOI: 10.1109/MS.2014.97

"The capacity to **reflect on past practice** is important for continuous learning in software development. Reflection often takes place in cycles of experience followed by conscious application of learning from that experience, during which a software developer might explore comparisons, ponder alternatives, take diverse perspectives, and draw inferences, especially in new and/or complex situations."

Tore Dyba, Neil Maiden, Robert Glass, ***The Reflective Software Engineer: Reflective Practice****, IEEE Software, July 2014.*[884]

[884] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2014.97

"**Project retrospectives** can be powerful tools for project teams to collectively identify communication gaps and practices to improve for future projects. However, even if project members take the time for a retrospective, it can be hard to **correctly remember** and **jointly discuss** past events in a constructive way. **Fact-based timelines** that visualize a project's events offer a possible solution."

Elizabeth Bjarnason, Anne Hess, Richard Berntsson Svensson, Bjorn Regnell, Joerg Doerr, ***Reflecting on Evidence-Based Timelines****, IEEE Software, July 2014.*[885]

[885] DOI: 10.1109/MS.2014.26

"**Learning** is a **lifelong process**, especially in the fast-paced software industry. In addition to **formal training** courses, good software developers continually **learn by reflecting** on what they've done in the past. However, reflective practice is **rarely taught** explicitly in university software engineering education. One way to teach reflective techniques from the start is through **studio-based learning**."

Christopher N. Bull, Jon Whittle, ***Supporting Reflective Practice in Software Engineering Education through a Studio-Based Approach****, IEEE Software, July 2014.*[886]

[886] DOI: 10.1109/MS.2014.52

"The theoretical underpinnings of **agile principles** emphasize **regular reflection** as a means to attain a **sustainable development pace** and continuous learning. In practice, high **iteration pressure** can diminish opportunities for ongoing learning and reflection threatening to deprive software teams of learning and reflection and possibly stagnating process evolution."

Jeffry Babb, Rashina Hoda, Jacob Norbjerg, ***Embedding Reflection and Learning into Agile Software Development****, IEEE Software, July 2014.*[887]

[887] DOI: 10.1109/MS.2014.54

"A **coderetreat** is an event where software developers gather to **spend a day exploring** their craft in an informal yet intellectually challenging environment. It encourages **reflective practice** by addressing **a single programming problem** from different perspectives, with multiple coding partners, freed from the daily pressures of deadlines and the need to deliver completed artifacts."

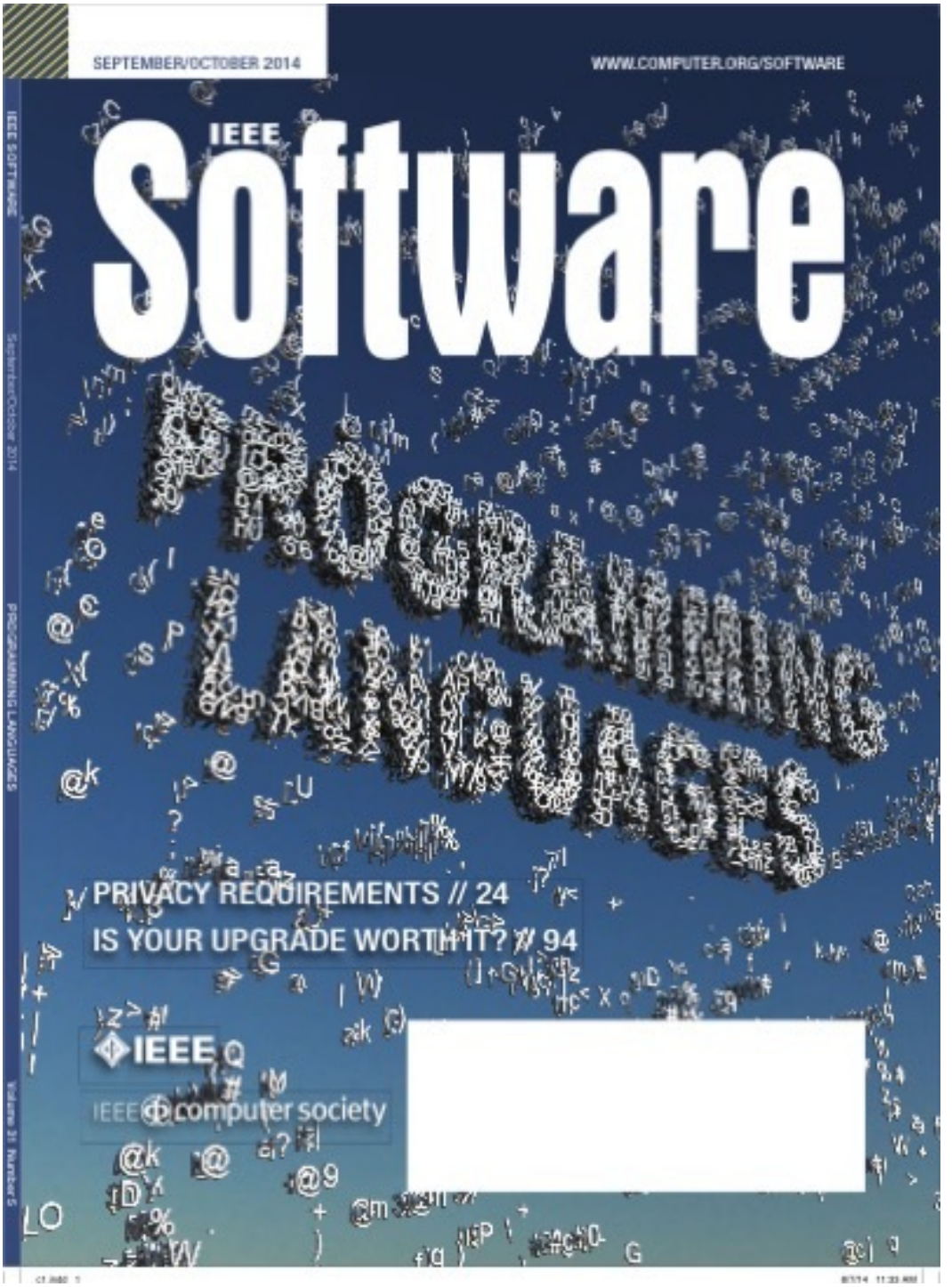
David Parsons, Anuradha Mathrani, Teo Susnjak, Arno Leist, ***Coderetreats: Reflective Practice and the Game of Life****, IEEE Software, July 2014.*[888]

[888] DOI: 10.1109/MS.2014.25

"**Microblogging** is a popular form of **social media** that has quickly permeated both enterprise and open source communities. However, exactly how open source communities can leverage microblogging isn't yet well understood. … how **Drupal's** open source community uses **Twitter** … community members often express **positive emotions** when tweeting about work, which reinforces a sense of community."

Xiaofeng Wang, Ilona Kuzmickaja, Klaas-Jan Stol, Pekka Abrahamsson, Brian Fitzgerald, ***Microblogging in Open Source Software Development: The Case of Drupal and Twitter****, IEEE Software, July 2014.*[889]

[889] DOI: 10.1109/MS.2013.98


SEPTEMBER/OCTOBER 2014
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
PROGRAMMING LANGUAGES
PRIVACY REQUIREMENTS // 24
IS YOUR UPGRADE WORTH IT? // 94
IEEE computer society
IEEE SOFTWARE
PROGRAMMING LANGUAGES

"**Privacy** is a **critical design principle** that must be balanced with how we **utilize personal data** in software. … the increasing importance of privacy in emerging **software ecosystems**, legal and standards **compliance**, and software design practice."

Travis Breaux, ***Privacy Requirements in an Age of Increased Sharing****, IEEE Software, September 2014.*[890]

[890] DOI: 10.1109/MS.2014.118

"An impressive number of **new startups** are launched every day as a result of growing new markets, accessible technologies, and venture capital. New ventures such as **Facebook**, **Supercell**, **Linkedin**, **Spotify**, **WhatsApp**, and **Dropbox**, to name a few, are good examples of startups that evolved into successful businesses. However, despite many successful stories, the **great majority** of them **fail** prematurely. Operating in a **chaotic** and **rapidly evolving domain** conveys new uncharted challenges for startuppers."

Carmine Giardino, Michael Unterkalmsteiner, Nicolo Paternoster, Tony Gorschek, Pekka Abrahamsson, ***What Do We Know about Software Development in Startups?****, IEEE Software, September 2014.*[891]

[891] DOI: 10.1109/MS.2014.129

"One thing we know for certain is that the **dominant programming language** of today is the **legacy language** of tomorrow. Sometimes languages are sidelined due to **fashion**, but changes are generally due to **new languages** being applicable to a wider or different class of problems than their predecessors. Maybe one day this process will stop, but it seems unlikely that you'd lose money betting on it to continue for a while yet."

Laurence Tratt, Adam Welc K1 software engineering, ***Programming Languages****, IEEE Software, September 2014.*[892]

[892] DOI: 10.1109/MS.2014.119

"**IDEs** are essential for programming language developers, and state-of-the-art IDE support is mandatory for programming languages to be successful. Although IDE features for mainstream programming languages are typically implemented manually, this often isn't feasible for programming languages that must be developed with significantly fewer resources. The **Spoofax language workbench** is a platform for developing textual programming languages with state-of-the-art IDE support."

Guido H. Wachsmuth, Gabriël D.P. Konat, Eelco Visser,

***Language Design with the Spoofax Language Workbench**,*

IEEE Software, September 2014.[893]

[893] DOI: 10.1109/MS.2014.100

"**Scripting languages** are very popular and are being used to implement a wide range of applications. Meanwhile, multi-core processors are everywhere, from dektop computers to mobile devices, and concurrency has become the only means to improve performance. However, **concurrent programming** remains difficult and despite some interest in researching new concurrency models for compiled languages, the conventional concurrency support in scripting languages is still lacking."

Alexandre Skyrme, Noemi Rodriguez, Roberto Ierusalimschy, ***Scripting Multiple CPUs with Safe Data Sharing****, IEEE Software, September 2014.*[894]

[894] DOI: 10.1109/MS.2014.102

"**Modern software** differs significantly from **traditional** computer applications that mostly process reasonably small amounts of **static input data**-sets in batch mode. Modern software increasingly processes **massive amounts of data**, whereby it is also often the case that new input data is produced and/or existing data is **modified on the fly**. Consequently, **programming models** that facilitate the development of such software are emerging. What characterizes them is that **data**, respectively changes thereof, **implicitly flow** through computation modules."

Guido Salvaneschi, Patrick Eugster, Mira Mezini, ***Programming with Implicit Flows****, IEEE Software, September 2014.*[895]

[895] DOI: 10.1109/MS.2014.101

"In large-scale software development, there is typically **a conflict** between being responsive to **individual customers**, while at the same time achieving scale in terms of delivering a high number of features to a **large customer base**. Most often, organizations **focus on scale** and individual customer requests are viewed as problematic since they **add complexity** to product variation and version control."

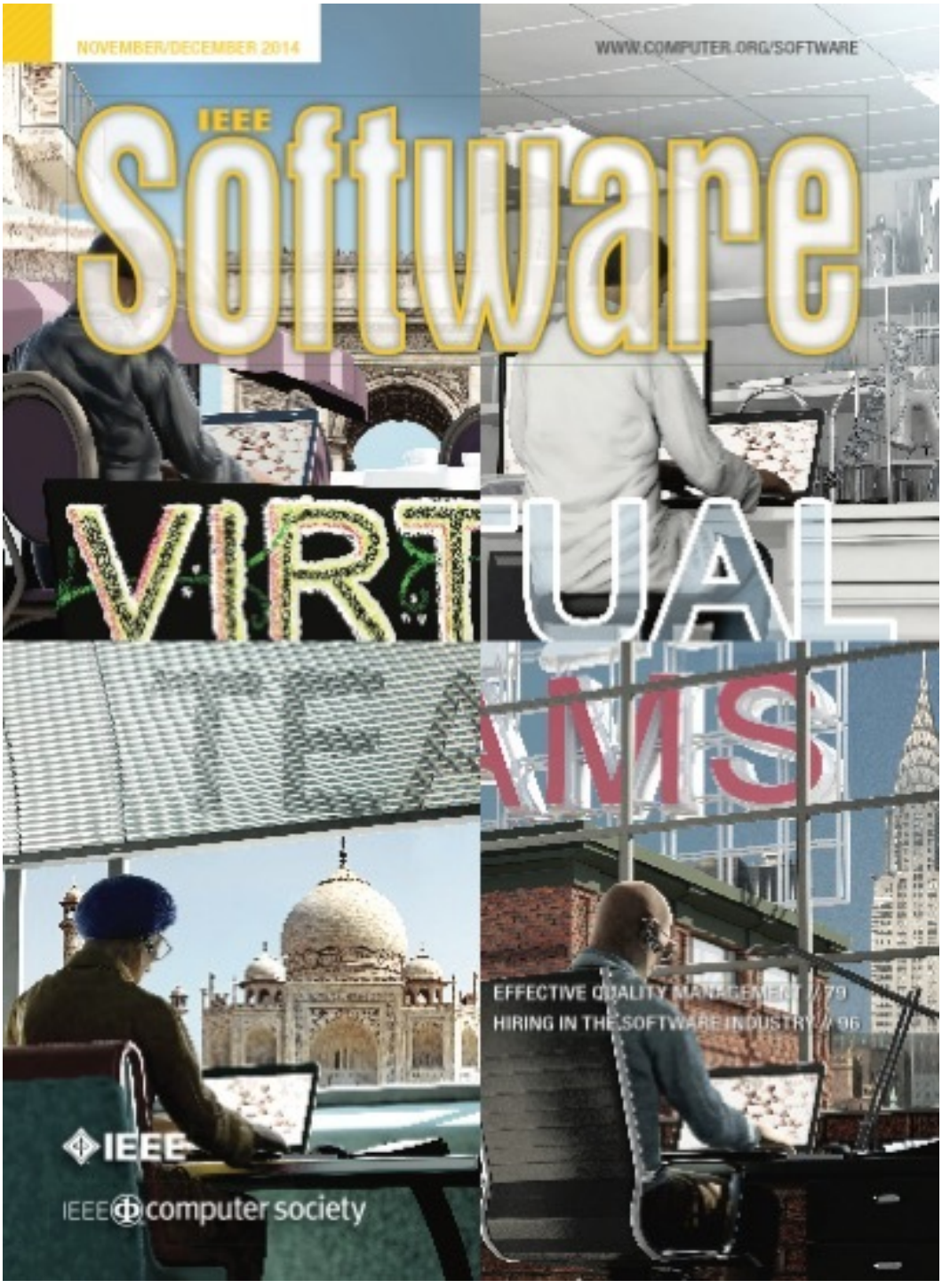
Helena Olsson, Anna Sandberg, Jan Bosch, Hiva Alahyari, ***Scale and Responsiveness in Large-Scale Software Development****, IEEE Software, September 2014.*[896]

[896] DOI: 10.1109/MS.2013.139

“There’s a joke that **Go** was conceived while **waiting for** a **C++** program **to compile**, which is kind of half true. “

Jeff Meyerson, ***The Go Programming LanguageK1 syntax****, IEEE Software, September 2014.*[897]

[897] DOI: 10.1109/MS.2014.127


NOVEMBER/DECEMBER 2014
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
VIRTUAL
AIMS
EFFECTIVE QUALITY MANAGEMENT // 79
HIRING IN THE SOFTWARE INDUSTRY // 96
IEEE
IEEE computer society

“Over the past decades, today, and in the future, business contexts in software organizations and the common ways of developing software are changing dramatically. Formation of **teams in distributed environments**, virtual or not, calls for new ways of working across **geographic**, **temporal**, and **cultural boundaries**. This, however, also requires **effective leadership** approaches enabled through systems, processes, technology, and people.”

Darja Smite, Marco Kuhrmann, Patrick Keil, ***Virtual Teams [Guest editors’ introduction]****, IEEE Software, November 2014.*[898]

[898] DOI: 10.1109/MS.2014.149

“Software engineering is a field in which distributed development through **virtual teams** is a **fact of life**. … a set of eight **core requirements** for support environments for virtual software teams … (1) Enable **Unobtrusive Awareness** Information Exchange … (2) Make Basic Work-Related **Data Available** … (3) Provide **Multisource Data** Combinations … (4) **Filter** Irrelevant Information … (5) Represent and Recognize **Current Contexts** of Team Members … (6) Support the **Overhearing** of Conversations … (7) Support **Mood Sharing** … (8) … “

Kevin Dullemond, Ben van Gameren, Rini van Solingen, ***Collaboration Spaces for Virtual Software Teams****, IEEE Software, November 2014.*[899]

[899] DOI: 10.1109/MS.2014.105

"In today's world, many companies turn to open source projects as a method to increase productivity and innovation. A major challenge with managing this kind of development is the **onboarding of new developers** into the virtual teams that drive such projects. There's little guidance on how to initiate new members into such teams and how to overcome the learning curve. This case study on open source software projects shows that **mentoring** can have a significant impact on onboarding new members into virtual software development teams."

Fabian Fagerholm, Alejandro Sanchez Guinea, Jay Borenstein, Jurgen Munch, ***Onboarding in Open Source Projects****, IEEE Software, November 2014.*[900]

[900] DOI: 10.1109/MS.2014.107

“Do I think that there are some universals? Absolutely I do. And looking for **a team or cultural fit**, looking for people who are **motivated** and have good **communication** and good **collaboration**, my suspicion is that those are universal qualities that make people successful.”

Tobias Kaatz, ***Hiring in the Software Industry***, *IEEE Software, November 2014.*[901]

[901] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2014.140

2015


JANUARY/FEBRUARY 2015
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
MEANINGFUL INDUSTRIAL—ACADEMIC PARTNERSHIPS 18
MOBILE MONEY IN TANZANIA 29
IEEE
IEEE computer society

"Parallels exist between the **Industrial Revolution** and our current computing revolution regarding **risk**, **transparency**, and **responsibility**."

Grady Booch, ***Of Boilers, Bit, and Bots***, *IEEE Software, January 2015.*[902]

[902] DOI: 10.1109/MS.2015.13

"**Software-driven industries** are advancing in five dimensions: **collaboration**, **comprehension**, **connectivity**, **cloud**, and **convergence**. However, companies often can get stuck in an overly narrow technology focus. To avoid this, they should **connect architecture and functionality**, master the entire software development life cycle, strengthen globally distributed teams, and streamline development."

---

*Christof Ebert, Gerd Hoefner, Mani V.S., **What Next? Advances in Software-Driven Industries**, IEEE Software, January 2015.*[903]

[903] DOI: 10.1109/MS.2015.21

"**Software engineering** for **Internet computing** involves the architecting, development, deployment, management, and quality assurance of software supporting Internet-based systems. It also addresses **global-development issues** such as **communication complexity**, **distributed control**, **governance policies**, and **cultural differences**."

Antonia Bertolino, M. Brian Blake, Pankaj Mehra, Hong Mei, Tao Xie, ***Software Engineering for Internet Computing: Internetware and Beyond [Guest editors' introduction]****, IEEE Software, January 2015.*[904]

[904] DOI: 10.1109/MS.2015.16

"The **Internet of Things (IoT)** has the strong potential to support a human society interacting more symbiotically with its **physical environment**. Indeed, the emergence of tiny devices that sense environmental cues and trigger actuators after consulting logic and human preferences promises a **more environmentally** aware and less wasteful society."

Patrick Eugster, Vinaitheerthan Sundaram, Xiangyu Zhang, ***Debugging the Internet of Things: The Case of Wireless Sensor Networks****, IEEE Software, January 2015.*[905]

[905] DOI: 10.1109/MS.2014.132

"**System operations** (such as deployment, upgrade, and reconfiguration) **for cloud** applications are **failure prone**. These failures occur because these operations are performed through **cloud APIs** provided by cloud providers and because cloud APIs, in turn, are **failure prone**."

Qinghua Lu, Xiwei Xu, Len Bass, Liming Zhu, Weishan Zhang, ***A Tail-Tolerant Cloud API Wrapper****, IEEE Software, January 2015.*[906]

[906] DOI: 10.1109/MS.2015.2

“**Microservices** is the coming together of a bunch of better practices from a number of different communities. It is a combination of great stuff from the **domain-driven-design** community around **strategic design**, **bounded context**, subdomains, how to separate out your domains, and how to partition a very big problem domain into smaller domains so that you can manage them. “

---

*Johannes Thones, **Microservices**, IEEE Software, January 2015.*[907]

[907] DOI: 10.1109/MS.2015.11


MARCH/APRIL 2015
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
RELEASE
CODE INFLATION // 10
MANAGING TECHNICAL DEBT // 22
IEEE
computer society

"**Technical debt** refers to maintenance obligations that software teams accumulate as a result of their actions. Empirical research has led researchers to suggest **three dimensions** along which software development teams should map their technical-debt metrics: **customer satisfaction** needs, **reliability** needs, and the **probability of technology disruption**."

Narayan Ramasubbu, Chris F. Kemerer, C. Jason Woodard, ***Managing Technical Debt: Insights from Recent Empirical Evidence****, IEEE Software, March 2015.*[908]

[908] DOI: 10.1109/MS.2015.45

"A **refactoring** aims to **improve a certain quality** while **preserving others**. For example, **code refactoring** restructures code to make it more maintainable without changing its observable behavior."

Olaf Zimmermann, ***Architectural Refactoring: A Task-Centric View on Software Evolution****, IEEE Software, March 2015.*[909]

[909] DOI: 10.1109/MS.2015.37

“The concept of **cloud computing** has existed for **50 years**, since the beginning of the Internet. **John McCarthy** devised the idea of **time-sharing** in computers as a utility in 1957. Since then, the concept’s name has undergone several changes: from **service bureau**, to application **service provider**, to the **Internet as a service**, to **cloud computing**, and to **software-defined datacenters**, with each name having different nuances. “

Nicolas Serrano, Gorka Gallardo, Josune Hernantes, ***Infrastructure as a Service and Cloud Technologies****, IEEE Software, March 2015.*[910]

[910] DOI: 10.1109/MS.2015.43

"**Mobile deployments** are **more challenging** than Web deployments because we **don't own the ecosystem**, so we can't do all the things that we would normally do. And the canaries are huge. We watch **cold start**, **warm start**, the **app size**, and the numbers of photos uploaded, comments, and ads being displayed or clicked. Growth and **engagement numbers** and the **crash rate** are important to the company. If the crash rate fluctuates, we immediately take action to understand why."

Bram Adams, Stephany Bellomo, Christian Bird, Tamara Marshall-Keim, Foutse Khomh, Kim Moir, ***The Practice and Future of Release Engineering: A Roundtable with Three Release Engineers****, IEEE Software, March 2015.*[911]

[911] DOI: 10.1109/MS.2015.52

"**Continuous delivery** (CD) has emerged as an auspicious **alternative** to **traditional release engineering**, promising to provide the capability to release valuable software continuously to customers."

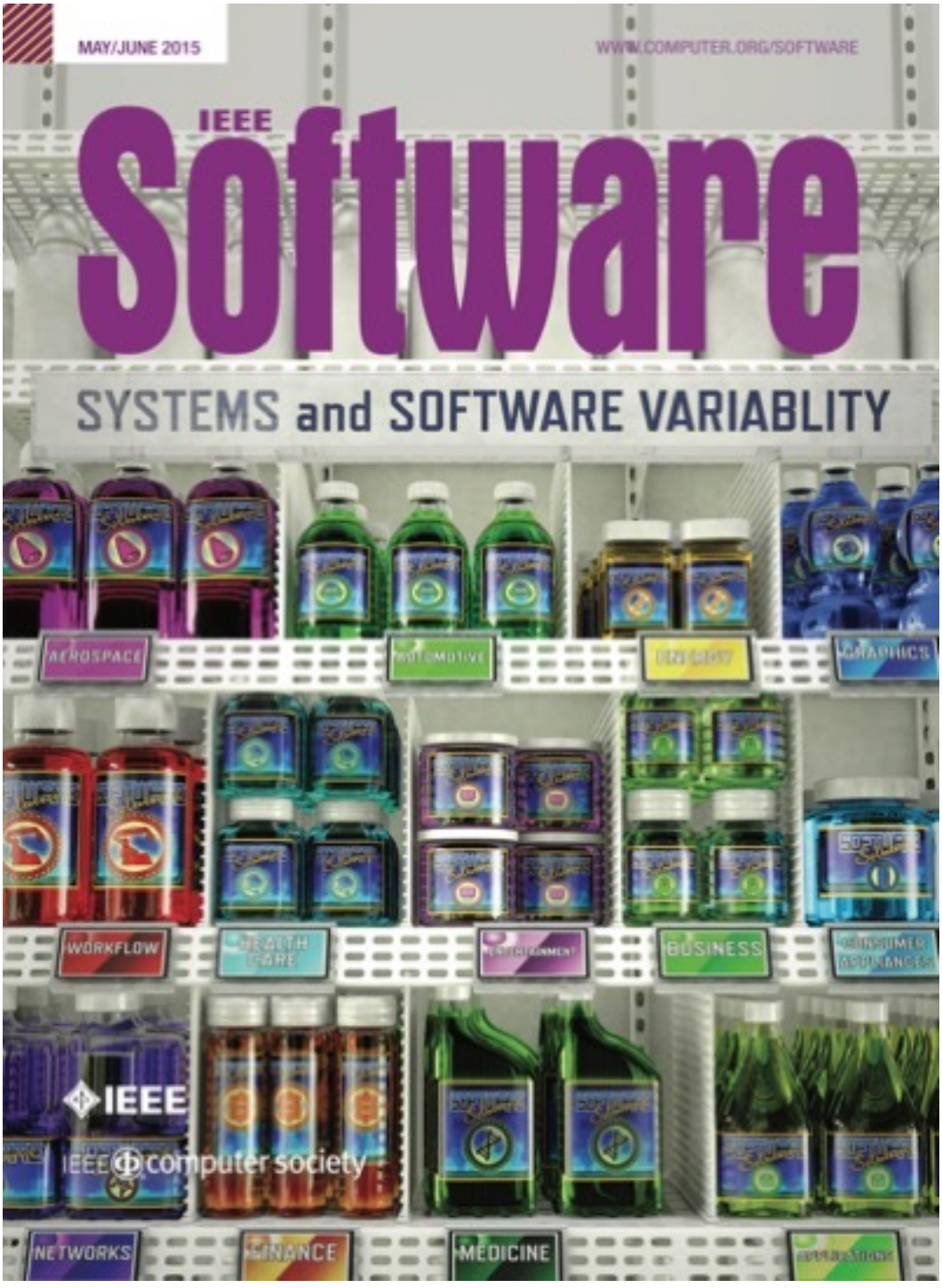
Lianping Chen, ***Continuous Delivery: Huge Benefits, but Challenges Too****, IEEE Software, March 2015.*[912]

[912] DOI: 10.1109/MS.2015.27

"**Continuous delivery** and deployment are dramatically shortening release cycles from months to hours. Cloud applications with **high-frequency releases** often rely heavily on automated tools and **cloud infrastructure APIs** to deploy new software versions."

Liming Zhu, Donna Xu, An Binh Tran, Xiwei Xu, Len Bass, Ingo Weber, Srini Dwarakanathan, ***Achieving Reliable High-Frequency Releases in Cloud Environments****, IEEE Software, March 2015.*[913]

[913] DOI: 10.1109/MS.2015.23


MAY/JUNE 2015
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
SYSTEMS and SOFTWARE VARIABLITY
AEROSPACE
AUTOMOTIVE
GRAPHICS
WORKFLOW
HEALTH CARE
BUSINESS
IEEE
IEEE computer society
NETWORKS
FINANCE
MEDICINE

"Recently proposed **model repositories** aim to support specific needs–for example, **collaborative modeling**, the ability to use different modeling tools in software life-cycle management, tool interoperability, increased model reuse, and the integration of heterogeneous models."

Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, Alfonso Pierantonio, ***Collaborative Repositories in Model-Driven Engineering [Software Technology]****, IEEE Software, May 2015.*[914]

[914] DOI: 10.1109/MS.2015.61

"An **architecture haiku** aims to capture software system architecture's most important details on **a single piece of pape**r. An architecture haiku helps development teams focus on the most essential information relevant to the architecture, provides clear guidance for construction, and encourages collaboration."

Michael Keeling, ***Architecture Haiku: A Case Study in Lean Documentation [The Pragmatic Architect]****, IEEE Software, May 2015.*[915]

[915] DOI: 10.1109/MS.2015.59

"In **variant-rich workflow-based systems**, a major concern for process variability is the context-aware configuration of the variants. This means that context information, not users, drives process configuration. "

Aitor Murguzur, Salvador Trujillo, Hong-Linh Truong, Schahram Dustdar, Oscar Ortiz, Goiuria Sagardui, ***Run-Time Variability for Context-Aware Smart Workflows****, IEEE Software, May 2015.*[916]

[916] DOI: 10.1109/MS.2015.70

"**Smart manufacturing** networks describe a production chain as a marketplace that delivers **products on demand**. In this chain, partners collaborate in product work routings that connect dispersed service-enabled systems with **resources**, **materials**, **human expertise**, and operation-equipment **combinations**."

Michael P. Papazoglou, Willem-Jan van den Heuvel, Julien Etienne Mascolo, ***A Reference Architecture and Knowledge-Based Structures for Smart Manufacturing Networks****, IEEE Software, May 2015.*[917]

[917] DOI: 10.1109/MS.2015.57

"**Defect taxonomies** collect and organize experts' domain knowledge and project experience and are valuable for requirements-based testing. They provide **systematic backup** for the test design, support decisions for allocating testing resources, improve the review of requirements, and are suitable for measuring product quality."

Michael Felderer, Armin Beer, ***Using Defect Taxonomies for Testing Requirements****, IEEE Software, May 2015.*[918]

[918] DOI: 10.1109/MS.2014.56

"**Docker** is a container virtualization technology. So, it's like a very **lightweight virtual machine** VM. In addition to building containers, we provide what we call **a developer workflow**, which is really about helping people build containers and applications inside containers and then share those among their teammates."

Charles Anderson, ***Docker [Software engineering]****, IEEE Software, May 2015.*[919]

[919] DOI: 10.1109/MS.2015.62


JULY/AUGUST 2015
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
Security and Privacy on the Web
DOES OFFSHORING SAVE MONEY? // 26
ARCHITECTING IN THE GAPS // 33
IEEE
IEEE computer society

"The SPOT (**Single Point of Truth**) principle says that developers should specify **key pieces of information** in one and **only one place** in their code. Unfortunately, they frequently violate this principle."

---

*Gerard J. Holzmann, **Points of Truth**, IEEE Software, July 2015.*[920]

[920] DOI: 10.1109/MS.2015.103

"Software enables every aspect of the Web. Everything from device communication to online social networks is achievable only because of multiple lines of code. For various reasons, designing and building **security and privacy** into Web software is often **an afterthought** for most developers. This results in easily **compromised systems** that pose significant privacy and security risks to users."

Tyrone Grandison, Larry Koved, ***Security and Privacy on the Web [Guest editors' introduction]****, IEEE Software, July 2015.*[921]

[921] DOI: 10.1109/MS.2015.86

"Large organizations often face difficult tradeoffs in **balancing** the need to **share information** with the need to **safeguard sensitive data**. A prominent way to deal with this tradeoff is **on-the-fly screen** masking of sensitive data in applications."

Abigail Goldsteen, Ksenya Kveler, Tamar Domany, Igor Gokhman, Boris Rozenberg, Ariel Farkash, ***Application-Screen Masking: A Hybrid Approach****, IEEE Software, July 2015.*[922]

[922] DOI: 10.1109/MS.2015.75

"**Adversaries** employ sophisticated **fingerprinting** techniques to **identify Web users** and record their browsing history and Web interactions. Fingerprinting leaves no footprint on the browser and is invisible to general Web users, who often lack basic knowledge of it."

Amin Faiz Khademi, Mohammad Zulkernine, Komminist Weldemariam, ***An Empirical Evaluation of Web-Based Fingerprinting****, IEEE Software, July 2015.*[923]

[923] DOI: 10.1109/MS.2015.77

“**Inner source**, the adoption and tailoring of open source development practices in organizations, is receiving increased interest.”

Klaas-Jan Stol, Brian Fitzgerald, ***Inner Source–Adopting Open Source Development Practices in Organizations: A Tutorial****, IEEE Software, July 2015.*[924]

[924] DOI: 10.1109/MS.2014.77

"Formal documentation can be a crucial resource for learning to how to use an **API**. However, producing **high-quality documentation** can be nontrivial. … The three severest problems were **ambiguity**, **incompleteness**, and **incorrectness** of content."

Gias Uddin, Martin P. Robillard, ***How API Documentation Fails****, IEEE Software, July 2015.*[925]

[925] DOI: 10.1109/MS.2014.80

"**Software engineers** today must be **lifelong learners** or risk finding themselves **out of a job**, with totally obsolete skills to sell."

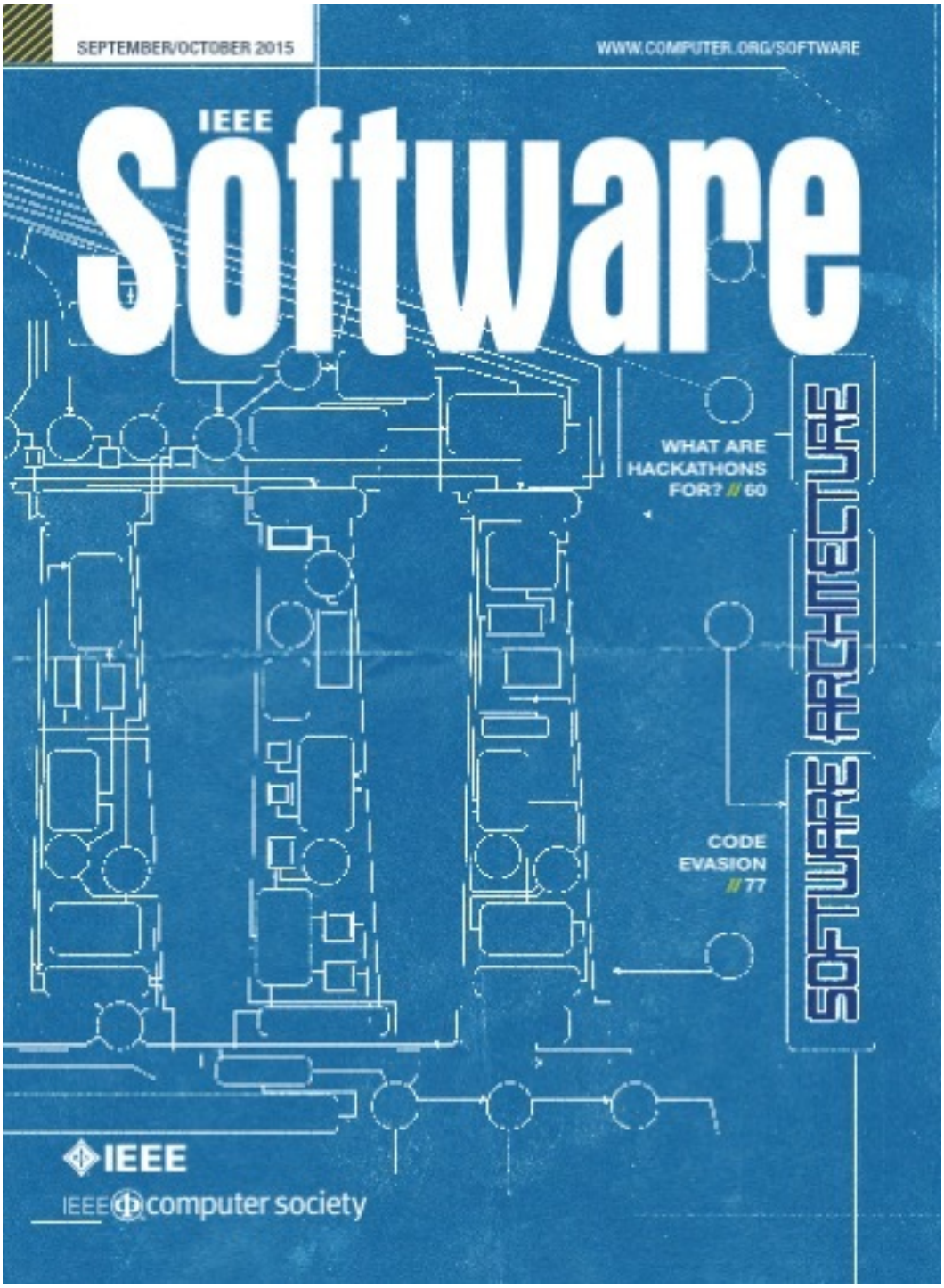
Philippe Kruchten, ***Lifelong Learning for Lifelong Employment****, IEEE Software, July 2015.*[926]

[926] DOI: 10.1109/MS.2015.97

"**Monitoring** is critical to IT system health and thus to businesses' bottom line."

Josune Hernantes, Gorka Gallardo, Nicolas Serrano, ***IT Infrastructure-Monitoring Tools****, IEEE Software, July 2015.*[927]

[927] DOI: 10.1109/MS.2015.96


SEPTEMBER/OCTOBER 2015
WWW.COMPUTER.ORG/SOFTWARE
IEEE
Software
WHAT ARE HACKATHONS FOR? // 60
SOFTWARE ARCHITECTURE
CODE EVASION // 77
IEEE
IEEE computer society

"**Agile teams** strive to balance **short-term feature** development with **longer-term quality** concerns. These evolutionary approaches often hit a '**complexity wall**" from the **cumulative effects** of **unplanned changes**, resulting in unreliable, poorly performing software."

Stephany Bellomo, Ian Gorton, Rick Kazman, ***Toward Agile Architecture: Insights from 15 Years of ATAM Data****, IEEE Software, September 2015.*[928]

[928] DOI: 10.1109/MS.2015.35

"Two innovations are enhancing programming languages' capabilities. First, **modularity** lets you combine independently developed languages without changing their respective definitions. A language is no longer a fixed quantity; you can extend it with **domain-specific constructs** as needed. Second, projectional editing lets you **build editors and IDEs** that don't require parsers. Such editors and IDEs support a range of tightly integrated notations, including textual, symbolic, tabular, and graphical notations."

Markus Voelter, Jos Warmer, Bernd Kolb, ***Projecting a Modular Future****, IEEE Software, September 2015.*[929]

[929] DOI: 10.1109/MS.2014.103

"A swift execution from idea to market has become a key competitive advantage for software companies to enable them to survive and grow in turbulent business environments. To combat this challenge, companies are using **hackathons**. A hackathon is a **highly engaging**, **continuous event** in which people in small groups produce working software **prototypes** in a limited amount of time. … However, hackathons pose the challenge of how to transform those promising **prototypes** into **finalized products** that create revenue and real business value."

Marko Komssi, Danielle Pichlis, Mikko Raatikainen, Klas Kindstrom, Janne Jarvinen, ***What are Hackathons for?****, IEEE Software, September 2015.*[930]

[930] DOI: 10.1109/MS.2014.78

"**Software adaptation** has become prominent owing to the proliferation of software in **everyday devices**. In particular, computing with the **Internet of Things** requires adaptability. Traditional software maintenance, which involves long, energy-consuming cycles, is no longer satisfactory. Adaptation is a lightweight software evolution that provides more transparent maintenance for users."

Franck Barbier, Eric Cariou, Olivier Le Goaer, Samson Pierre, ***Software Adaptation: Classification and a Case Study with State Chart XML****, IEEE Software, September 2015.*[931]

[931] DOI: 10.1109/MS.2014.130


NOVEMBER/DECEMBER 2015
IEEE
Software
REFACTORING
REFACTORING
IEEE
IEEE computer society
WWW.COMPUTER.ORG/SOFTWARE

"**Refactoring** changes a program's source code **without changing** its **external behavior**, typically to improve the software's design."

Emerson Murphy-Hill, Don Roberts, Peter Sommerlad, William F. Opdyke, ***Refactoring [Guest editors' introduction]****, IEEE Software, November 2015.*[932]

[932] DOI: 10.1109/MS.2015.136

"**The safety issue**—that a **refactoring** shouldn't **break working code**—was recognized as critical to industrial adoption. It also raised other interesting research issues."

---

*William G. Griswold, William F. Opdyke, **The Birth of Refactoring: A Retrospective on the Nature of High-Impact Software Engineering Research**, IEEE Software, November 2015.*[933]

[933] DOI: 10.1109/MS.2015.107

"Developers won't use tools that seem **unreliable**. So, the widespread use of **refactoring tools** speaks to their apparent reliability. However, they **aren't error-free**. They work just well enough to be useful, and they break in relatively **unimportant ways**."

---

*Munawar Hafiz, Jeffrey Overbey, **Refactoring Myths**, IEEE Software, November 2015.*[934]

[934] DOI: 10.1109/MS.2015.130

“**Refactoring** is a key approach for managing **technical debt**. In the past few years, refactoring techniques and tools have received considerable attention from researchers and tool vendors.”

Tushar Sharma, Girish Suryanarayana, Ganesh Samarthyam, ***Challenges to and Solutions for Refactoring Adoption: An Industrial Perspective****, IEEE Software, November 2015.*[935]

[935] DOI: 10.1109/MS.2015.105

"To **improve responsiveness**, developers often use **asynchronous programming**. In the post-PC era, **asynchronous programming** is even more in demand because mobile and wearable devices have limited resources and access the network excessively. One current development task is refactoring long-running, **blocking synchronous code** (for example, accessing the Web, a cloud, a database, or a file system) into nonblocking asynchronous code."

Danny Dig, ***Refactoring for Asynchronous Execution on Mobile Devices****, IEEE Software, November 2015.*[936]

[936] DOI: 10.1109/MS.2015.133

"Although **database refactoring** has been advocated as an important area of database development, little research has studied its implications. … The experience led to five key lessons learned: refactoring should be **automated** whenever possible, the database **catalog** is crucial, refactoring is easier when it's done **progressively**, refactoring can help **optimize** an application and streamline its code base, and refactoring related to application development requires a **complex skill set** and must be applied sensibly."

Gregory Vial, ***Database Refactoring: Lessons from the Trenches****, IEEE Software, November 2015.*[937]

[937] DOI: 10.1109/MS.2015.131

2016


JANUARY/FEBRUARY 2016
IEEE
Software
WWW.COMPUTER.ORG/SOFTWARE
the FUTURE of SOFTWARE ENGINEERING
TINY PROGRAMMING TOOLS // 24
REQUIREMENTS & SOCIAL RESPONSIBILITY // 109
IEEE computer society

"We live in a world of **unprecedented complexity** and **astonishing possibility**. We should **never forget our past**, for those who came before us in computing enabled those possibilities."

Grady Booch, ***Remembrance of Things Past***, *IEEE Software*, *January 2016.*[938]

[938] DOI: 10.1109/MS.2016.10

"The debacle with the **VW 'defeat device'** raises some unsettling questions. Are any other companies doing this, or-if we take a more cynical standpoint-how many are doing this? If they aren't, are they still using software practices almost as dubious? How do we decide **what's reasonable**, given software's extraordinary ability to give hardware its character?"

Les Hatton, Michiel van Genuchten, ***When Software Crosses a Line****, IEEE Software, January 2016.*[939]

[939] DOI: 10.1109/MS.2016.6

“This year marks the 50th anniversary of the Turing Award, which was first given to **Alan Perlis**, an oft-quoted mathematician who described **the relationship** between **humans and computers** as having ‘a vitality like a **gangly youth** growing out of his clothes within an **endless puberty**.’ Now that our dependence on software permeates nearly every aspect of our lives, it’s time to ask ourselves where this relationship is headed and, even though software engineering is still a relatively new discipline, **how much we’ve matured**.”

Forrest Shull, Anita Carleton, Jeromy Carriere, Rafael Prikladnicki, Dongmei Zhang, ***The Future of Software Engineering****, IEEE Software, January 2016.*[940]

[940] DOI: 10.1109/MS.2016.8

“Tim O’Reilly: Code for America is **changing government** and changing how we think about our role as software professionals. Its work is deeply rooted in the notion that you can **no longer govern without** using **digital technology**. Technology is central to how we deliver services today and how people access them. “

Andrew Moore, Tim O’Reilly, Paul D. Nielsen, Kevin Fall, ***Four Thought Leaders on Where the Industry Is Headed****, IEEE Software, January 2016.*[941]

[941] DOI: 10.1109/MS.2016.1

“To most people, “**massive systems**” probably means those systems run by NASA, airline companies, or large banks, or operating systems such as Microsoft Windows. What’s in common? They all have **complex components or subsystems**, deal with massive data, support millions of customers, require **real-time response**, and more. If they malfunction, catastrophe might ensue. By those standards, many systems run by **today’s Internet companies** also **qualify as massive**. As the Internet grows so quickly, many Internet companies are suffering the same problems that **massive systems** have suffered. “

Zhengrong Tang, Melissa Yang, Joshua Xiang, John Liu, ***The Future of Chinese Software Development****, IEEE Software, January 2016.*[942]

[942] DOI: 10.1109/MS.2016.19

"**Practitioners** must become **mediators** of the process of **creating a humane experience** and expand their practice to draw from disciplines such as **experience design**, systems thinking, **economics**, and digital strategy. They must do what they can to mitigate the **negative consequences** of technology while continuing to exploit and amplify its positive impacts."

Claudia de O. Melo, Ronaldo Ferraz, Rebecca J. Parsons, ***Brazil and the Emerging Future of Software Engineering****, IEEE Software, January 2016.*[943]

[943] DOI: 10.1109/MS.2016.28

"Software is being produced so fast that its growth hinders its sustainability. **Technical debt**, which encompasses **internal software quality**, evolution and maintenance, reengineering, and economics, is growing such that its management is becoming the dominant driver of software engineering progress. It spans the **software engineering life cycle**, and its management capitalizes on recent advances in fields such as source code analysis, quality measurement, and project management. **Managing technical debt** will become an investment activity applying **economic theories**."

Paris Avgeriou, Philippe Kruchten, Robert L. Nord, Ipek Ozkaya, Carolyn Seaman, ***Reducing Friction in Software Development****, IEEE Software, January 2016.*[944]

[944] DOI: 10.1109/MS.2016.13

"Almost surreptitiously, **crowdsourcing** has entered software engineering practice. **In-house development**, contracting, and outsourcing still dominate, but many development projects use crowdsourcing-for example, to squash bugs, test software, or gather alternative UI designs. Although the overall impact has been mundane so far, crowdsourcing could lead to **fundamental, disruptive changes** in how software is developed."

Thomas D. LaToza, Andre van der Hoek, ***Crowdsourcing in Software Engineering: Models, Motivations, and Challenges****, IEEE Software, January 2016.*[945]

[945] DOI: 10.1109/MS.2016.12

"An evaluation of recent industrial and societal trends revealed three key factors driving **software engineering's future**: **speed**, **data**, and **ecosystems**."

Jan Bosch, ***Speed, Data, and Ecosystems: The Future of Software Engineering****, IEEE Software, January 2016.*[946]

[946] DOI: 10.1109/MS.2016.14

"Today's **social-coding tools** foreshadow a transformation of the software industry, as it relies increasingly on **open libraries**, frameworks, and **code fragments**. Our vision calls for new **intelligently transparent services** that support **rapid development** of innovative products while helping developers manage risk and issuing them **early warnings** of looming failures. Intelligent transparency is enabled by an infrastructure that applies **analytics** to data from all phases of the life cycle of open source projects, from development to deployment. Such an infrastructure brings stakeholders the information they need when they need it."

James Herbsleb, Christian Kastner, Christopher Bogart, ***Intelligently Transparent Software Ecosystems****, IEEE Software, January 2016.*[947]

[947] DOI: 10.1109/MS.2015.156

"We're living in a **physical world** that's **moving at the speed of software**. This means that software's trajectory will drive software engineering, not vice versa. However, software engineering is also driven by visionary corporate leaders, backed by skilled software developers."

George Hurlburt, Jeffrey Voas, ***Software is Driving Software Engineering?****, IEEE Software, January 2016.*[948]

[948] DOI: 10.1109/MS.2016.22

“**Apache Mesos**, … abstracts away many of the hassles of managing a **distributed system**. … You don’t have to think about how you’re going to **get your task** to a **different machine** with Mesos. You just tell it, ‘**Run** this task using **these resources**; those resources are tied to a particular machine,’ and then **Mesos takes care** of getting the task there, **starting** the task, and **watching** it while it’s actually running on that machine. Mesos provides **primitives** that somebody building a distributed system can take advantage of. “

Jeff Meyerson, ***Ben Hindman on Apache Mesos****, IEEE Software, January 2016.*[949]

[949] DOI: 10.1109/MS.2016.18


MARCH/APRIL 2016
IEEE
Software
WWW.COMPUTER.ORG/SOFTWARE
CODE CLARITY 22
SOFTWARE ON A COMET 81
IEEE
computer society
CELEBRATING 70 YEARS

"**Different ages** of humanity have required **different modes of thinking**. These modes aren't only reflections of the particular circumstances of life in each age; they're also projections of the forces that propel us to the next."

---

*Grady Booch, **The Computational Human**, IEEE Software, March 2016.*[950]

[950] DOI: 10.1109/MS.2016.59

“A **software retrofit** can address problems of business-critical systems that are no longer maintainable.”

Thomas Ronzon, ***Software Retrofit in High-Availability Systems: When Uptime Matters***, *IEEE Software*, *March 2016.*[951]

[951] DOI: 10.1109/MS.2016.49

"**Naming conventions** affect the readability of your code and the ease with which you can find your way around when you're reviewing that code. Naming conventions **aren't meant to help the compiler**. A compiler has no trouble distinguishing names, no matter how long, short, or obscure they are. But to us humans, they can matter a great deal."

---

*Gerard J. Holzmann, **Code Clarity**, IEEE Software, March 2016.*[952]

[952] DOI: 10.1109/MS.2016.44

"Software engineering for **big data systems** is complex and faces challenges including **pervasive distribution**, write-heavy **workloads**, **variable request** loads, computation-intensive **analytics**, and high **availability**."

Ian Gorton, Ayse Basar Bener, Audris Mockus, ***Software Engineering for Big Data Systems****, IEEE Software, March 2016.*[953]

[953] DOI: 10.1109/MS.2016.47

“Conventional horizontal **evolutionary prototyping** for small-data system development is inadequate and too expensive for identifying, analyzing, and mitigating risks in big data system development. RASP (**Risk-Based, Architecture-Centric Strategic Prototyping**) is a model for cost-effective, systematic risk management in agile big data system development. It uses prototyping strategically and only in areas that architecture analysis can’t sufficiently address.”

---

*Hong-Mei Chen, Rick Kazman, Serge Haziyev, **Strategic Prototyping for Developing Big Data Systems**, IEEE Software, March 2016.*[954]

[954] DOI: 10.1109/MS.2016.36

"**Video data** has become the **largest source of big data.** Owing to video data's complexities, velocity, and volume, public security and other surveillance applications require efficient, intelligent runtime **video processing**."

Weishan Zhang, Liang Xu, Zhongwei Li, Qinghua Lu, Yan Liu, ***A Deep-Intelligence Framework for Online Video Processing****, IEEE Software, March 2016.*[955]

[955] DOI: 10.1109/MS.2016.31

"**Big data systems** (BDSs) are complex, consisting of multiple interacting hardware and software components, such as distributed computing nodes, databases, and middleware. Any of these components can fail. Finding **the failures' root causes** is extremely laborious. Analysis of **BDS-generated logs** can speed up this process. The logs can also help improve testing processes, detect security breaches, customize operational profiles, and aid with any other tasks requiring runtime-data analysis. However, **practical challenges** hamper log analysis tools' adoption."

Andriy Miranskyy, Abdelwahab Hamou-Lhadj, Enzo Cialini, Alf Larsson, ***Operational-Log Analysis for Big Data Systems: Challenges and Solutions****, IEEE Software, March 2016.*[956]

[956] DOI: 10.1109/MS.2016.33

“Many real-world data analysis scenarios require **pipelining** and integration of multiple (big) **data-processing** and **data-analytics jobs**, which often execute in heterogeneous environments, such as **MapReduce**; **Spark**; or **R, Python**, or **Bash scripts**. Such a pipeline requires much **glue code** to get data across environments.”

Dongyao Wu, Liming Zhu, Xiwei Xu, Sherif Sakr, Daniel Sun, Qinghua Lu, ***Building Pipelines for Heterogeneous Execution Environments for Big Data Processing****, IEEE Software, March 2016.*[957]

[957] DOI: 10.1109/MS.2016.35

"Clemens Szyperski (Microsoft), Martin Petitclerc (IBM), and Roger Barga (Amazon Web Services) answer three questions: What major challenges do you face when building **scalable, big data systems**? How do you **address these challenges**? Where should the research community focus its efforts to create **tools and approaches** for building highly reliable, scalable, big data systems?"

Clemens Szyperski, Martin Petitclerc, Roger Barga, ***Three Experts on Big Data Engineering****, IEEE Software, March 2016.*[958]

[958] DOI: 10.1109/MS.2016.58

“It’s also important to understand the difference between what a single programmer can do and what large teams of programmers can do. Even **the best practices of refactoring are really a joke** in the context of **a large legacy application**. Refactoring tools really don’t help you with large legacies.”

Dave Thomas, ***Innovating Legacy Systems***, *IEEE Software*, *March 2016.*[959]

[959] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2016.38


MAY/JUNE 2016
IEEE
Software
SOFTWARE ENGINEERING FOR DEVOPS
DEVELOPMENT
OPERATIONS
SOFTWARE
SOFTWARE ENERGY CONSUMPTION // 83
REACTIVE PROGRAMMING // 109
IEEE
IEEE computer society
CELEBRATING 70 YEARS
WWW.COMPUTER.ORG/SOFTWARE

"The next generation of software-intensive systems will be **taught instead of programmed**. This poses considerable pragmatic challenges in how we develop, deliver, and evolve them."

---

*Grady Booch, **It Is Cold. And Lonely.**, IEEE Software, May 2016.*[960]

[960] DOI: 10.1109/MS.2016.85

"Following certain best practices for **visual communication** can help bridge the gap between IT architects and business stakeholders. These practices stem from disciplines such as **psychology**, **graphic design**, **communication science**, and **cartooning**."

Jochem Schulenklopper, Eelco Rommes, ***Why They Just Don't Get It: Communicating about Architecture with Business Stakeholders****, IEEE Software, May 2016.*[961]

[961] DOI: 10.1109/MS.2016.67

"The emerging **DevOps movement** emphasizes **development and operations staff** working together **as early as possible**–sharing tools, processes, and practices that smooth the production path."

Eoin Woods, ***Operational: The Forgotten Architectural View****, IEEE Software, May 2016.*[962]

[962] DOI: 10.1109/MS.2016.86

"Building a **secure system** requires **proactive**, rigorous **analysis of the threats** to which it might be exposed, followed by systematic transformation of those threats into security-related requirements."

Jane Cleland-Huang, Tamara Denning, Tadayoshi Kohno, Forrest Shull, Samuel Weber, ***Keeping Ahead of Our Adversaries****, IEEE Software, May 2016.*[963]

[963] DOI: 10.1109/MS.2016.75

"**DevO**ps aims to reduce the time between **committing** a system change and placing the change **into normal production**, while ensuring **high quality**."

---

*Liming Zhu, Len Bass, George Champlin-Scharff, **DevOps and Its Practices**, IEEE Software, May 2016.*[964]

[964] DOI: 10.1109/MS.2016.81

“Modern software-based services are implemented as distributed systems with **complex behavior** and failure modes. Many large tech organizations are using **experimentation** to verify such systems’ reliability. **Netflix** engineers call this approach **chaos engineering**.”

Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, Casey Rosenthal, ***Chaos Engineering****, IEEE Software, May 2016.*[965]

[965] DOI: 10.1109/MS.2016.60

"When **DevOps** started gaining momentum in the software industry, one of the first service-based architectural styles to be introduced, be applied in practice, and become popular was **microservices**. Migrating **monolithic** architectures to **cloud-native** architectures such as microservices reaps many benefits, such as adaptability to technological changes and independent resource management for different system components."

Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, ***Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture****, IEEE Software, May 2016.*[966]

[966] DOI: 10.1109/MS.2016.64

"Wotif Group used **DevOps principles** to recover from the **downward spiral** of **manual release** activity that many IT departments face. Its approach involved the idea of '**making it easy to do the right thing**.' By defining the right thing (**deployment standards**) for development and operations teams and making it easy to adopt, Wotif drastically improved the average release cycle time."

Matt Callanan, Alexandra Spillane, ***DevOps: Making It Easy to Do the Right Thing****, IEEE Software, May 2016.*[967]

[967] DOI: 10.1109/MS.2016.66

"**Interconnected computing systems** in various forms will soon permeate our lives, realizing the **Internet of Things** (IoT) and letting us enjoy novel, enhanced services that promise to improve our everyday life. Nevertheless, this new reality introduces significant challenges in terms of **performance, scaling, usability**, and **interoperability**. Leveraging the benefits of service-oriented architectures (SOAs) can help alleviate many of the issues that developers, implementers, and users alike must face in the context of the IoT."

Konstantinos Fysarakis, Damianos Mylonakis, Charalampos Manifavas, Ioannis Papaefstathiou, ***Node.DPWS: Efficient Web Services for the Internet of Things****, IEEE Software, May 2016.*[968]

[968] DOI: 10.1109/MS.2015.155

“With the increasing importance, size, and complexity of **automated test suites**, the need exists for suitable methods and tools to develop, assess the quality of, and **maintain test code** (scripts) in parallel with **regular production** (application) code.”

Vahid Garousi, Michael Felderer, ***Developing, Verifying, and Maintaining High-Quality Automated Test Scripts****, IEEE Software, May 2016.*[969]

[969] DOI: 10.1109/MS.2016.30

"Building on **lean and agile** practices, **DevOps** means **end-to-end automation** in software development and delivery. Hardly anybody will be able to approach it with a cookbook-style approach, but most developers will benefit from better connecting the previously isolated silos of development and operations. Many **DevOps tools** exist that can help them do this."

---

*Christof Ebert, Gorka Gallardo, Josune Hernantes, Nicolas Serrano, **DevOps**, IEEE Software, May 2016.*[970]

[970] DOI: 10.1109/MS.2016.68


JULY/AUGUST 2016
IEEE
Software
SOFTWARE QUALITY
ARCHITECTURAL DESIGN PRINCIPLES // 15
WHY LARGE IT PROJECTS FAIL // 117
IEEE
IEEE computer society
CELEBRATING 70 YEARS
WWW.COMPUTER.ORG/SOFTWARE

“**Mobile apps** increasingly constitute **complete ecosystems** to support businesses such as farming. … Having the **right data** at the **right time** at the **right place** is crucial for high user productivity and a good user experience. In particular, **offline capability** is important but difficult. “

Susanne Braun, Ralf Carbon, Matthias Naab, ***Piloting a Mobile-App Ecosystem for Smart Farming****, IEEE Software, July 2016.*[971]

[971] DOI: 10.1109/MS.2016.98

“Many organizations use **business process models** to document business operations and formalize business requirements in software-engineering projects. The **Business Process Model and Notation** (BPMN), a specification by the Object Management Group, has evolved into the **leading standard** for **process modeling**. One challenge is **BPMN’s complexity**: it offers a huge variety of elements and often several representational choices for the same semantics.”

Henrik Leopold, Jan Mendling, Oliver Gunther, ***Learning from Quality Issues of BPMN Models from Industry****, IEEE Software, July 2016.*[972]

[972] DOI: 10.1109/MS.2015.81

"In the **mobile-app ecosystem**, **user ratings** of apps (a measure of **user perception**) are extremely important because they correlate strongly with downloads and hence revenue."

Hammad Khalid, Meiyappan Nagappan, Ahmed E. Hassan, ***Examining the Relationship between FindBugs Warnings and App Ratings****, IEEE Software, July 2016.*[973]

[973] DOI: 10.1109/MS.2015.29

"A proposed **data-driven** software **quality improvement** method has three elements. First, the downstream **Customer Quality Metric** (CQM) quantifies quality as customers perceive it. On the basis of data collected after systems are deployed, it measures how serious defects affect customers. Second, the upstream Implementation **Quality Index** (IQI) measures the effectiveness of error removal during development. IQI predicts future customer quality; it has a positive correlation with CQM. Finally, **prioritization tools** and techniques help focus limited development resources on the riskiest files in the code."

Randy Hackbarth, Audris Mockus, John Palframan, Ravi Sethi, ***Improving Software Quality as Customers Perceive It****, IEEE Software, July 2016.*[974]

[974] DOI: 10.1109/MS.2015.76

"**Measurement** of **software security** is an ongoing research field. **Privacy** is also becoming an imperative target as social networking and ubiquitous computing evolve and users exchange high volumes of personal information. However, security and privacy alone don't guarantee proper data protection; software must **also be dependable**."

George Hatzivasilis, Ioannis Papaefstathiou, Charalampos Manifavas, ***Software Security, Privacy, and Dependability: Metrics and Measurement****, IEEE Software, July 2016.*[975]

[975] DOI: 10.1109/MS.2016.61

“**Dynamic program analysis**, such as with **profiling**, **tracing**, and **bug-finding tools**, is essential for software engineering. Unfortunately, implementing dynamic analysis for managed languages such as Java is unduly difficult and error prone because the runtime environments provide only complex low-level mechanisms.”

Yudi Zheng, Stephen Kell, Lubomir Bulej, Haiyang Sun, Walter Binder, ***Comprehensive Multiplatform Dynamic Program Analysis for Java and Android****, IEEE Software, July 2016.*[976]

[976] DOI: 10.1109/MS.2015.151

"A well-known adage is '**diversity brings innovation.**' Diversity can be in **culture, thinking, discipline, gender**, and many more aspects. The result is the same: the chances for creating innovation in a given context increase when diversity is involved. To some extent, this principle should also hold for **gender diversity** in software teams. Achieving gender diversity in **IT-related fields** has been a goal for decades, but still, **too few women** choose such a career. But what skills or traits assigned to **the feminine role** bring concrete advantages to software teams?"

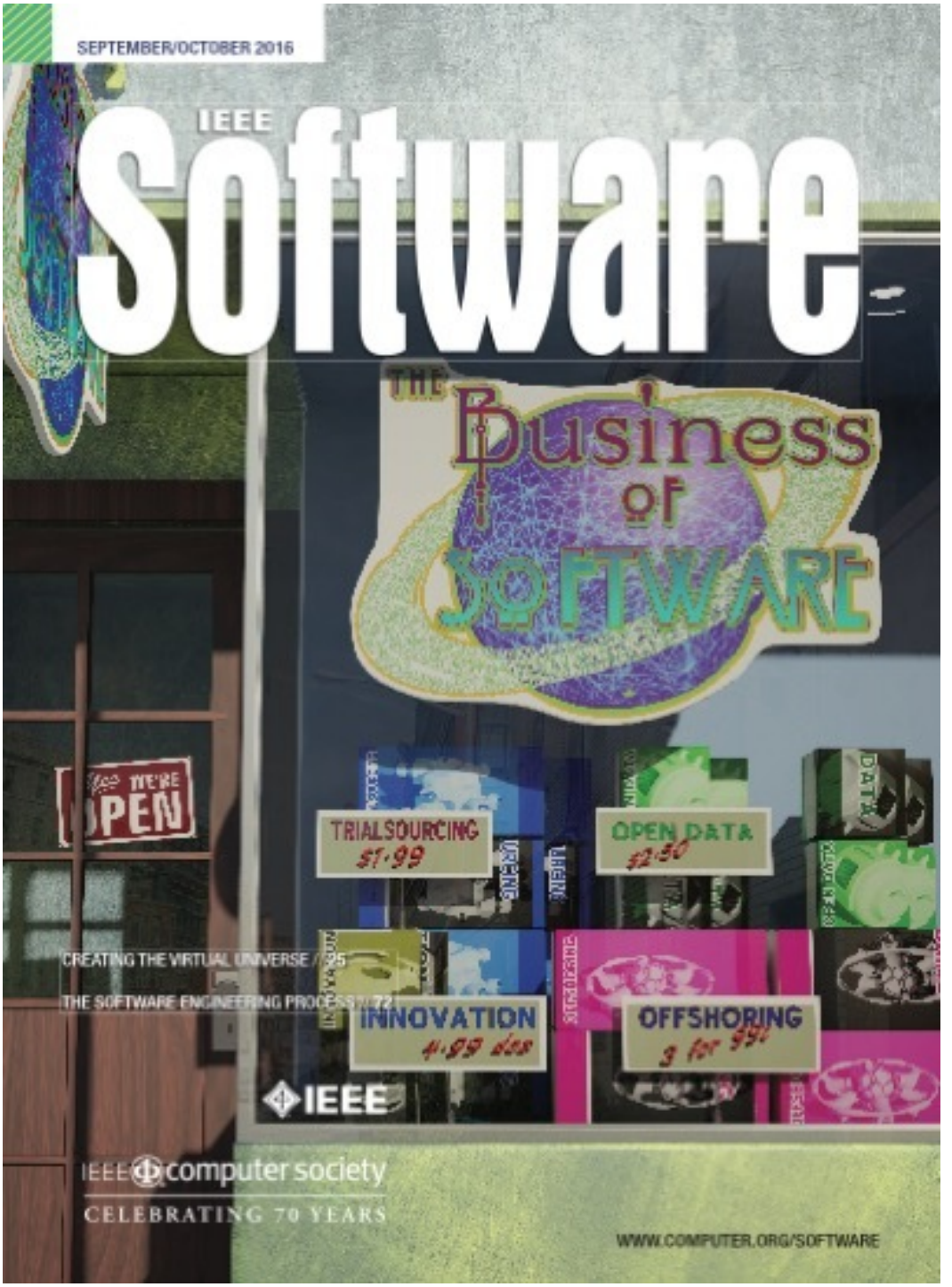
Maryam Razavian, Patricia Lago, ***Feminine Expertise in Architecting Teams****, IEEE Software, July 2016.*[977]

[977] DOI: 10.1109/MS.2015.84

"Users continue to stumble upon **software bugs**, despite developers' efforts to build and test high-quality software. Although **traditional testing** and **quality assurance** techniques are extremely valuable, software testing should pay more **attention to exploration**. Exploration can **directly apply knowledge** and learning to the core of industrial software testing, revealing more relevant bugs earlier."

Juha Itkonen, Mika V. Mantyla, Casper Lassenius, ***Test Better by Exploring: Harnessing Human Skills and Knowledge****, IEEE Software, July 2016.*[978]

[978] DOI: 10.1109/MS.2015.85


SEPTEMBER/OCTOBER 2016
IEEE
Software
THE Business of SOFTWARE
TRIALSOURCING
$1.99
OPEN DATA
$2.50
INNOVATION
4.99 ea
OFFSHORING
3 for $9!
IEEE
CREATING THE VIRTUAL UNIVERSE / 25
THE SOFTWARE ENGINEERING PROCESS / 72
IEEE computer society
CELEBRATING 70 YEARS
WWW.COMPUTER.ORG/SOFTWARE

"**Huge industries**, from the automotive and healthcare industries to finance and entertainment, **center increasingly on software**. **Managing** such a **software business** is tough because software's ethereal nature offers infinite lucrative or catastrophic choices. The main things to manage are the business model, the **execution strategy**, the product or service, and the development process."

Diomidis Spinellis, ***Managing a Software Business****, IEEE Software, September 2016.*[979]

[979] DOI: 10.1109/MS.2016.111

"**Computational humor** is a technically intriguing problem. And, in the journey to understand the theories, mechanisms, and algorithms that discern and **define funny**, we learn something about ourselves and what it means to be human."

---

*Grady Booch, **No Laughing Matter**, IEEE Software, September 2016.*[980]

[980] DOI: 10.1109/MS.2016.127

"Developers of **systems of systems** face challenges such as **heterogeneous**, **inconsistent**, and changing elements; continuous evolution and deployment; decentralized control; and inherently **conflicting** and often **unknowable** requirements."

Michael Vierhauser, Rick Rabiser, Paul Granbacher, ***Monitoring Requirements in Systems of Systems****, IEEE Software, September 2016.*[981]

[981] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2016.112

"Developers of **systems of systems (SoSs)** face challenges such as heterogeneous, inconsistent, and changing elements; continuous evolution and deployment; decentralized control; and inherently conflicting and often unknowable requirements."

Michael Vierhauser, Rick Rabiser, Paul Grunbacher, ***Monitoring Requirements in Systems of Systems****, IEEE Software, September 2016.*[982]

[982] DOI: 10.1109/MS.2016.112

"**Simulation software** is important to our understanding of the universe. The intrinsic **multiphysics** aspects are spiced with a range of temporal scales and spatial scales, both of which cover **more digits** than are available in the **standard hardware**."

Simon Portegies Zwart, Jeroen Bedorf, ***Creating the Virtual Universe****, IEEE Software, September 2016.*[983]

[983] DOI: 10.1109/MS.2016.113

“The **software architecture** pendulum is swinging away from traditional practices and toward **agile and continuous** practices. To be successful in this new world, architects should emphasize **products over projects**, drive architectural decisions, **understand code**, and communicate and collaborate effectively with delivery teams.”

Murat Erder, Pierre Pureur, ***What’s the Architect’s Role in an Agile, Cloud-Centric World?****, IEEE Software, September 2016.*[984]

[984] DOI: 10.1109/MS.2016.119

"**Small** and **medium-sized enterprises** depend heavily on their capability to **differentiate themselves** from their competitors through innovative approaches. **Innovation management** assumes that systematically applying strategies combined with appropriate methods and tools increases the ability to build innovative products and services. To leverage their competitive capabilities, small companies involved in software development must combine **innovation management** and **software engineering practices**."

Ricardo Eito-Brun, Miguel-Angel Sicilia, ***Innovation-Driven Software Development: Leveraging Small Companies' Product-Development Capabilities****, IEEE Software, September 2016.*[985]

[985] DOI: 10.1109/MS.2016.63

"**Software providers** differ widely in **productivity and quality**. Traditional means of evaluation, such as CVs and client references, fail to separate the competent from the incompetent. **Trialsourcing**–having **multiple providers** create **sample pieces of software** for evaluation–can help clients select providers."

Magne Jorgensen, ***Better Selection of Software Providers through Trialsourcing****, IEEE Software, September 2016.*[986]

[986] DOI: 10.1109/MS.2015.24

"Most companies have learned that **cost calculations** for **offshore outsourcing** shouldn't be **limited to hourly wages**. Looking at salaries alone, you could naively hope for cost reductions of up to 90 percent. However, don't underestimate the cost of **knowledge transfer**, **travel**, **attrition**, **miscommunication**, and so on. … The offshore team's true hourly costs took three years to become comparable with those of the in-house team. Getting close to the break-even point took five years. **Learning costs** due to **offshore employee turnover** were the **primary cost** factor to get under control."

Darja Smite, Rini van Solingen, ***What's the True Hourly Cost of Offshoring?****, IEEE Software, September 2016.*[987]

[987] DOI: 10.1109/MS.2015.82

"You could view **maintenance** as an impending **operational cost tsunami**, owing to seismic development activities. It's no longer tenable to keep creating new individual solutions to the same basic problems because those solutions must be maintained as long as they live, binding expensive human resources that are constantly declining."

---

*Harry M. Sneed, Chris Verhoef, **From Software Development to Software Assembly**, IEEE Software, September 2016.*[988]

[988] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2015.78

"In **machine learning**, a computer first learns to perform a task by studying **a training set** of examples. The computer then performs the same task with data it hasn't encountered before."

---

*Panos Louridas, Christof Ebert, **Machine Learning**, IEEE Software, September 2016.*[989]

[989] DOI: 10.1109/MS.2016.114

"**Too many tests** is the same as **not enough tests**. In both cases it's **suboptimal**. Whether you waste time **debugging** because you don't have enough tests or you waste time **maintaining tests** that don't need to be there, at the end of the day both of those things amount to waste."

---

*Stefan Tilkov, **Jay Fields on Working with Unit Tests**, IEEE Software, September 2016.*[990]

[990] DOI: 10.1109/MS.2016.121


NOVEMBER/DECEMBER 2016
IEEE
Software
The Role of the
Software
Architect
SOFTWARE AND THE ENVIRONMENT // 23
THE TRAGEDY OF DEFECT PREDICTION // 102
IEEE
IEEE computer society
CELEBRATING 70 YEARS

"Being **a good software architect** has never been easy. Changes in the software industry are making the job even more challenging. The key drivers are the rising role of software in systems and their operation; more emphasis on **reuse**, **agility**, and **testability** during software development; and several quality elements increasingly affected by architectural choices."

Diomidis Spinellis, ***The Changing Role of the Software Architect****, IEEE Software, November 2016.*[991]

[991] DOI: 10.1109/MS.2016.133

“Documenting **the time dimension** part of your architecture might look like extra work. However, **anticipation** should be a large part of your job as an architect, anyway. If you **communicate your anticipation** as an evolution viewpoint or architecture roadmap, your architecture description will stay valid longer. And, you’ll have a ready answer when stakeholders ask how you’ve addressed their change and planning concerns.”

Eltjo Poort, ***Just Enough Anticipation: Architect Your Time Dimension****, IEEE Software, November 2016.*[992]

[992] DOI: 10.1109/MS.2016.134

“As new and exciting healthcare applications arise that use **smart technologies**, the **Internet of Things**, **data analytics**, and other technologies, a critical problem is emerging: the potential loss of caring. Although these exciting technologies have improved patient care by allowing for better assessment, surveillance, and treatment, their use can disassociate the caregiver from the patient, essentially removing the ”care” from healthcare. So, you can view caring as an undiscovered -ility that ranks at least as important as other well-known -ilities in healthcare systems.”

Nancy Laplante, Phillip A. Laplante, Jeffrey Voas, ***Caring: An Undiscovered ”Super -ility” of Smart Healthcare****, IEEE Software, November 2016.*[993]

[993] DOI: 10.1109/MS.2016.136

“When customers visit a **Brazilian e-commerce** site and search for a product, they’re likely using software developed by **Neemu**, a start-up created in **Manaus**, a city in the heart of the **Amazon rainforest**. Nowadays, millions of people throughout Brazil use this software, which demonstrates alternative economic **development in Amazonia** that has low impact on the environment.”

Edleno Silva de Moura, Mauro Rojas Herrera, Leonardo Santos, Tayana Conte, ***When Software Impacts the Economy and Environment****, IEEE Software, November 2016.*[994]

[994] DOI: 10.1109/MS.2016.135

“Forty years ago, **Thomas McCabe** introduced his famous **cyclomatic complexity** (CC) metric. Today, it’s still one of the **most popular and meaningful** measurements for analyzing code. “

---

*Christof Ebert, James Cain, **Cyclomatic Complexity**, IEEE Software, November 2016.*[995]

[995] DOI: 10.1109/MS.2016.147

"**Internet scale**, the increasing rate of technology evolution, and the broad adoption of lean and agile methods have triggered a profound change in not only application and infrastructure architectures but also the **software architect**'s roles and responsibilities."

Gregor Hohpe, Ipek Ozkaya, Uwe Zdun, Olaf Zimmermann, ***The Software Architect's Role in the Digital Age****, IEEE Software, November 2016.*[996]

[996] DOI: 10.1109/MS.2016.137

"The popularity of agile methods such as **Scrum** and **Kanban**, with their clear focus on **team collaboration**, threatens many roles traditionally assigned to individual experts. Some organizations are even challenging the raison d'être of the software architect role. However, researchers' experiences developing connected-vehicle software revealed **two reasons** why successful projects still often assign **architecture-related responsibilities** to individual experts acting as software architects. First, **the experts** help effectively **manage complexity**; second, they act as **knowledge multipliers** when development must scale up."

Soren Frey, Lambros Charissis, Jens Nahm, ***How Software Architects Drive Connected Vehicles****, IEEE Software, November 2016.*[997]

[997] DOI: 10.1109/MS.2016.145

"**Software architects** are key assets for successful development projects. … researchers investigated how architects at **Ericsson** were organized, their roles and responsibilities, and **the effort** they spent guarding and governing a large-scale legacy product developed by teams at multiple locations. … the architectural decisions were centralized to a **team of architects**. The team extensively **used code reviews** to not only check the code's state but also reveal defects that could turn into maintainability problems. … **the effort** architects spend designing architecture, guarding its integrity and evolvability, and mentoring development teams is directly related to **team maturity**."

Ricardo Britto, Darja Smite, Lars-Ola Damm, ***Software Architects in Large-Scale Distributed Projects: An Ericsson Case Study****, IEEE Software, November 2016.*[998]

[998] DOI: 10.1109/MS.2016.146

“Owing to the increasing amount of **computation** in **electromechanical devices**, the role of **software architect** is often found in **embedded-systems** development. However, because computer scientists usually have limited knowledge of **embedded-systems concepts** such as **controllers**, **actuators**, and **bu**ses, embedded-software architects are often engineers with no education in software architecture basics, which is normally a topic in computer science courses.”

Pablo Oliveira Antonino, Andreas Morgenstern, Thomas Kuhn, ***Embedded-Software Architects: It’s Not Only about the Software****, IEEE Software, November 2016.*[999]

[999] DOI: 10.1109/MS.2016.142

"**Software architects** don't just design architecture components or champion architecture qualities; they often must guide and **harmonize** the entire **community of project stakeholders**. The **community-shepherding** aspects of the architect's role have been gaining attention, given the increasing importance of complex 'organizational rewiring' scenarios such as DevOps, open source strategies, transitions to agile development, and corporate acquisitions"

Damian A. Tamburri, Rick Kazman, Hamed Fahimi, ***The Architect's Role in Community Shepherding****, IEEE Software, November 2016.*[1000]

[1000] DOI: 10.1109/MS.2016.144

"As software systems have evolved, so has software architecture, with practices growing to meet each era's new challenges. The next phase of evolution–**intelligent connected systems**–promises to be an exciting time for **software architects**."

Eoin Woods, ***Software Architecture in a Changing World****, IEEE Software, November 2016.*[1001]

[1001] DOI: 10.1109/MS.2016.149

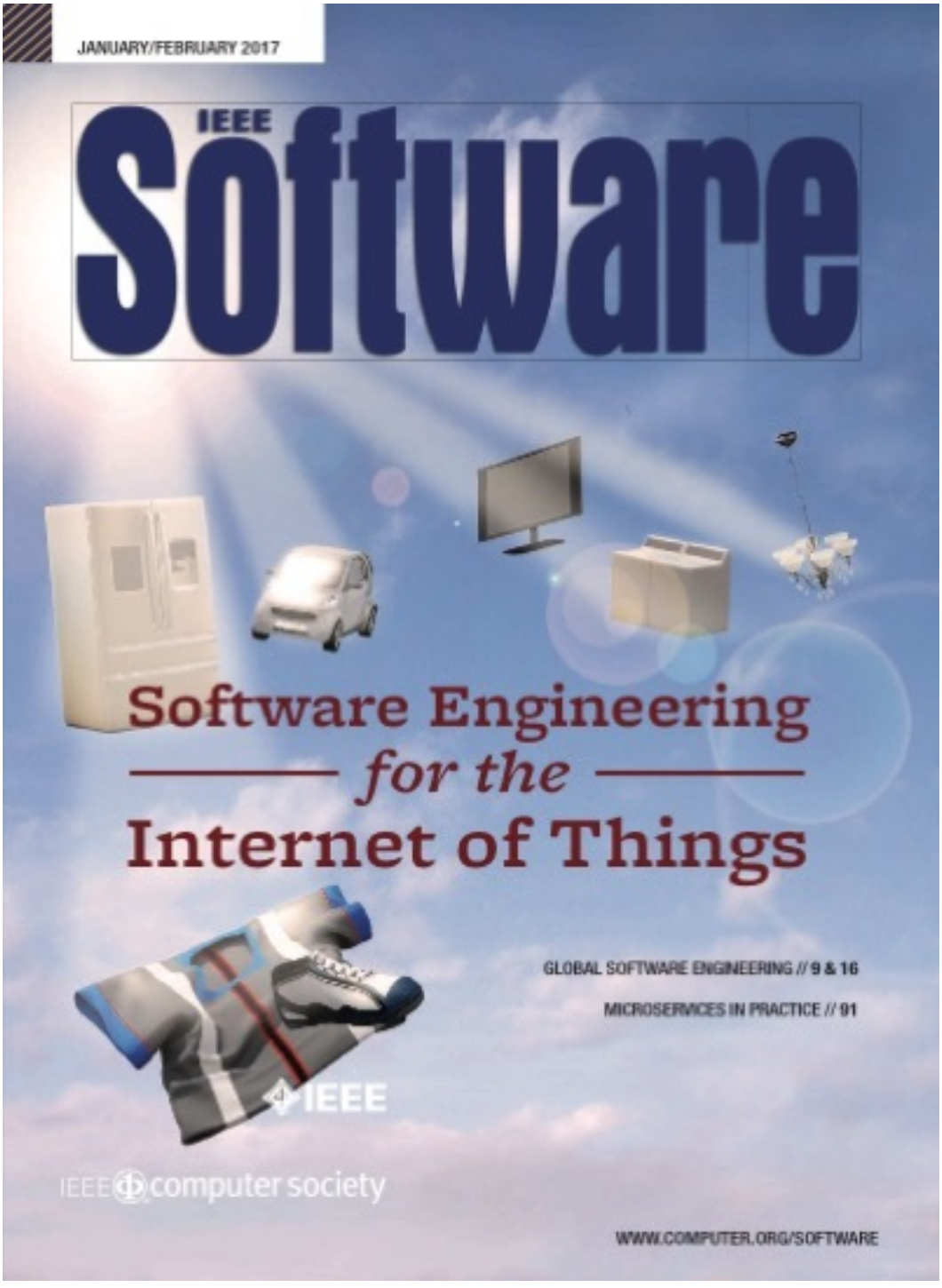
"**High-maintenance code** not only is verbose but also tends to rely on unstated, poorly stated, or incompletely stated assumptions. If you want to understand that type of code, you need long chains of reasoning to figure out how and why it works, and under which conditions it could start failing when other parts of the system are updated. **The reliance on hidden assumptions** is probably the most telling feature of high-maintenance code."

---

*Gerard J. Holzmann, **Hi Maintenance**, IEEE Software, November 2016.*[1002]

[1002] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2016.153

2017


JANUARY/FEBRUARY 2017
IEEE
Software
Software Engineering
for the
Internet of Things
GLOBAL SOFTWARE ENGINEERING // 9 & 16
MICROSERVICES IN PRACTICE // 91
IEEE
IEEE computer society
WWW.COMPUTER.ORG/SOFTWARE

“New wiring transformed **ENIAC** into a versatile stored-program computer. **Rewiring Internet of Things infrastructures** into a **general-purpose computing fabric** can similarly change how modern computation interfaces with our environment.”

Diomidis Spinellis, ***Software-Engineering the Internet of Things***, *IEEE Software, January 2017.*[1003]

[1003] DOI: 10.1109/MS.2017.15

"The proper **alignment of requirements engineering and testing** (RET) can be key to software's success. Three practices can provide effective RET alignment: using **test cases** as requirements, harvesting **trace links**, and **reducing distances** between requirements engineers and testers."

Elizabeth Bjarnason, Markus Borg, ***Aligning Requirements and Testing: Working Together toward the Same Goal****, IEEE Software, January 2017.*[1004]

[1004] DOI: 10.1109/MS.2017.14

"No consolidated set of software engineering **best practices** for the **Internet of Things (IoT)** has yet emerged. Too often, the landscape resembles **the Wild West**, with unprepared programmers putting together IoT systems in ad hoc fashion and throwing them out into the market, often poorly tested. In addition, the academic sector is in danger of fragmenting into specialized, often unrelated research areas."

Xabier Larrucea, Annie Combelles, John Favaro, Kunal Taneja, ***Software Engineering for the Internet of Things****, IEEE Software, January 2017.*[1005]

[1005] DOI: 10.1109/MS.2017.28

"The **Internet of Things (IoT)** is a challenging combination of **distribution and heterogeneity**. A number of software engineering solutions address those challenges in isolation, but few solutions tackle them in combination, which poses a set of concrete challenges. The **ThingML** (Internet of Things Modeling Language) approach attempts to address those challenges."

Brice Morin, Nicolas Harrand, Franck Fleurey, ***Model-Based Software Engineering to Tame the IoT Jungle****, IEEE Software, January 2017.*[1006]

[1006] DOI: 10.1109/MS.2017.11

"Despite the progress in **Internet of Things (IoT)** research, a general software engineering approach for systematic development of IoT systems and applications is still missing. A synthesis of the state of the art in the area can help frame the **key abstractions** related to such development."

Franco Zambonelli, ***Key Abstractions for IoT-Oriented Software Engineering***, *IEEE Software, January 2017.*[1007]

[1007] DOI: 10.1109/MS.2017.3

“**Mission-critical Internet of Things (MC-IoT)** systems involve heterogeneous things from both the **digital and physical worlds**. They run applications whose failure might cause significant and possibly **dramatic consequences**, such as interruption of public services, significant business losses, and deterioration of enterprise operations. These applications require not only high **availability**, **reliability**, **safety**, and **security** but also **regulatory compliance**, **scalability**, and **serviceability**. At the same time, they’re exposed to various facets of uncertainty, spanning from software and hardware variability to mission planning and execution in possibly unforeseeable environments. **Model-driven engineering** can potentially meet these challenges and better enable the adoption of MC-IoT systems.”

Federico Ciccozzi, Ivica Crnkovic, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, Romina Spalazzese,

Model-Driven Engineering for Mission-Critical IoT Systems,

IEEE Software, January 2017.[1008]

[1008] DOI: 10.1109/MS.2017.1

"A roadmap from today's cloud-centric, data-centric IoT systems to the **Programmable World** highlights the technical challenges that deserve to be part of developer education and deserve deeper investigation beyond those IoT topics that receive the most attention today."

Antero Taivalsaari, Tommi Mikkonen, ***A Roadmap to the Programmable World: Software Challenges in the IoT Era****, IEEE Software, January 2017.*[1009]

[1009] DOI: 10.1109/MS.2017.26

"**Microservices** are in many ways **a best-practice approach** for realizing **SOA**."

Cesare Pautasso, Olaf Zimmermann, Mike Amundsen, James Lewis, Nicolai Josuttis, ***Microservices in Practice, Part 1: Reality Check and Service Design****, IEEE Software, January 2017.*[1010]

[1010] DOI: 10.1109/MS.2017.24

"Just as physicists infer dark matter's presence on the basis of its gravitational effects on visible matter, we can conceptualize a '**darkitecture**' that outlines visible software architectures."

Balaji Prasad, ***Darkitecture: The Reality Skirted by Architecture****, IEEE Software, January 2017.*[1011]

[1011] DOI: 10.1109/MS.2017.7

"**Computer games** are rich, complex, and often large-scale software applications. They're a significant, interesting, and often compelling domain for innovative research in software engineering techniques and technologies. **Computer games** are progressively changing the everyday world in many positive ways. Game developers, whether focusing on entertainment market opportunities or game-based applications in nonentertainment domains such as education, healthcare, defense, or scientific research (that is, serious games), thus share a common interest in how best to engineer game software."

Walt Scacchi, ***Practices and Technologies in Computer Game Software Engineering****, IEEE Software, January 2017.*[1012]

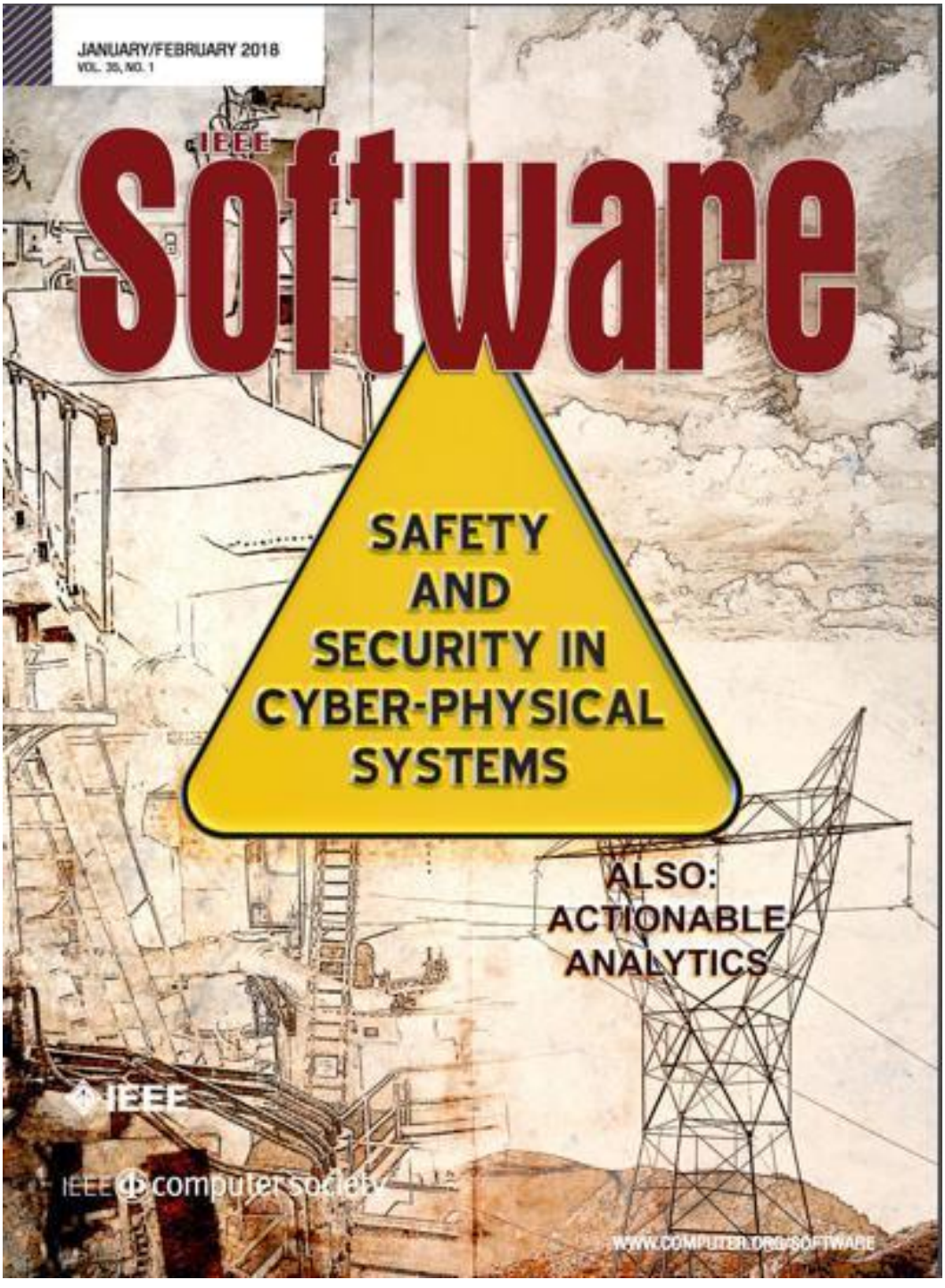
[1012] DOI: 10.1109/MS.2017.20

“‘What is **Infrastructure as code**? There are a lot of ways to answer that. One is that automation is the “CALM” of DevOps. CALM stands for **culture, automation, learning, and measurement**. Infrastructure as Code is about the automation piece. That’s how people who have been doing DevOps for a while approach it, using tools like Chef, Puppet, Ansible, and SaltStack. The philosophy behind this is that infrastructure has become like data: the physical layer has been abstracted. It’s **become software**, as opposed to being a physical thing. We can use infrastructure tools the same way we use software. We can **bring in best practices from software development**, such as continuous integration CI, test-driven development, and continuous delivery CD version control systems, and apply them to managing our infrastructure. “

Sven Johann, ***Kief Morris on Infrastructure as Code***, *IEEE Software, January 2017.*[1013]

[1013] DOI: 10.1109/MS.2017.13

2018


JANUARY/FEBRUARY 2018
VOL. 35, NO. 1
IEEE
Software
SAFETY AND SECURITY IN CYBER-PHYSICAL SYSTEMS
ALSO: ACTIONABLE ANALYTICS
IEEE computer society
WWW.COMPUTER.ORG/SOFTWARE

"Although intensive research on **software analytics** has been going on for nearly a decade, a repeated complaint in software analytics is that industrial practitioners find it **hard to apply the results** generated from data science."

Ye Yang, Davide Falessi, Tim Menzies and Jairus Hihn, ***Actionable Analytics for Software Engineering****, IEEE Software, January 2018.*[1014]

[1014] DOI: https://doi.ieeecomputersociety.org/10.1109/MS.2017.4541039