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. Such damage isn’t limited just to software. More than one IT-dependent business has folded after years of changes eroded its critical software infrastructure to the point where it could no longer support the company’s full weight.

Consider a few examples. Twenty years ago, most software engineers would have burst out laughing if someone had asked them to accept daily binary patches to critical pieces of their basic software infrastructure. Nowadays, those same engineers worry when they don’t see such patches for a week or two—perhaps an ominous sign that application providers have thrown in the towel in dealing with the exponential growth of spyware, viruses, worms, and Trojans. Similarly, software and hardware installations have undergone an astonishing transformation. Installations and upgrades that once took hours, days, or even weeks have been automated by installation processes that let even casual users add, change, or remove software, and even hardware, in minutes.

At a finer level of detail, one that’s often below users’ perception, changes are occurring even more rapidly. Mobile code’s rapid growth has blurred the once-solid boundary between data and software, letting distressingly powerful programs wink in and out of existence on PCs like fireflies on a summer night. If we’re prudent and a bit lucky, these ephemeral fireflies of functional change will remain as transient and harmless as their real-life counterparts. But even the most harmless of these mobile code fireflies represent a level of automated change to system configurations that two decades ago (and still today) would’ve had configuration managers waking up in a cold sweat.

Eroding “-ilities”

We’d like to suggest, however, that accelerating change’s most dangerous impact isn’t configuration control per se, but something more subtle and ultimately far more dangerous: The rapid erosion of hard-won software properties—the “-ilities” of traditional software engineering—just when they’re needed most. To put it another way: While we’re moving from building isolated homes to creating globe-spanning edifices, the rains of rapid
Once we learn how to build and sustain industrial-strength software properties that can weather even unrelenting downpours of change, how should we use them? Should we continue designing software much as we’ve done in the past, or will this new capability lead to deeper changes?

I believe the latter to be true. Although we can benefit from persistent software attributes in any form, taking full advantage of them will require a fundamental change in software engineering. I can sum up that change in two words: space awareness. Since its earliest days, software engineering has been unique among engineering disciplines in its ability to ignore real space. Many early programs and processes sprang up, lived, and died entirely within the confines of isolated computers, never having to face real-space location and geometry issues that are critical to success in disciplines such as civil, mechanical, and power engineering.

That world is gone. Nowadays, a task as simple as asking Google how to spell a word can result in electronic queries that travel across continents, and the success of many space programs depends on our ability to build and use interplanetary networks. In just a few decades, software has expanded from its cozy niches within isolated computers to vast distributed systems that encompass global and even interplanetary volumes of real space.

So, how is software’s expansion into real space related to creating enduring software properties? Simple: When you build in real space, you use different properties for different system parts. Although a house made using only steel would certainly be strong, it would also be cold from a lack of insulation, dark from a lack of windows, dangerous to use with electricity, and far more costly than a house made using materials with varying properties. As we become better at building software that exhibits specific industrial-strength properties such as persistent security, availability, adaptability, and scalability, I believe that better options for composing such properties in real space will emerge. With that realization, software engineering will expand to take on a great new challenge: understanding how best to configure and compose software components in real space.

In this New Software Engineering, we’ll select off-the-shelf components on the basis of the specific persistent properties they can contribute to larger systems, not just on functionality. Global skeletons with persistent availability will support local nodes that focus on persistent adaptability and scalability, while complex, location-aware security systems guard the entire system from malicious attacks. The New Software Engineering will be more complex, but it’ll also lower costs and increase capabilities the way that good house construction does: by placing the right properties at the right real-space locations. Ironically, bringing awareness of real space issues into software design will also bring software engineering closer to its roots in older forms of engineering, ones that haven’t had the luxury of neglecting a concept as basic as where components are located in space.

IEEE Software, I expect, will be sharing more of this vision of New Software Engineering in upcoming issues.

—Terry Bollinger
**Persistent software attributes**

The real message, then, is this: We need new software engineering building materials, ones whose properties won’t crumble like mud under the rains of constant change. We need updated methods for achieving maintainability, scalability, security, privacy, adaptability, and a long list of other -ilities—methods that embrace change instead of flinching from it. We’ll end up with the same -ilities, but this time wrought in steel and stone instead of sand and clay. The authors of the articles in this focus section describe how they’ve embraced this difficult challenge of building such persistent software attributes. Their lessons show that constructing PSAs will necessarily be more automated, complex, dynamic, and geographically distributed than constructing simple -ilities.

The biggest challenge, however, lies ahead. In a world in which it seems that everyone is writing software, it’s time for software engineering to take up the challenge of architectural leadership. We need to not just understand and demonstrate how to design software that stands the tests of time but also show how such -ility-first designs will increase range, scale, and even beauty of future globe-spanning software creations.

We can’t afford to live in a world of clay.

---

**National ICT Australia Limited**

Australia’s New Centre of ICT Excellence

National ICT Australia (NICTA) is Australia’s national research centre in Information and Communications Technology (ICT), with a focus on fundamental and use-inspired research.

**Researcher (Level B/C)/Senior Principal Researcher (Level D/E)**

Two positions - FORMAL METHODS

Combining NICTA researchers and researchers seconded from the University of New South Wales, the Formal Methods Program has expertise in program refinement, model checking, formal modelling of programming languages, and specialised methods for embedded and real-time systems, concurrency and distributed systems. The program is part of the Sydney Research Laboratory of NICTA and is located at the University of New South Wales (UNSW).

The program currently has two openings, in the range Researcher-Senior Principal Researcher (Assistant Professor/Lecturer-Full Professor equivalent). An appointment at Principal/Senior Principal Researcher level will be expected to contribute to leadership of the program and may provide the opportunity for periods in the role of Program Leader.

Applications from outstanding applicants in all areas of formal methods are welcome, however specific consideration will be given to the applicant’s capacity to contribute to NICTA’s priority research challenges, information concerning which can be found on NICTA’s website (http://www.nicta.com.au).

NICTA’s intellectual property policies provide for significant researcher benefit from technology transfer and commercialisation. There will also be opportunities for PhD student supervision and teaching of postgraduate and advanced level undergraduate courses through a conjoint appointment at the School of Computer Science and Engineering, UNSW. This school is one of Australia’s top schools in the ICT area, and has a large body of both local and international students of outstanding quality.

We expect the position to be highly competitive and excellence of research record and potential to contribute to the mission of NICTA will be key factors in selecting applicants.

Enquiries concerning this position should be addressed to A/Prof. Ron van der Meyden at meyden@nicta.com.au or +612 9385 4897.

**Applications:** Please visit NICTA Careers to view the criteria essential to this role and apply online (http://nicta.com.au/director/careers.cmf).

**Closing date:** 22 November 2004.

---

**About the Guest Editors**

**Terry Bollinger** is an IT analyst at Mitre. His research interests focus on applying technology to large, complex software systems and networks. He received his MS in computer science from the University of Missouri at Rolla. He is the primary author of the Software Construction section of Version 1.0 of the international IEEE Software Engineering Body of Knowledge (SwEBOK) standard. He was an IEEE Software editorial board member and associate editor in chief (software construction). He received the IEEE Millennium Medal for his many years of service to the software community. Contact him at 7515 Colshire Dr., McLean, VA 22102-7508; terry@mitre.org.

**Jeffrey Voas** is the chief scientist of Cigital. His research interests include software reliability, safety, fault tolerance, metrics and measurement, standards, software certification, and product liability. He received his PhD in computer science from the College of William & Mary. He is an advisory board member and past associate editor in chief (quality) of IEEE Software as well as an associate editor in chief of IT Professional. He is the 2003–2004 president of the IEEE Reliability Society and was named the 2000 IEEE Reliability Engineer of the Year. He is a Senior Member of the IEEE. Contact him at Cigital, 21351 Ridgetop Cir., Ste. 400, Dulles, VA 20166-6503; jvvoas@cigital.com.

**Maarten Boasson** is a member of the Department of Computer Science at the University of Amsterdam. His research focuses on preventing unnecessary complexity in system design. He received his MS in mathematics from Groningen University. He is a member of the IEEE Computer Society, ACM, and American Association of Artificial Intelligence, and cofounder of the IEEE task force on computer-based systems engineering. Contact him at the Dept. of Computer Science, Univ. of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, Netherlands; boasson@science.uva.nl.