ARTICLE SUMMARIES

Good Ideas, Through the Looking Glass
pp. 28-39
Niklaus Wirth

Computing's history has been driven by many good and original ideas, but a few turned out to be less brilliant than they first appeared.

Much can be learned from analyzing not only bad ideas and past mistakes, but also good ones. After all, thorough self-critique is the hallmark of any subject claiming to be a science.

Ten Commandments of Formal Methods … Ten Years Later
pp. 40-48
Jonathan P. Bowen and Michael G. Hinchey

More than a decade ago, the authors offered practical guidelines for projects that sought to use formal methods. Over the years, that article has been widely cited and generated much positive feedback. However, despite this apparent enthusiasm, formal methods use has not greatly increased and some of the same attitudes about the infeasibility of adopting them persist.

Yet 10 years later, the original formal methods commandments remain valid. The use of formal methods is not as prevalent as hoped, but formal approaches will always have a niche in computer-based systems development, especially when correct functioning is critical. The next 10 years should see some significant progress in integrating formal methods and traditional development practices.

The Pentium Chronicles: Introduction
pp. 49-54
Robert P. Colwell

An excerpt from *The Pentium Chronicles: The People, Passion, and Politics Behind the Landmark Chips* (Wiley-IEEE Computer Society Press, 2006) offers a project manager's first-hand account of the technical and management challenges facing the team that conceived Intel's P6 microarchitecture—the most successful general-purpose processor ever created.

Successfully Outsourcing Embedded Software Development
pp. 55-61
Joseph W. Rottman

A large US company, not discouraged by its first failed attempt at offshore outsourcing, capitalized on that experience to build a global development model designed to improve quality, shorten cycle time, and decrease development costs. The company's first failed attempt exceeded project budgets, decreased software quality, and left projects unfinished. In its second offshore attempt, however, development costs decreased 10 to 15 percent compared to onshore costs, large government-mandated regulatory projects completed in less time, and internal IT staff morale increased.

This experience shows how both the people involved in offshore projects and the projects themselves must be treated differently from internally developed projects. Even with the high complexities and intellectual property concerns surrounding embedded software development, the company has used these practices to establish processes that ensure successful delivery and protection of its intellectual property.

NASA's Exploration Agenda and Capability Engineering
pp. 63-73
Daniel E. Cooke, Matt Barry, Michael Lowry, and Cordell Green

Past NASA missions have been scripted, meaning that engineers developed hardware and software systems to respond mainly to foreseen circumstances, making them less able to handle unforeseen problems or exploration opportunities. The primary responsibility for handling unanticipated situations resides with humans, either onboard the spacecraft or in mission control.

The authors posit that software development will not achieve the level of hardware development without the introduction and use of model-based languages. Further, model-based development will not be widely accepted without risk analysis methodologies on par with those employed in hardware development.

A Layered Software Architecture for Quantum Computing Design Tools
pp. 74-83
Krysta M. Svore, Alfred V. Aho, Andrew W. Cross, Isaac Chuang, and Igor L. Markov

Despite convincing laboratory demonstrations of quantum information processing, it remains difficult to scale because it relies on inherently noisy components. Adequate use of quantum error correction and fault tolerance theoretically should enable much better scaling, but the sheer complexity of the techniques involved limits what is achievable today.

The authors propose a layered software architecture consisting of a four-phase computer-aided design flow that assists with such computations by mapping a high-level language source program representing a quantum algorithm onto a quantum device. By weighing different optimization and error-correction procedures at appropriate phases of the design flow, researchers, algorithm designers, and tool builders can trade off performance and accuracy.

Published by the IEEE Computer Society