# 50 & 25 YEARS AGO

EDITOR **ERICH NEUHOLD**
University of Vienna
erich.neuhold@univie.ac.at

## AUGUST 1972

In the early years, *Computer* was only published bimonthly. Therefore, we will have to skip our interesting and/or informative extractions for August. The next one will appear in the September 2022 issue of *Computer*, and we hope you will eagerly wait for our next publication of this column.

## AUGUST 1997

*https://www.computer.org/csdl/magazine/co/1997/08*

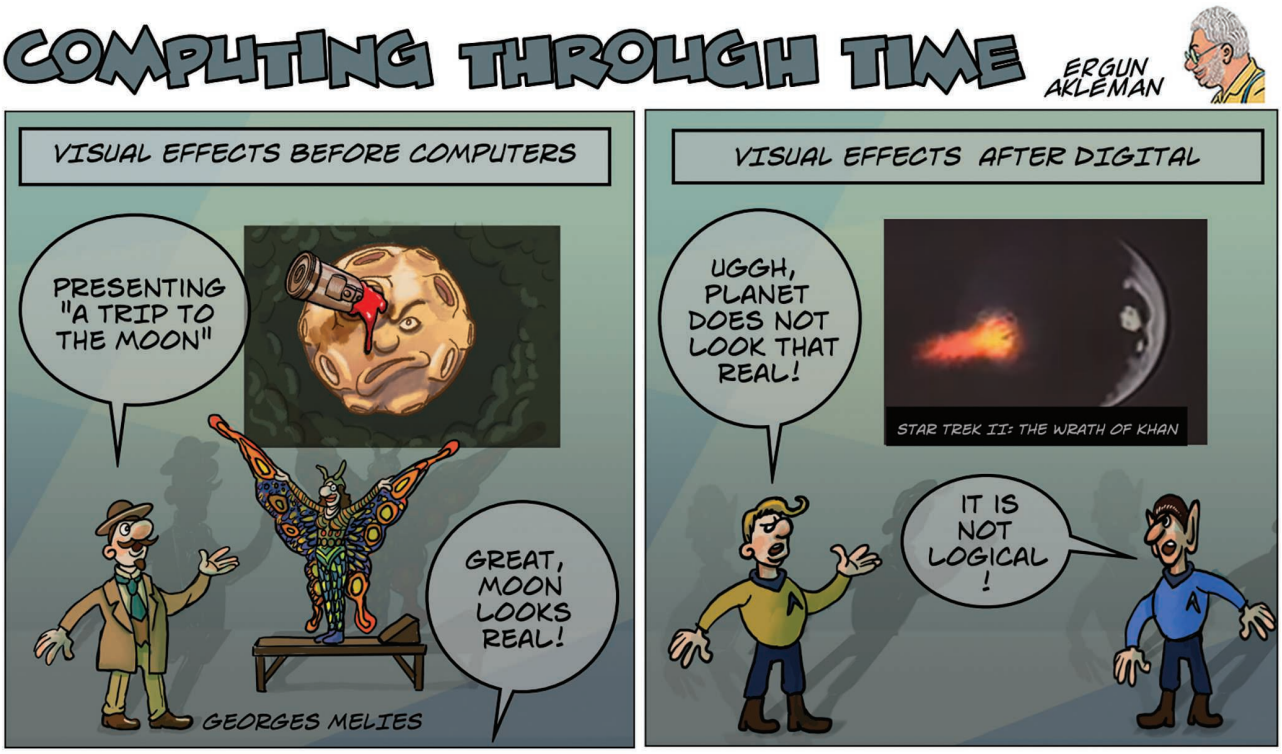
**Voice-Based Interfaces Make PCs Better Listeners; John Edwards** (p. 14) "Researchers have been refining voice recognition for close to 35 years. ... There are two types of voice-recognition systems—command-recognition systems and voice dictation systems." (p. 15) "Continuous-dictation systems, which let users speak to their computers at a normal rate in a nonstop stream of words, take advantage of faster processors and better speech-recognition algorithms, ..." (p. 16) "Researchers are trying to improve speech-recognition software. As the software improves and processors get speedier, voice-based interfaces will get faster and more accurate. They then may become commonplace in a couple of years." *[Editor's note: The "couple of years" actually took more like 15 to 20 years before natural speech processing became good enough for everyday use, as, for example, with the Alexa service. Of course, smartphones and such also were not thought of at that time.]*

**Reengineering with Reflexion Models: A Case Study; Gail C. Murphy and David Notkin** (p. 29) "Reengineering large and complex software systems is often very costly. Reflexion models let software engineers begin with a structural high-level model that they can selectively refine to rapidly gain task-specific knowledge about the source code. The authors describe how a Microsoft engineer used this technique in an experimental reengineering of Excel." (p. 35) "The engineer found it useful to be able to view the system in terms of a refined reflexion model, yet he also found it valuable to build up an understanding of how the high-level view connected to the source code. Consistent with previous studies on program comprehension, he moved between these levels repeatedly. ...We provided both textual and graphical interfaces to the reflexion model tools. Surprisingly, the engineer drove almost all the investigation of the reflexion model and the source code from textual information." *[Editor's note: This very interesting article explores the proposed tool using a software engineer and a real-world example (Excel). With this approach, it has been possible to verify the approach but also adopt changes to the model.]*

**10 Potholes in the Road to Information Quality; Diane M. Strong et al.** (p. 38) "Poor information quality can create chaos. Unless its root cause is diagnosed, efforts to address it are akin to patching potholes. This article describes 10 key causes, warning signs, and typical patches. ... Like potholes, IQ problems often arise unexpectedly and cause major negative consequences before they are resolved. ... If they know what to look for, organizations can anticipate and handle IQ problems before they trigger a crisis." (p. 39ff) "1. Multiple sources; 2. Subjective production; 3. Production errors; 4. Too much information; 5. Distributed Systems; 6. Nonnumeric information; 7. Advanced analysis requirements; 8. Changing task needs; 9. Security and privacy requirements; 10. Lack of computing resources." *[Editor's note: The article analyzes 10 information-quality issues that frequently arise, even today, and proposes how to better control/avoid them. A worthwhile article to read.]*

**Fighting Complexity in Computer Systems; Alexander D. Stoyen** (p. 47) "From the earliest hardware and software through the latest, complexity has been something computer engineers, scientists, and programmers have had to fight. Important fields of research and technology have originated, developed, or matured as side effects of this fight. ... Generally speaking, increased sophistication of hardware or low-level software (the OS or language services) has led to an increase in the complexity of high-level software (the application). ...

*Digital Object Identifier 10.1109/MC.2022.3176987*
*Date of current version: 2 August 2022*




GEORGES MELIES (1861–1938), A FRENCH FILMMAKER, POPULARIZED SPECIAL EFFECTS TECHNIQUES IN MOVIES, SUCH AS SUBSTITUTION SPLICES, MULTIPLE EXPOSURES, TIME-LAPSE PHOTOGRAPHY, DISSOLVES, AND HAND-PAINTED COLOR. ONLY AFTER 1980, COMPUTERS STARTED TO BE USED FOR SPECIAL EFFECTS. *STAR TREK II: THE WRATH OF KHAN* MADE IN 1982 WAS THE FIRST FEATURE FILM CONTAINING A COMPLETELY-COMPUTER-GENERATED (CGI) CINEMATIC IMAGE SEQUENCE. THE SEQUENCE LASTED 60 SECONDS.

*Digital Object Identifier 10.1109/MC.2022.3176992*
*Date of current version: 2 August 2022*

The three articles included in this theme issue present very different approaches to fighting complexity." *[Editor's note: See the following three article extracts and my comments.]*

**Applying Software Product-Line Architecture; David Dikel et al.** (p. 49) "Product-line architecture not only reduces the complexity and cost of developing and maintaining code, but also streamlines the production of documentation, training materials, and product literature. ... Only in conjunction with appropriate organizational behaviors can software architecture effectively control project complexity." (p. 50) "a set of six organizational principles believed critical to the long-term success of a software architecture: • Focusing on simplification, minimization, and clarification. • Adapting the architecture to future customer needs, technology, competition, and business goals. • Establishing a consistent and pervasive architectural rhythm. • Partnering and broadening relations with stake-holders. • Maintaining a clear architecture vision across the enterprise. • Proactively managing risks and opportunities." (p. 54) "Our advisers helped us develop six critical organizational principles, and our study of Nortel confirmed each of them." *[Editor's note: The analysis in this article of these six principles clearly shows the benefit for each of them. Unfortunately, even today many development teams only haphazardly follow those principles.]*

**Using Genetic Algorithms to Design Mesh Networks; King-Tim Ko et al.** (p. 56) "Designing mesh communication networks is a complex, multiconstraint optimization problem. The design of a network connecting 10 Chinese cities demonstrates the elegance and simplicity that genetic algorithms offer in handling such problems." ... To formulate a genetic algorithm method for mesh communication networks, we needed to define the essential network architecture and the design parameters. We based our design on a well-known problem formation for packet-switched communications networks." (p. 59) "Our approach breaks the problem of network design into three optimization processes—for topology, routing, and capacity—and applied the genetic algorithm technique to each." *[Editor's note: Through this detailed analysis, the authors show that using genetic algorithms leads to better results than using other known algorithms. Looking at the literature, they are not the only ones, but clearly early ones, to use the genetic algorithm approach.]*

**Object Structures for Real-Time Systems and Simulators; K.H. (Kane) Kim** (p. 62) "The ideal representation (or modeling) scheme should be effective not only for abstracting system designs but also for representing the application environment. It should also be capable of manipulating logical values and temporal characteristics at varying degrees of accuracy. ... The object structure, which evolved out of the abstract concept formulated by Hermann Kopetz and myself, has a concrete syntax structure and execution semantics. It is called TMO (time-triggered message-triggered object; formerly called RTO.k). TMO promotes the uniform, integrated design of real time, distributed systems and the real time simulators

of their application environments." *[Editor's note: Using an extensive example of a military command and control system (CAMIN), the author shows the applicability and advantage of the TMO approach. TMO is still around, and over the years it has been shown to be a useful tool for real-time system design using object-oriented technology.]*

**Could LDAP Be the Next Killer DAP?; Charles Severance** (p. 88) "LDAP (*Ed: Lightweight Directory Access Protocol*) is a protocol that allows a program such as a browser or an e-mail package to perform directory lookups across a wide variety of directories, even if they run on different operating systems and directory environments." *[Editor's note: The author argues that LDAP will be the "Rosetta Stone" for directory access. It may not have been the Rosetta Stone, but it undeniably is still around and in use today.]*

**Good Enough Quality: Beyond the Buzzword; James Bach** (p. 96) "In this article, I'd like to examine one actual practice that is finally emerging as a describable method: Good Enough Software." (p. 97) "One reason why Good Enough is not better described may be that it is so much a part of our experience that it seems too obvious to mention. Only when contrasted with idealistic, normative models of software engineering does Good Enough stand out as a separate paradigm" (p. 98) "I hope the framework makes it clear that Good Enough has nothing to do with mediocrity. It has to do with rational choices, as opposed to compulsive behavior. If something really is good enough, in terms of this framework, then further improvement means making an investment that has an inadequate return." *[Editor's note: A very interesting article that rationalizes the "Good Enough" idea as something that is based on our knowledge that software systems will contain bugs and that we have to live with them.]*

**The Allegory of the Humidifier: ROI for Systems Engineering; Mark E. Sampson** (p. 104) Systems engineering makes sense, but its financial benefits are often hard to pin down. Although many high-cost, project runaways stem from mistakes in requirements definition, accountants and managers continue to balk at investing in cost avoidance, risk reduction, and customer understanding. They prefer making decisions based on hard-to-peg return-on-investment (ROI) numbers. Put another way, systems engineers want to avoid risk (future costs), while accountants want to save money today. What follows is a true story that represents a simple but poignant example of the hidden costs of not doing systems engineering." *[Editor's note: The example concerns humidifiers bought at Sears instead of professional equipment needed for a static plotter room. The follow-up costs far outweighed the cost of a professional solution. This short allegory is well worth reading.]*



*Digital Object Identifier 10.1109/MC.2022.3188066*