50 & 25 YEARS AGO

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

FEBRUARY 1971

In the early years, *Computer* was only published bimonthly. Therefore, we will have to skip our interesting and/or informative extractions for February. The next one will appear in the March 2021 issue of *Computer*, and we hope you will eagerly wait for our next publication of this column.

FEBRUARY 1996

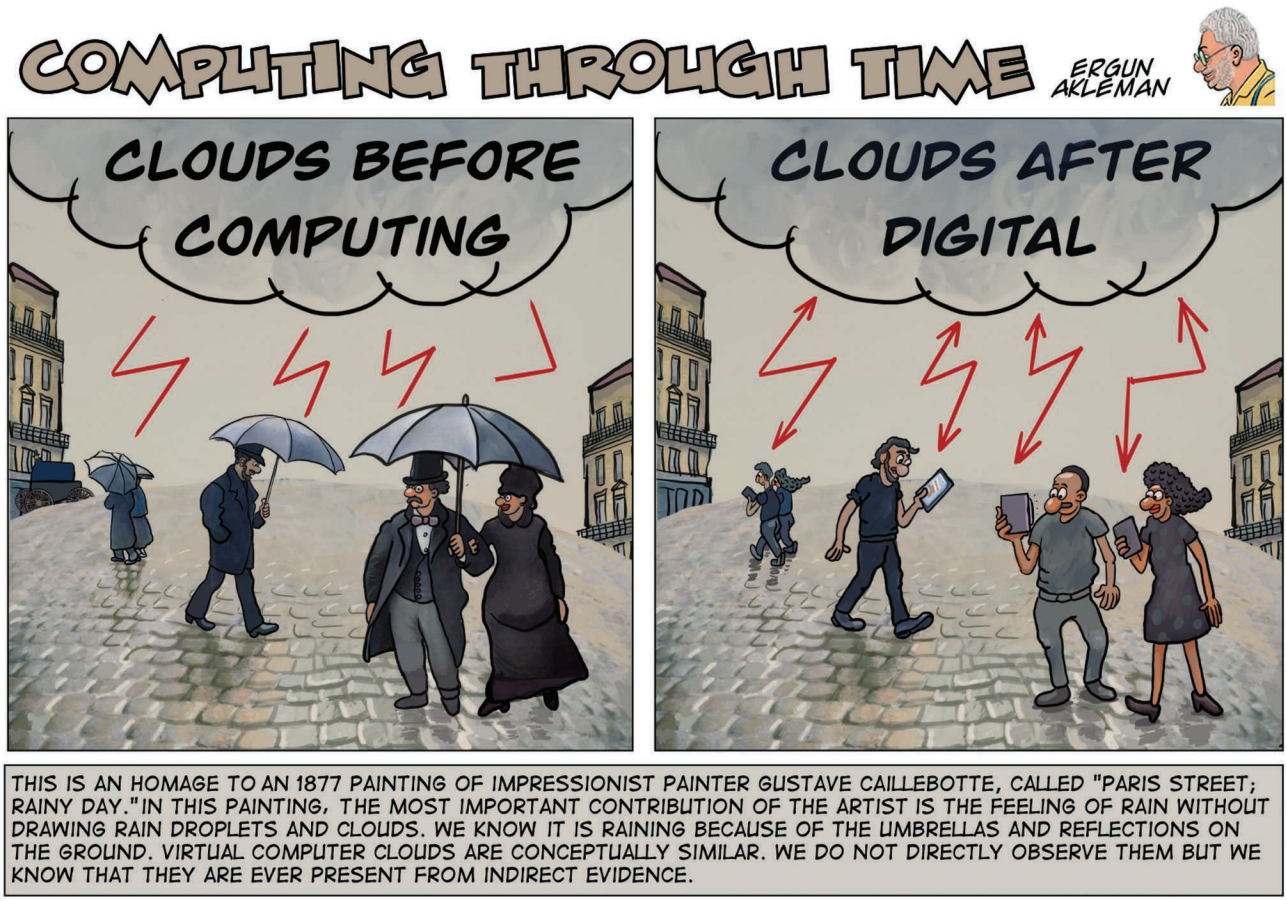
https://www.computer.org/csdl/magazine/co/1996/02

Immigration Issue Divides US Computer Industry; John Sterlicchi et al. (p. 10) "The US Labor Department and the coalition, which is headed by the US Activity Division of IEEE USA, claim foreign workers take jobs from US citizens because they work for lower salaries. ... On the other side such industry powerhouses as Intel, Hewlett Packard and Sun tell politicians that foreign workers are vital. The companies contend ... that they cannot find properly trained US workers ... The committee founder is Lawrence Richards, who quit IBM and set up SoftPac last year after many colleagues were laid off and replaced by lower paid programmers from India." *[Editor's note: Unfortunately, the trend to get rid of usually higher-paid (older) employees to replace them with usually lower-paid (younger) employees, whether foreigners or citizens, has been going on continuously all over the corporate world. This controversy is still around today (hear President Trump) but far from being resolved as long as profit remains the dominant decision factor.]*

TreadMarks: Shared Memory Computing on Networks of Workstations; Cristiana Amza et al. (p. 18) "Shared memory facilitates the transition from sequential to parallel processing. Since most data structures can be retained, simply adding synchronization achieves correct, efficient programs for many applications. ... In terms of performance, networked workstations approach or exceed supercomputer performance for some applications. These loosely coupled multiprocessors will by no means replace the more tightly coupled designs ... However, advances in networking technology and processor performance are expanding the class of applications that can be executed efficiently on networked workstations. ... In this article, we discuss our experience with parallel computing on networks of workstations using the TreadMarks distributed shared memory (DSM) system." (p. 20) "Two simple problems (larger applications are discussed later) illustrate the TreadMarks APL Jacobi iteration (Figure 3) shows the use of barriers, and the traveling salesman problem (Figure 4) shows the use of locks." *[Editor's note: Distributed processing problems have been around forever and are still here; just think of the network, grid, edge, and fog computing buzzwords and the huge computing farms around. This article provides a very good analysis of the behavior of DSMs for a number of nontrivial applications. However, none of them are prone to the map-reduce type of solutions so frequently discussed today.]*

Improving System Usability Through Parallel Design; Jakob Nielsen et al. (p. 29) "Unfortunately, testing and redesigning take time, thus delaying product release. Because major delays are intolerable, much effort has gone into improving user interface design efficiency, prototyping, and evaluation. ... To yield final designs faster, we want parts of the usability engineering life cycle to take place at the same time, in a process we call parallel user interface design. ... A weakness of parallel design is the waste of resources when several designers do the same work, even though some design ideas will not be used. ... Therefore, parallel design is best suited for projects where reduced time-to-market is essential and makes the up-front investment acceptable." (p. 30) "The case study concerned screen-based user interfaces to advanced telephone services like call forwarding, where incoming calls are routed to another telephone, and call waiting, where you are notified if somebody calls while you are on the line." (p. 34) "In our project, parallel design was 73% more expensive than iterative design. We still recommend parallel design because it achieves major usability improvements

Digital Object Identifier 10.1109/MC.2020.3041888
Date of current version: 11 February 2021



Digital Object Identifier 10.1109/MC.2020.3043063
Date of current version: 11 February 2021

very fast." *[Editor's note: I believe the case studied—user interface improvements—is a typical one where alternatives can easily be tested in parallel and the results then integrated into the "final" design. Unfortunately, only usability improvements have been evaluated and not the time saved, as was the original claim.]*

Role-Based Access Control Models; Ravi S. Sandhu et al. (p. 38) "A family of increasingly sophisticated models shows how RBAC works. ... A role can represent specific task competency, such as that of a physician or a pharmacist. A role can embody the authority and responsibility of, say, a project supervisor. Authority and responsibility are distinct from competency." (p. 40) "To explore RBAC's various dimensions, we have defined a family of four conceptual models. Figure la shows the model relationships and Figure 1b portrays their essential characteristics. $RBAC_0$, as the base model at the bottom, is the minimum requirement for an RBAC system. Advanced models $RBAC_1$ and $RBAC_2$ include $RBAC_0$, but $RBAC_1$ adds role hierarchies (situations where roles can inherit permissions from other roles), whereas $RBAC_2$ adds constraints (which impose restrictions on acceptable configurations of the different components of RBAC). $RBAC_1$ and $RBAC_2$ are incomparable to one another. The consolidated model, $RBAC_3$, includes $RBAC_1$ and $RBAC_2$ and, by transitivity, $RBAC_0$." *[Editor's note: The model includes many aspects of access control but stays on a rather abstract level. It would have been helpful if a sample case had been included.]*

Logical Time: Capturing Causality in Distributed Systems; Michel Raynal et al. (p. 49) "Causality—determining which event happens before what others—is vital in distributed computations. Distributed systems can determine causality using logical clocks. ... The notion of time is basic to capturing the causality between events. However, distributed systems have no built-in physical time and can only approximate it. Even the Internet's Network Time Protocols,[1] which maintain a time accurate to a few tens of milliseconds, are not adequate for capturing causality in distributed systems. ... This article presents a general framework of a system of logical clocks in distributed systems and discusses three

methods—scalar, vector, and matrix—for implementing logical time in these systems." *[Editor's note: This is a very interesting article that discusses in detail the properties of the various logical time methods and refers to the original papers where these concepts were introduced.]*

Why Software Jewels Are Rare; David Lorge Parnas (p. 57) "Occasionally, I find a real jewel, a well-structured program written in a consistent style, free of kludges, developed so that each component is simple and organized, and designed so that the product is easy to change. ... Most of the software we see or buy is ugly, unreliable, hard to change, and certainly not something that Wirth or Dijkstra would admire." (p. 58) "Often, software has grown large and its structure has degraded because designers have repeatedly modified it to integrate new systems or add new features. ... Offered a jewel or a more useful tool, most customers choose utility. To sell products, you have to add the features the market demands." (p. 59) "Indeed, we'd all do better if we could start with all the knowledge we will have later when a product is mature. Unfortunately, commercial designers don't have that chance very often. ... When Wirth asks, rhetorically, how Oberon could be so small, he doesn't give the whole answer. The Oberon design team obviously learned a great deal from the mistakes of others, and those others have not had a chance to return the compliment." (p. 60) "My engineering teachers laid down some basic rules: 1. Design before implementing. 2. Document your design. 3. Review and analyze the documented design. 4. Review implementation for consistency with the design." *[Editor's note: David Parnas, when discussing this issue, raises many valid points why software does not follow the ideals Wirth and Dijkstra promote. However, we would all be much better off if today's designers would always follow the four development rules presented. Unfortunately, methods like Power Programming and today's development tools often stand in the way of such processes.]*

Automated Object Design: The Client-Server Case; Philippe Desfray (p. 62) "The most difficult aspect of large-scale applications development is not programming but technical design. This article explores a methodology that formalizes and automates object-based technical design in the domain of information management systems. ...So we developed a new methodology, called hypergenericity, which formalizes and automates object-based technical design. ... To be totally applicable, the methodology demands that every model element—class, attribute, method, parameter, and so forth—be annotated. The annotations, or directives, are named *@name*." (p. 65) "The implementation relies on a class library, encapsulating monitor accesses and factoring in processing such as error management. In this manner, monitors can be changed without extensive rule modifications. (The same logic for library construction is applied, whether the code is hypergeneric or manually produced.)" *[Editor's note: This article investigates and suggests a solution to "automatic" code generation from abstract object-oriented specifications. It discusses, using a complex example, the problems arising when doing that. Of course, today's programming also relies heavily on parameterized class libraries, especially when producing new apps for tablets and smartphones.]*

Software Change Management; Capers Jones (p. 80) "Modern change management, or configuration control, tools must encompass changes affecting every kind of software deliverable and artifact: requirements, project plans, project cost estimates, contracts, design, source code, user documents, illustrations and graphics, test materials, and bug reports. Ideally, these tools would use hypertext to handle cross-references among deliverables so that when something changes, corresponding material is modified appropriately." (p. 82) "Change management is one of the most important aspects of successful software development. Evidence of this fact are the new companies building integrated change management tools that handle much more than source code revisions. Certainly, function-point metrics, which quantify the costs of change with a previously impossible precision, are partly behind the emergence of these companies." *[Editor's note: The article suggests that change management should not be concerned just with design and implementation issues but also with changes in all of the accompanying other documents.]*