50 & 25 YEARS AGO

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

FEBRUARY 1970

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

FEBRUARY 1995

www.computer.org/csdl/mags/co/1995/02/index.html

Is the Macintosh Dead Meat? (p. 6) "If you believe what you read, Microsoft will scorch a path to desktop domination. ... It's irresponsible for anyone to claim victory months or years before a product is released. So I will add my two cents' worth to the pot and stir it around: The 32-bit operating system wars mean everything, and they mean nothing! ... IBM is claiming strong support for OS/2, citing a 20 percent demand for OS/2 Warp among OEMers. ... It seems that killing Apple will be more difficult than the press pundits would have us believe." (p. 7) "Now comes the surprise ending. The sleeper column of Table 1 is full of players who can upset the platform cart. If NeXT, Taligent, or Microsoft successfully covers all major operating systems with a thick layer of application frameworks, as each is attempting to do, then the underlying OS no longer makes any difference." *[Editor's note: Despite this concluding statement, it turned out that Microsoft and Apple are practically the only operating systems covering the PC and laptop market. Only the appearance of smartphones allowed Google's Android to establish itself on the market.]*

Reducing the Time to Market Through Rapid Prototyping (p. 14) "The term Rapid System Prototyping (RSP) signifies the need to develop systems in significantly less time or with significantly less effort than is currently possible, and thus provides the context for the driving problem in the design community in the present and for years to come. ... Activities in the RSP community break down into three substantial areas of research: formalizing the design process so that a system can be described by formal methods ... developing CAD tools that can synthesize a system expressed in a formal language ... reducing these two steps to practice. ... The articles in this issue of *Computer* reflect the diversity and state of the field."

Real-Time Image Processing on a Custom Computing Platform (p. 16) "The authors explore the utility of custom computing machinery for accelerating the development, testing, and prototyping of a diverse set of image-processing applications. ... Due to the enormous processing time required to simulate a complex image-processing system, executing a VHDL model with a representative data set even on a fast workstation is not practical. Days, or even weeks, are commonly needed to simulate the processing of a single full-sized image. ... With our system, a designer can proceed from a behavioral description of the image-processing task to a functioning prototype that can perform the task at full speed (rapid prototyping)." (p. 17) "Splash-2 is an attached processor featuring programmable processing elements (PEs) and communication paths. The Splash-2 system uses arrays of RAM-based field-programmable gate arrays (FPGAs), crossbar networks, and distributed memory to accomplish the needed flexibility and performance." (p. 19) "Figure 2 shows the VTSplash laboratory system we developed. A video camera or a VCR creates a standard RS-170 video stream. The signal produced from the camera is digitized with a custom-built frame-grabber card. This board not only captures images but also performs any needed sequencing or simple pixel operations before the data are presented to Splash-2." (p. 20) "Although Splash-2 represents the state-of-the-art in custom computing processors—both in hardware capabilities and software support—it requires a substantial time investment to develop an application. To make this class of machinery more widely accepted and cost-effective, methods must be developed to reduce application-development time. Several promising endeavors focus on this issue. ... Image operations have been classified into five generic classes." *[Editor's note: This article explores these five classes and*

Digital Object Identifier 10.1109/MC.2019.2957560
Date of current version: 12 February 2020

demonstrates that the flexible architecture of Splash-2 allows those applications to vastly outperform general-purpose machines like the Sun SparcStation. Unfortunately, it does not illustrate any of the gains in development time in the sense of rapid system prototyping.]

RPM: A Rapid Prototyping Engine for Multiprocessor Systems (p. 26) "The RPM (Rapid Prototyping engine for Multiprocessors) project is exploring a rapid-prototyping methodology for multiprocessor systems that is based on hardware emulation. The flexibility of emulation is important, since the design space for multiprocessor systems is arguably much wider than that of uniprocessors. ... For programming ease, the shared-memory model has thus far been the favored transition path from uniprocessors to multiprocessors. On the other hand, message-passing systems are generally perceived as more scalable than shared-memory systems." (p. 27) "Several technologies, including field-programmable gate arrays (FPGAs), and efficient computer-aided design (CAD) tools are converging, making it possible to build and program flexible multiprocessor emulators." (p. 30) "Since the speeds of the hardware emulation and target differ, timing measured on the emulator must be related to the timing in the target machine. ... Time scaling preserves the relative timing of components in the emulator and target, and simple scaling arguments yield absolute times in the target." (p. 33) "Finally, as with any hardware, an emulator's efficiency advantage over simulation erodes every year, as faster workstations and PCs are introduced. Nevertheless, given RPM's current speed, we expect that it will remain competitive with software simulators for at least 10 years." *[Editor's note: It is interesting to observe how limited execution speed and storage size had been in 1995. As the authors confess, the limited budget of a university also hampered the size of the multiprocessor systems that can be emulated.]*

Grape-II: A System-Level Prototyping Environment for DSP Applications (p. 35) "Grape-II has been used successfully in three real-world DSP applications. Its structured prototyping methodology reduces programming effort: its use of general-purpose reusable hardware minimizes development cost. ... Digital signal processing is used for speech synthesis and recognition, telecommunications, image and video processing, and robotics, as well as for consumer products, i.e., compact disk players, digital audio tape recorders, and digital radio receivers. The increasing complexity and data rates of these DSP applications demand application-specific integrated circuits. ... The general-purpose hardware consists of commercial DSP processors, bond-out versions of core processors, and field-programmable gate arrays (FPGAs) linked to form a powerful, heterogeneous multiprocessor." (p. 40) "**Example: Rapid prototyping a video encoder**. As an example, let's look at the rapid prototyping of a video encoder that generates compressed video for a wireless local area network. We'll go through the steps we followed in prototyping the video encoder component, but first, let's look at the entire video-on-demand system." (p. 42) "This case study resulted in an operational prototype at full speed. It shows the feasibility of our strategy for prototyping real time color video compression on a commercial DSP multiprocessor." *[Editor's note: This article provides a quite detailed description of the various tasks involved in building an actual prototype within the architecture proposed.]*

Synthesis Steps and Design Models for Codesign (p. 44) "Growing design complexity and an urgent need for early prototypes have become limitations in electronic systems design. A mature codesign offers advantages to overcome this challenge. ... The fields of specification, design, and synthesis of mixed hardware/software systems are becoming increasingly more popular. The renewed interest in codesign is driven by increasing design complexity and the need for early prototypes to validate the specification and provide the customer with feedback during the design process. ... The main steps needed to transform a system-level specification into a mixed hardware/software specification are system partitioning, communication synthesis, and architecture generation." (p. 45) "Cosmos, a codesign environment, starts from the system-level specification language SOL and produces a heterogeneous architecture including hardware descriptions in VHOL and software descriptions in C." (p. 51) "A codesign example: Figure 8 illustrates the overall codesign process for a real time system interface (RTSI) between sensor-producing digital signals and a storage disk. As explained above, the codesign process includes several steps and intermediate models. The input to Cosmos is an SDL description of this system, while the output is a mixed C/VHDL description that can be mapped onto a target architecture to produce a prototype. ... Our goal is to let the user do the intelligent work (such as deciding which part is in software and which is in hardware) and let the system perform the fastidious and error-prone tasks." *[Editor's note: As in most of those systems, the key issue is the existence of an abstract precise model of the system to be designed. Even then, as this article points out, many decisions are left to the human.]*

A Formal Approach to Managing Design Processes (p. 54) "Product development cycles keep speeding up. To cope with the demands of ever-shorter design cycles, a methodology management system controls the design process during rapid prototyping." (p. 56) "Our prototype system uses a methodology specification based on process flow graphs and design process grammars. Process flow graphs describe the information flow of a design process, and graph grammars are a means for transforming high-level process flow graphs into progressively more detailed flow graphs." (p. 58) "Figure 4 illustrates the architecture of our proposed system, which applies the theory developed in the previous section. Decisions to select or invoke tools are split between the designer and a set of manager programs, with manager programs making the routine decisions and the designer making decisions

that require higher-level thinking." *[Editor's note: As in the other articles in this issue, the tool very much depends on the precise (formal) description of the intended system to be prototyped. The claim here is that modifications during the prototyping process are easily accomplished through a strong modularization of the design.]*

A Plea for Clean Software (p. 64) "About 25 years ago, an interactive text editor could be designed with as little as 8,000 bytes of storage. (Modern program editors request 100 times that much!) An operating system had to manage with 8,000 bytes, and a compiler had to fit into 32 Kbytes, whereas their modern descendants require megabytes. Has all this inflated software become any faster? On the contrary. Were it not for a thousand times faster hardware, modern software would be utterly unusable. ... With a touch of humor, the following two laws reflect the state of the art admirably well—Software expands to fill the available memory (Parkinson),—Software is getting slower more rapidly than hardware becomes faster (Reiser)." (p. 65) "Initial designs for sophisticated software applications are invariably complicated, even when developed by competent engineers. Truly good solutions emerge after iterative improvements or after redesigns that exploit new insights. ... Instead, software inadequacies are typically corrected by quickly conceived additions that invariably result in the well-known bulk." (p. 66) "Between 1986 and 1989, Jurg Gutknecht and I designed and implemented a new software system—called Oberon—for modern workstations, based on nothing but hardware. ... The system includes: storage management, a file system, a window display manager, a network with servers, a compiler, and text, graphics, and document editors." (p. 67) "The strategy of keeping the core system simple but extensible rewards the modest user. The Oberon core occupies fewer than 200 Kbytes, including editor and compiler. A computer system based on Oberon needs to be expanded only if large, demanding applications are requested, such as CAD with large memory requirements. If several such applications are used, the system does not require them to be simultaneously loaded." (p. 68) "With Project Oberon we have demonstrated that flexible and powerful systems can be built with substantially fewer resources in less time than usual. The plague of software explosion is not a 'law of nature.' It is avoidable, and it is the software engineer's task to curtail it." *[Editor's note: The statements of this article hold a lot of truth that applies even today. What would the author say about today's systems? The megabyte systems of 1995, the ones he complains about, have become gigabyte systems today. Despite all of the incredible speedup in compute power, Windows needs the same time or even longer for startup than it did 25 years ago.]*

Determining Software Schedules (p. 73) "But now we can measure these factors with reasonable accuracy and collect empirical data on both 'average' and 'best-in-class' results. We are particularly interested in the wide performance gaps between laggards, average enterprises, and industry leaders, as well as differences among the various software domains. ... Although many commercial software cost-estimating tools can predict schedules with fairly good accuracy, as of 1994 only about 15 percent of US software managers—and even less abroad—were using them. ... Not only are both ends of software projects ambiguous and difficult to determine, but the middle can get messy, too. Even with the 'waterfall model' of development, there is always overlap and parallelism between adjacent activities, so that a project's end-to-end schedule is never the same as the duration of those activities. Software requirements are usually only about 75 percent defined when design starts. Design is often little more than 50 percent complete when coding starts. Integration and testing can begin when coding is less than 20 percent complete. And user documentation usually starts when coding is about 50 percent finished." (p. 74) "But the function-point metric provides a convenient, quick estimator for schedule durations that can be applied early in a software project's development cycle." *[Editor's note: The article claims that a simple metric based on function points will give the projects duration rather accurately. However, it does not explain how the "power level" numbers are derived, which are instrumental for this calculation, especially since the later part of the article clearly distinguishes between the four chosen application fields and their requirements.]*

Open Channel: Can System Integrators Learn From Baggage Crisis? (p. 112) "This was a tantalizingly detailed look at the system testing problems with the automated airport baggage handling system in Denver. ... I submit to the systems engineering community that the pending government investigations represent a once-in-a-decade opportunity to assemble a benchmark/definitive study of a major system integration/software development effort. ... It can become the cornerstone of a systems engineering knowledge base. Names of the perpetrators could be sanitized so as not to needlessly jeopardize their careers." *[Editor's note: I do not know whether the suggested analysis actually took place, but lessons learned in the analysis of such a "disastrous" failure could clearly have helped to improve methodologies of system development. However, looking at the recent large systems failures that seem to occur with ever-increasing frequency, such lessons have not really been adopted into our complex systems development of today.]*