50 & 25 YEARS AGO

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

OCTOBER 1972

In the early years, *Computer* was published only bimonthly. Therefore, we will have to skip our "interesting and/or informative" extractions for the October issue. The next one will appear in the November 2022 issue of *Computer*, and we hope you will eagerly await our next publication of this column. In some cases these needs are real, but often they are fanciful and lack any sound rationale.

OCTOBER 1997

https://www.computer.org/csdl/magazine/co/1997/10

Will New Fiber Technology Quench the Thirst for Bandwidth?; David Clark (p. 14) "Researchers say DWDM (*Ed: Dense wavelength-division multiplexing*) could help quench the world's increasing thirst for bandwidth. … While less expensive than new fiber, DWDM is still expensive to implement." (p. 15) "A 16-wavelength DWDM system costs just under $1 million for each node on a network segment, or just under $2 million for each point-to-point connection." *[Editor's note: All in all, as we now know, WDM, DWDM, and later, CWDM, became successful optical transmission technologies for core and metro networks, much as they were foreseen in this article.]*

Dawn of the Internet Appliance; George Lawton (p. 16) "Developers can now put Internet connectivity in a variety of devices, including factory machinery, VCRs, and handheld appliances, such as personal digital assistants. … Internet-enabled appliances, which began to appear in 1995, communicate in a variety of ways, such as by using wireless or modem technology." (p. 18) "POTENTIAL BENEFITS: Internet appliances would create many benefits for users. … Embedded Internet servers could then make it easy to remotely access important information, and monitor and control devices. … POTENTIAL PROBLEMS: Hackers could break into Internet-enabled devices and change settings or learn private information about users. … They could also insert programs that would display odd messages on a user's TV or shut down an entire device. … Harmon said the telephone may be one of the most profitable Internet appliances, because it is used throughout the world." *[Editor's note: What a true prediction 25 years ago about Internet-enabled (smart) devices and smart telephones but also about the risks that are all too present today and will stay with us for the time being.]*

Deep Blue's Hardware-Software Synergy; Scott Hamilton et al. (p. 29) "It's true that powerful hardware forms the basis of the chess computer's capabilities. However, improved software-based search techniques and innovations in partitioning the problem for a multiprocessor also played important roles. … Thus it's not sheer speed alone—it's the ability to manage the complexity of searching these combinations. Deep Blue does this via an optimized game tree search tuned with expert knowledge." (p. 31) "In manual tuning, grand master Joel Benjamin would play a form of take-back chess until he felt Deep Blue had misevaluated a particular position. … The team could then alter the parameter and change the position to see if the evaluation also changed." *[Editor's note: The article analyzes the strategy that was used by the IBM team to defeat Garry Kasparov in this 1997 game. It relies heavily on adjusting the evaluation parameter with the help of Grand Masters between the games and is mostly based on searching strategies, not on deep learning mechanisms.]*

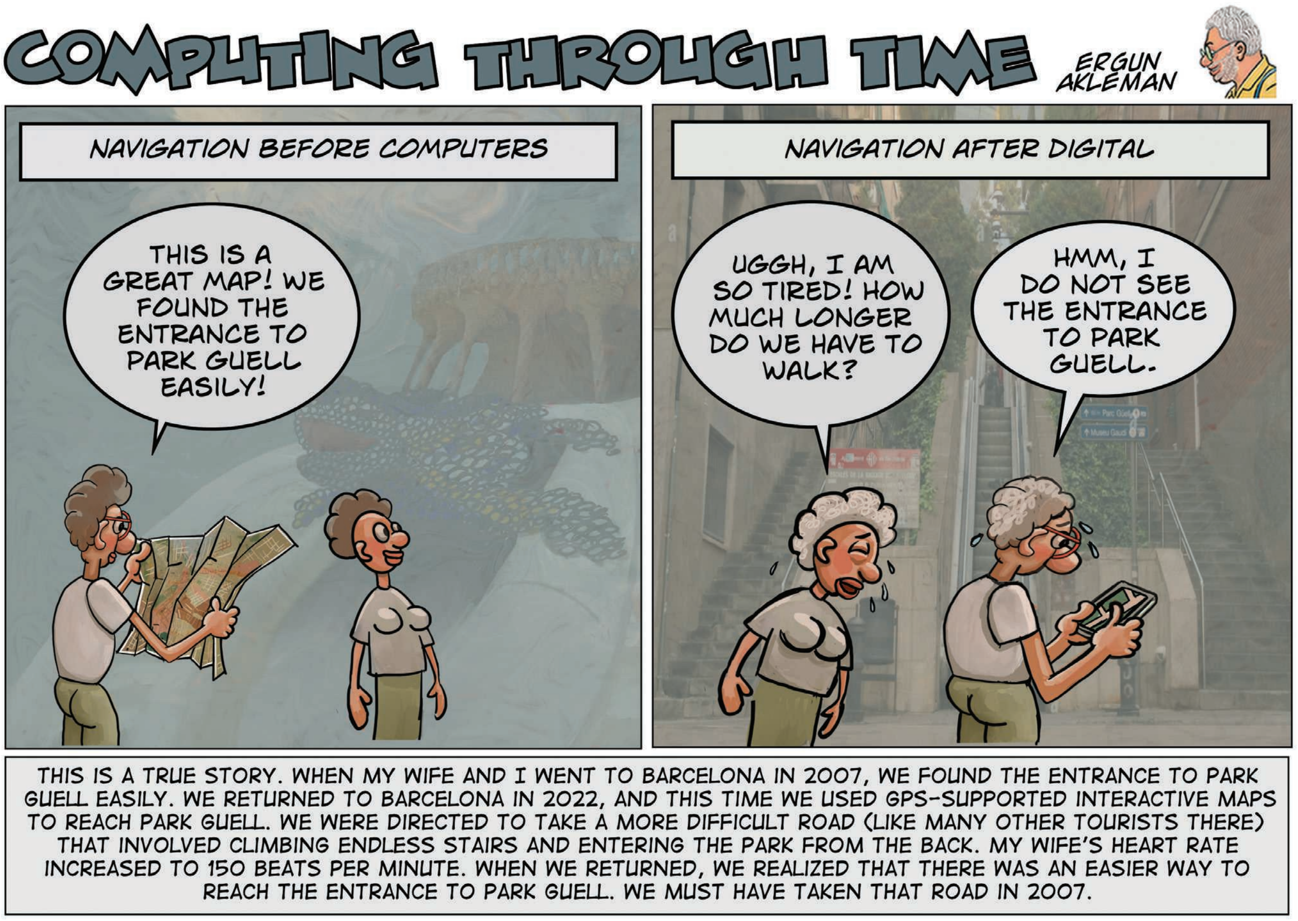
Making the Reuse Business Work; Ivar Jacobson et al. (p. 36) "Reuse technology is ready now. In fact, enough companies have demonstrated substantial improvement, often as much as 90 percent reuse, to assure us that it can be achieved." (p. 37) "The integration that a systematic reuse program involves means that only management can lead it, which means that the upper management (division head and department directors) of a division has to understand what reuse can accomplish and how to go about it." (p. 38) "This ever-increasing complexity is inherent in large-scale reusable systems. It cannot be sidestepped. It requires a plan. This plan is called the architecture. … When an organization plans a family of large systems and expects

Digital Object Identifier 10.1109/MC.2022.3192699
Date of current version: 26 September 2022

substantial software reuse, it will need component systems reusable in dozens of systems for years to come. This requires coming to terms with significant planning, and specifically, with architecture." (p. 40) "At the same time, object-oriented ways of thinking can be a great aid to carrying through the software engineering processes that enable reuse to take place." *[Editor's note: This very interesting article, taken from a book published by the same authors, explains the need for software reuse and then follows with many good ideas on how to modify business and software architectures to enable wide reuse of software components. Today of course, that is not only well established in complex system development, but also in the many libraries that enable, via reuse, the rapid development of apps in today's Internet world.]*

Guest Editor's Introduction: Supporting Reuse With Object Technology; David C. Rine (p. 43) "Is there a way to develop software that is of higher quality and yet takes less time and effort to produce? Many software developers believe there is, through reusing high-quality, tested software already developed." (p. 44) "In the past 10 years, a conjecture has taken hold that says that object technology supports reuse. The studies we examined now show that object technology can successfully support reuse in three ways: • Distributed objects support client-server reuse, through Object Request Brokers (ORBs)—a part of the Common Object Request Broker Architecture (CORBA). • Object modeling is appropriate for the development of domain models (including object-oriented frameworks and patterns), notably via the Unified Modeling Language (UML). • Object-based components can be made portable and interoperable, using Java, DCOM, and similar technologies." *[Editor's note: The following four articles investigate different aspects of software reuse. All of them exploit object-oriented (OO) technology, and I will only very briefly outline their main focus but encourage the reader to explore the articles as they illustrate the early struggle to reuse software. Today, it is a common technique to increase quality of the software and efficiency of the development process.]*

Object Technology and Reuse: Lessons From Early Adopters; Robert G. Fichman et al. (p. 47) "Four longitudinal case studies illustrate the costs and risks of early OO adoption, including the difficulty of achieving systematic reuse in practice. The authors recommend general strategies and specific tactics for overcoming adoption barriers." (p. 55) "LESSONS LEARNED: • Invest in organizational learning • Develop a complete architecture • Hire an OO architect • Simplify architectural demands • Limit initial development • Find proven


COMPUTING THROUGH TIME
ERGUN AKLEMAN
NAVIGATION BEFORE COMPUTERS
THIS IS A GREAT MAP! WE FOUND THE ENTRANCE TO PARK GUELL EASILY!
NAVIGATION AFTER DIGITAL
UGGH, I AM SO TIRED! HOW MUCH LONGER DO WE HAVE TO WALK?
HMM, I DO NOT SEE THE ENTRANCE TO PARK GUELL.
THIS IS A TRUE STORY. WHEN MY WIFE AND I WENT TO BARCELONA IN 2007, WE FOUND THE ENTRANCE TO PARK GUELL EASILY. WE RETURNED TO BARCELONA IN 2022, AND THIS TIME WE USED GPS-SUPPORTED INTERACTIVE MAPS TO REACH PARK GUELL. WE WERE DIRECTED TO TAKE A MORE DIFFICULT ROAD (LIKE MANY OTHER TOURISTS THERE) THAT INVOLVED CLIMBING ENDLESS STAIRS AND ENTERING THE PARK FROM THE BACK. MY WIFE'S HEART RATE INCREASED TO 150 BEATS PER MINUTE. WHEN WE RETURNED, WE REALIZED THAT THERE WAS AN EASIER WAY TO REACH THE ENTRANCE TO PARK GUELL. WE MUST HAVE TAKEN THAT ROAD IN 2007.

Digital Object Identifier 10.1109/MC.2022.3193008
Date of current version: 26 September 2022

architectural examples • Develop missing components • View reuse as separate." *[Editor's note: The lessons mentioned here are still valid today, but now, many tools help with adhering to them. It is worth reading as many lessons are explained in detail.]*

Implementing Reuse With RAD Tools' Native Objects; John C. Zubeck (p. 60) "Developers and project managers need to understand how to constrain project requirements to reap the reuse advantages that RAD (*Ed: Rapid application development)* tools offer. Designing exclusively within the capabilities of RAD Business Objects can stabilize projects, even enough to support larger applications. …The DCI/Droege Developers' Competition annually showcases the phenomenon of RAD development at its best." (p. 61) "If a new project does not respect the standard RAD-Business-Objects described in this table, it has failed to reuse the objects that have been prefabricated by the RAD tool manufacturer." (p. 64) "Programmers need clear explanations about how to code, set, and trigger all these ever more complex RAD-Business-Objects." *[Editor's note: The investigation centers on the experience gained by four RAD tool kits: Visual Basic, Lotus Notes, Magic, and PowerBuilder, which have all been a part of the competition. Many more such tools are now around and successfully used.]*

Automatically Identifying Reusable OO Legacy Code; Letha H. Etzkorn et al. (p. 66) "Much object-oriented code has been written without reuse in mind, making identification of useful components difficult. The Patricia system automatically identifies these components through understanding comments and identifiers." (p. 68) "The module of the Patricia system that handles program understanding and information extraction is called Chris (Conceptual Hierarchy for Reuse Including Semantics). In the case of comments, Chris first completely parses a sentence using a simple natural language parser, then uses its inference engine to semantically process the various parses. In the case of identifiers, Chris uses empirical information on common formats for variable and function identifiers to syntactically tag subkeywords." *[Editor's note: The article shows how the "semantic" of an OO piece of code can be detected. That would then enable a human to insert the code for reuse into a class library.]*

A Descriptor-Based Approach to OO Code Reuse; Ernesto Damiani (p. 73) "The COOR environment exploits the advantages of OO code to promote software reuse, performing classification and analysis using a Software Descriptor method based on a fuzzy query language for component retrieval. Fuzzy weighting mechanisms highlight relevant features of components for reuse in specific industrial application domains." (p. 74) "In our approach, a human expert, or application engineer, maintains the system for an audience of users, or application developers. COOR provides tools for the AE and AD to classify components and search for reuse candidates." (p. 78) "Code search for reuse consists of locating and selecting software components stored in the Object Base and represented in the Descriptor Base; by exploiting SD features and terms of the controlled vocabulary the ADs can investigate component behavior." *[Editor's note: With the help of the application engineer (AE) and application developers (ADs), objects are classified into a library, and Classification of Object-Oriented Code for Reuse (COOR) then provides a search-and-retrieval facility to find OO components for reuse.]*

Software Engineering Code of Ethics, Version 3.0; Don Gotterbarn et al. (p. 88) "By January of 1994, both societies (Ed: IEEE-CS, ACM) formed a joint steering committee "To establish the appropriate set(s) of standards for professional practice of Software Engineering upon which industrial decisions, professional certification, and educational curricula can be based." … To ensure, as much as possible, that this power will be used for good, software engineers must commit themselves to making the design and development of software a beneficial and respected profession. In accordance with that commitment, software engineers shall adhere to the following Code of Ethics. The Code contains eight Principles related to the behavior of and decisions made by professional software engineers, be they practitioners, educators, managers and supervisors, or policymakers, as well as trainees and students of the profession." *[Editor's note: This preliminary version, jointly developed by the IEEE Computer Society and the Association for Computing Machinery (ACM), was published for discussion in 1997 and adopted by IEEE CS and ACM in 1999 (https://www.computer.org/education/code-of-ethics). In 2018, ACM adopted a new version of the Code of Ethics (https://www.acm.org/code-of-ethics), and in 2020, IEEE adopted a much shorter code for all of its Societies (https://www.ieee.org/about/corporate/governance/p7-8.html). It is quite interesting to compare this preliminary version with the one from 1999, but also with the new formulation found at ACM. In my mind, the much shorter 2020 version from IEEE sufficiently specifies the ethical behavior of scientists, engineers, managers, educators, and even the general public. Unfortunately, all of them are only applied sparingly when money or power are at stake.]*

The Interplay of Art and Science in Software; Terry Bollinger (p. 128) "Rigorous science provides a frame-work and a solid basis for further analysis. It prevents you from following a hunch into untestable speculation or, worse, into superstitious reliance on factors that are provably irrelevant or that even oppose the hypothesis. The artistic part of this process lets science move unexpectedly into new cur-rents and previously unmapped under-standing. What is most striking about the biographies of truly great physicists such as Albert Einstein and Richard Feynman is how consistently their greatest theories stemmed from pursuing seemingly minor or even irrelevant issues. … . To a software developer who operates in the same way as a physicist, the goal is to collect the most powerful and carefully generalized set of software "theorems" (modules or subsystems) possible for use in constructing still more powerful theorems." *[Editor's note: This is a very interesting article that establishes the need for human ingenuity in some stages of the software development cycle. Methodology and models alone are not the answer.]*