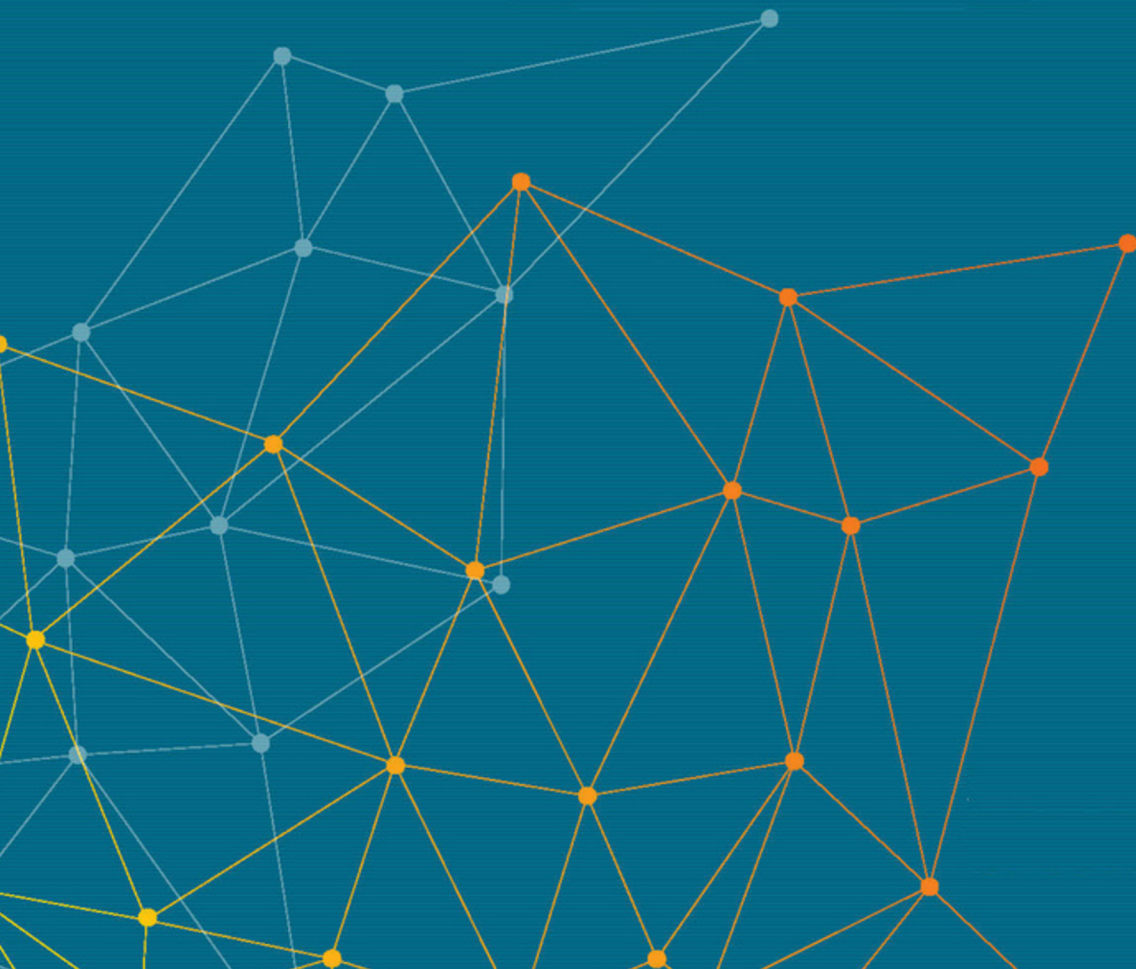




THE IMPACT OF CURIOSITY-DRIVEN INFORMATICS RESEARCH

An Engine of European Digital Sovereignty



The Impact of Curiosity-Driven Informatics Research

An Engine for European Digital Sovereignty

Authors

- Marco Aiello, University of Stuttgart (Germany)
- Lenuța Alboaie, Alexandru Ioan Cuza University (Romania)
- Jean-Marc Jézéquel, University of Rennes, CNRS, Inria, IRISA (France)
- Kim Mens, Université catholique de Louvain (Belgium)
- Simona Motogna, Babeș-Bolyai University (Romania)
- Ana C. R. Paiva, University of Porto (Portugal)
- Ina Schieferdecker, Technical University of Berlin (Germany)
- Petre Sora, University of Stuttgart (Germany)

April 2026

Published by:

Informatics Europe

Binzmühlestrasse 14/54

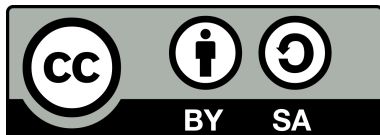
8050 Zurich, Switzerland

www.informatics-europe.org

communications@informatics-europe.org

DOI: 10.5281/zenodo.19659986

©Informatics Europe, 2026, CC BY-SA 4.0



This publication has been reviewed and approved by the Board of Informatics Europe.

Why Curiosity-Driven Informatics Research Demands Strategic Policy Support

A policy statement by the Board of Informatics Europe

Informatics serves as the central nervous system of the modern digital economy. Historically, Europe’s greatest digital contributions were not the result of rigid top-down directives, but of foundational curiosity-driven inquiry. However, as Generative AI and advanced computing usher in a transformative new era, the structural foundations that enable these paradigm-shifting discoveries are under severe strain. To secure Europe’s digital sovereignty and global competitiveness, policymakers must explicitly recognize software-based technologies of Informatics as Key Enabling Technologies (KETs) and urgently recalibrate the ecosystem that supports them.

The trajectory of technological progress is inextricably linked to the depth and autonomy of foundational scientific inquiry. Curiosity-driven research is sparked by an authentic desire to uncover truths and explore the unknown, rather than to achieve immediate utilitarian outcomes. In Informatics, this autonomy is critical. While mission-directed research, where external agencies define the questions, plays a vital role in application, it is curiosity-driven Informatics that empowers scientists to push the research frontier. This systematic investigation provides the intellectual substrate and serendipitous breakthroughs upon which all subsequent applied innovations are built.

Yet, current academic ecosystems inadvertently stifle this vital inquiry. Institutional incentives are increasingly plagued by “utilization” plans and “metric myopia”—a hyper-focus on research that should lead to short-term economic returns and on results that are quantitatively and simply measurable. This misaligned incentive structure inadvertently degrades research methodology, forcing researchers toward “safe,” incremental work and penalizing the bold, high-risk exploration necessary for foundational breakthroughs.

To safeguard Europe’s technological future, we call on European policymakers to implement structural reforms across four key pillars:

Securing Academic Sovereignty and Data Access: Safeguarding European scientific sovereignty requires legislating mandatory, zero-cost access to platform data for non-profit, independent researchers. Crucially, academics investigating digital markets must be shielded by robust anti-SLAPP (Strategic Lawsuits Against Public Participation) legal protections to preserve their ability to scrutinize powerful tech ecosystems without fear of corporate retaliation.

Elevating Informatics Education, including AI Literacy: Informatics must be recognized as a fundamental school subject from primary education onward. Integrating mandatory AI literacy and human-centric design will ensure the next generation can effectively orchestrate human-AI collaborations. This must be matched by a comprehensive European digital skills strategy, encompassing reskilling and upskilling initiatives for the current workforce.

Modernizing Research Evaluation and Funding: We must transition from volume-based evaluations to artifact-based responsible bibliometrics that properly recognize software, code,

and datasets as substantive scientific contributions. Furthermore, funding agencies must integrate agile, risk-tolerant mechanisms, such as high-risk/high-reward grants and randomized allocations for equally meritorious proposals, to bypass the conservative biases of traditional peer review.

Ensuring Expert Representation: To ensure that nuanced technical expertise informs digital regulation, the academic Informatics community must be granted permanent representation on high-level European advisory bodies.

Curiosity-driven Informatics is not a luxury; it is a strategic necessity. By investing in foundational research, supporting open-source software ecosystems, and enacting these structural reforms, Europe can transcend its reliance on foreign digital infrastructures. Now is the time for decisive policy action to ensure Europe remains not just a regulator of the digital future, but its primary architect.

In the rest of this document, we focus on the historical perspective, which provides the foundation for the present pledge to support curiosity-driven, not necessarily immediately useful research.

Contents

IE Policy Statement on Curiosity-Driven Informatics Research	1
Abstract	4
1 Introduction	5
I Infrastructure: The Foundation for the Rest of It	8
2 The World Wide Web: How a side project changed the way we access information and interact with each other	9
3 Linux: the Technological Backbone of Modern Society	10
4 Grid Computing and the European Architecture of Collaboration	11
5 The Architecture of Interoperability: European Innovations in Specification and Testing for Telecommunications	13
6 Amazon Web Services: How an online bookstore became the top cloud service provider	16
II Programming Languages: The European Side of Babel	18
7 SIMULA: Europe's Groundbreaking Contribution to Object-Oriented Programming	20
8 PROLOG: The European Roots of Logic Programming	20
9 Teaching the World to Program: The Pascal-Modula Revolution	22
10 Eiffel - Contract, Clarity and Code	23
11 Python	24
III Artificial Intelligence: The European Part of the Story	25
12 Deep Learning: From European Curiosity to a Global Technological Shift	26
13 The Transformer Revolution in Artificial Intelligence: From Recurrence to Global Attention	28
14 Conclusions	30
References	31
About the Authors	36

The Impact of Curiosity-Driven Informatics Research

An Engine for European Digital Sovereignty

Abstract. Once fundamental new ideas enter the scientific record, their societal impact can be inevitable; even if they emerge decades later, unexpected, and in unforeseen forms. The present essay demonstrates how European basic informatics research has repeatedly followed this trajectory, transforming “useless” or curiosity-driven work into technologies that now underpin modern digital society. Without the ambition to be exhaustive, through selected European examples, including the World Wide Web, Artificial Intelligence, Python, Linux, grid computing, and foundational research in distributed systems, the essay shows how long-term investment in fundamental ideas shaped programming paradigms, software engineering, open systems, and global digital infrastructures.

By tracing the origins of these developments and their eventual industrial and societal adoption, the essay highlights Europe’s distinctive contribution to global computing innovation. It argues that demands for short-term impact risk undermining the very conditions that make transformative breakthroughs unavoidable, and calls for sustained support of basic research as a strategic necessity for Europe’s long-term technological leadership.

1 Introduction

Marco Aiello

SCIENCE requires patience. It is not for the scientific process to set short-term goals and guarantee the impact of the results. Or, in other terms, narrowing research to processes with guaranteed results necessarily forces it to be insufficiently ambitious and to follow conventional, stagnant paths to success. As 2003 ACM Turing Award recipient Alan Kay stated, “if you don’t fail at least 90 percent of the time, you’re not aiming high enough.”

This is not to mean that bad, superficial research is desirable, but rather that research should be left free to go in unexplored corners of human knowledge and nature, to leave freedom to curiosity to drive it in perhaps implausible directions, and to let scientists be free from worrying about failure. This can be disturbing and unpopular with the general public, and with the people in charge of providing resources to scientists, may these be policymakers or company shareholders, but it might be the only way to increase the likelihood of scientific breakthroughs with societal impact in the long run.

This concept was presented perfectly by Abraham Flexner, the founding director of the Princeton Institute of Advanced Studies, the institution that welcomed Einstein and Gödel in the 1930s, in the essay *On the Usefulness of Useless Knowledge* published in 1939 [27]. In that treatise, he argues that it is nearly impossible to assess the impact of research at the time it is funded, since true impact emerges from contextual factors. He illustrates this point with the example of the radio, which was immensely popular at the time, claiming that Marconi’s contribution to its invention was negligible. He writes, “Marconi was inevitable. The real credit for everything that has been done in the field of wireless belongs, as far as such fundamental credit can be definitely assigned to anyone, to Professor Clerk Maxwell, who in 1865 carried out certain abstruse and remote calculations in the field of magnetism and electricity.” He later adds, “neither Maxwell nor Hertz had any concern about the utility of their work.” Examples like these abound. Would we have the GPS or the Galileo localization system without Einstein’s curiosity about the nature of time and his foundational contribution to the theory of relativity? Would we have secure online communication and digital transactions without the curiosity of mathematicians such as Pierre de Fermat, Carl Friedrich Gauss, and G. H. Hardy, who devoted their lives to number theory with no practical purpose in mind? Hardy famously wrote in 1940 that “nothing I have ever done is of the slightest practical use” [37]. Yet, the abstract study of prime numbers and modular arithmetic that fascinated these early scholars later became the foundation of public-key cryptography. In the 1970s, Whitfield Diffie, Martin Hellman, and Ron Rivest, with their colleagues, transformed this once esoteric field into the mathematical backbone of digital security, enabling encrypted communication, online banking, and the entire architecture of today’s Internet economy [42, 64]. Equally emblematic is the story of the laser. When Charles Townes and Arthur Schawlow developed the maser and its optical successor in the 1950s, many colleagues dismissed their work as a scientific curiosity without application, when not dismissing it as unlikely to work, as recalled by Townes himself in the Interview #124 held at the IEEE History Center on August 2nd of 1991. Yet, their pursuit of fundamental questions about the interaction between light and matter led to one of the most versatile technologies of

the twentieth century. From barcode scanners and optical fiber communication to precision surgery and quantum computing, the laser exemplifies how curiosity-driven exploration can unexpectedly reshape entire industries and ways of life.

The current trend in research funding is that a concrete utilization of the proposed research must be in sight. Even more strictly, funding applications now often require a dedicated section explaining how the exploitation of research results will be achieved. While for some endeavors this may be reasonable, in general, it stifles curiosity-driven research. It risks discouraging exploration motivated by curiosity rather than immediate application. Once again, Flexner's words should guide us: "I am pleading for the abolition of the word "use," and for the freeing of the human spirit. [...] To be sure, we shall thus waste some precious dollars." Money wasted today might be the best investment ever made when looked at it several decades later.

In this essay, we will examine examples of basic research that, over time, proved to be transformative. The potential for practical use was not the driving motivation. The analysis will be limited to the European context, although similar examples could easily be found in the United States, Japan, and other countries. Focusing on Europe allows us to highlight the diversity and strength of its research ecosystems, and to remain aligned with the perspective of Informatics Europe, the association that represents and connects European academic and research institutions in informatics. There is no intention of being comprehensive, but rather to capture significant examples that had a wide societal impact or were transformative in science and technology. Similarly, the boundary between informatics and another discipline is difficult to delineate. For example, cases like the CD or Bluetooth were omitted as considered borderline in terms of the informatics discipline.

We will begin with fundamental developments that shaped computer science and society at large, such as the creation of the World Wide Web, to then perform an overview of programming languages originating in Europe, the most widely known of which is the Python programming language, and finally to conclude with a look into the origins of Generative AI and related techniques.

The World Wide Web is a prominent case. It was created by Tim Berners-Lee in the late 1980s as a way to organize the vast amount of unstructured information available at CERN. It then gained attention from his peers and was considered a good way to share scientific data among physics labs across the world, and in a few years, it became pervasive and society-changing. A few years later, at the Centrum Wiskunde & Informatica (CWI) in the Netherlands, Guido van Rossum designed Python, a high-level programming language aimed at readability and simplicity. Initially a personal side project, Python has become a cornerstone of education, software engineering, and artificial intelligence research.

Other European contributions demonstrate similar trajectories. The development of the logic programming language Prolog in Marseille and Edinburgh in the early 1970s inspired decades of work on knowledge representation and reasoning, forming a conceptual bridge to today's AI systems. The Norwegian language Simula, developed at the Norwegian Computing Center in the 1960s, introduced object-oriented programming, influencing C++, Java, and much of modern software design.

The essay also considers the European role in artificial intelligence. The deep learning

revolution owes much to foundational work by Bengio, Hinton, LeCun, and Schölkopf, while newer entities such as DeepMind, Hugging Face, and Stability AI exemplify how academic ideas rapidly evolve into societal applications, from reinforcement learning to open-source diffusion models.

Together, these examples show how curiosity-driven research, sometimes regarded as “useless” at its inception, has repeatedly produced technologies that define modern life. In turn, a plea to free academic research from the constraint of exploitation and business-oriented thinking, and the financing of curiosity-driven research, no matter how *useless* it may appear to be.

Part I

Infrastructure: The Foundation for the Rest of It

Petre Sora

INFRASTRUCTURE is a fuzzy term. Infrastructure is the base for whatever comes “on top.” Some decades ago a single computer would have counted for certain applications as infrastructure enough. Today, work without email, which itself requires a good deal of infrastructure, is hard to imagine. Furthermore, specific domains presuppose specific, highly specialized applications. What is infrastructure then? When it comes to computers and computing everything from a single piece of hardware to a highly specialized software can be considered infrastructure. Especially today’s applications of computing assume the availability of a significant amount of it: computers, peripherals, operating systems, databases, data centers, and networks are part of the necessary minimum. Understood this broadly, Europe has some impressive, albeit in part little-known, landmarks to show with respect to the topic of infrastructure.

The history of computers and computing is, compared to (Continental) Europe, to a large extent dominated by the United States. This disparity applies, of course, to varying degrees to comparisons with all geographies. The causes are multiple. Historical reasons like the brain drain before and during the Second World War, the war’s consequences, differences in mentality between Europe and America, and the difficulties of political cooperation in Western Europe all played significant roles. It is still worth mentioning, though, that the first digital, programmable computer was finished in the face of many adversities in wartime Berlin in 1941 by Konrad Zuse [34, 85], something which is still insufficiently acknowledged. The authorities’ appraisal of the Z3, as the computer was called, seems not to have been very far-sighted, as Zuse recounts ([85], page 64): “Officially, however, the Z3 was not considered vital. It was considered, more or less, to be something for the amusement and private pleasure of my friends and myself.”

Interesting as the hunt for “firsts” may be, the several places computers (with different capabilities) were built during the same period suggests that the computer’s time as a technology had simply come. Just like Flexner’s earlier quote suggests about the invention of radio. As such, the following sections will not delve that far back into the history of computing, instead, this text will focus on several of the closer of the consequential technologies to today’s everyday life by starting with one of the most prominent European inventions of the last decades: the World Wide Web.

2 The World Wide Web: How a side project changed the way we access information and interact with each other

Marco Aiello

WORLD WIDE WEB is the name given by Tim Berners-Lee to his creation, which originated as a side project while at CERN. Frustrated with the difficulty of retrieving and organizing relevant work information, he decided to create a software system in which information would be represented as entities connected by qualified links. He recalls, “I wrote it in my spare time and for my personal use, and for no loftier reason than to help me remember the connections among the various people, computers, and projects at the lab” [12]. The system soon caught the attention of his colleagues, and Berners-Lee was encouraged to seek institutional support to further develop it. His director at the time, Mike Sendall, famously labeled the project proposal “vague but exciting,” and granted him the necessary resources to pursue it.

What began as a local information management tool at a European research laboratory quickly became a global phenomenon. Within a few years, the World Wide Web evolved from a set of interconnected hypertext documents into the backbone of modern communication and commerce. The release of the first popular browser, Mosaic, in 1993, and later the emergence of Netscape, Microsoft Internet Explorer, and open standards coordinated by the World Wide Web Consortium (W3C), transformed the Web into a ubiquitous platform. From the early academic pages of universities and research institutes, the Web expanded to host millions of websites, eventually reshaping how societies produce, distribute, and consume knowledge. In addition, profoundly changing commerce and even social interactions.

Technologically, the Web’s evolution reflects a continuous layering of innovation. The initial Web of information, a system for linking and reading documents, grew into a Web of software and business services, enabling interaction and transaction through protocols and languages such as HTTP, HTML, SOAP, and later REST. This, in turn, enabled e-commerce, online banking, and the emergence of digital economies. In the 2000s, the rise of social platforms and user-generated content led to what is often referred to as the social Web, which fundamentally altered media, education, and politics. Today, Web-based technologies underpin artificial intelligence, cloud computing, and the Internet of Things.

The Web’s societal impact is difficult to overstate. Recent studies estimate that over five billion people, that is, roughly two-thirds of the world’s population, use the Web daily. The economic value created by Web-based services exceeds trillions of euros annually, fueling industries ranging from education to entertainment, logistics, and finance. Yet, none of this was anticipated in Berners-Lee’s original proposal. His goal was not to start a company or create a market, but simply to make knowledge easier to navigate. His motivation was intellectual curiosity and the pursuit of a more elegant way to manage information, not a plan for commercial transformation. The Web’s extraordinary success thus stands as a compelling validation of Flexner’s argument that freeing research from immediate utility is the most powerful path to societal progress. Furthermore, one can even dig further historically into

the impact of basic research on the success of the Web. Similar to what Flexner claimed of Marconi regarding the radio (see Section 1), one could argue that Tim Berners-Lee was inevitable. The Web can be seen as the culmination of a lineage of visionary ideas: Vannevar Bush’s concept of the *Memex*, a theoretical device for associative information storage; Douglas Engelbart’s *onLine System*, which introduced hypertext and interactive computing; and Bill Atkinson’s *HyperCard* environment for the Macintosh [8]. Each of these contributed essential elements that made the Web conceivable, yet it was Berners-Lee’s curiosity, persistence, and belief in open source that turned them into a reality.

In hindsight, the World Wide Web stands as one of the clearest examples of how basic, curiosity-driven research and creative experimentation, pursued without commercial intent, can profoundly reshape the world [68].

3 Linux: the Technological Backbone of Modern Society

Jean-Marc Jézéquel

LINUX is nowadays a critical layer of the technological substrate on which modern society depends.

That contrasts with its very modest origin, which can be traced back to 1991, when Linus Torvalds, then an undergraduate in computer science at the University of Helsinki, began developing a small, UNIX-like kernel to better understand operating-system internals and to create a more capable system for his personal computer. At the time, Torvalds was studying UNIX design principles through Prof. Tanenbaum’s book [73] “Operating Systems: Design and Implementation” (Vrije Universiteit Amsterdam) and using MINIX, a pedagogical UNIX clone. Although MINIX served instructional purposes, its limited functionality and restrictive licensing made it unsuitable for broader experimentation. These constraints motivated Torvalds to design an independent, freely modifiable kernel from scratch for the Intel 80386 architecture.

On 25 August 1991, Torvalds publicly announced his project on the `comp.os.minix` newsgroup, describing it modestly as a personal undertaking. The early release included basic process management, memory handling, and a simple file system. Crucially, Torvalds developed Linux to interoperate with the widely available GNU software ecosystem—including the GNU Compiler Collection (GCC), core utilities, and libraries—thereby enabling the construction of complete, functional UNIX-like systems from freely available components. In 1992, Torvalds relicensed the kernel under the GNU General Public License (GPLv2), ensuring that all derivative works would remain open and fostering a rapidly expanding collaborative development community.

Throughout the 1990s, Linux evolved from an experimental kernel into a robust and portable operating system supported by thousands of volunteer developers worldwide. The emergence of early distributions—such as Slackware, Debian, and Red Hat—formalized system integration and package management, making Linux increasingly accessible for research,

education, and commercial deployment. Its architectural clarity, permissive development model, and the accelerating maturity of surrounding open-source utilities positioned Linux as a powerful and flexible alternative to proprietary UNIX systems [56].

Linux rapidly became a widely adopted research platform due to its transparency, modifiability, and cost-free availability. It enabled researchers to experiment with kernel subsystems, networking stacks, distributed systems, filesystems, real-time extensions, and high-performance computing environments without the licensing barriers typical of commercial UNIX systems. As a result, Linux fostered a generation of academic work on systems design, security models, scheduling algorithms, virtualization, and cluster architectures. Its presence in research laboratories worldwide accelerated reproducibility and lowered the barrier to systems experimentation.

Linux has evolved into the most pervasive operating system in modern computing infrastructure. It powers the majority of web servers, provides the foundation for the Android mobile ecosystem, dominates supercomputing, and serves as the core of countless embedded and industrial systems, from telecommunications equipment to automotive control units. Its stability, security, and open-development model have made it indispensable to cloud services, scientific computing, global finance, and consumer technology. As a result, Linux has become a structural element of contemporary digital society, enabling innovation at a global scale while embodying the long-term societal value of open, collaborative scientific development.

4 Grid Computing and the European Architecture of Collaboration

Lenuta Alboaie

GRID computing emerged in the late 1990s as scientific research began to require more computational capacity than any single center could provide. Early architectural work outlined how distributed resources could be linked through common protocols and unified security mechanisms, forming a coordinated computational environment across dispersed infrastructures [29, 30]. The conceptual model emerged internationally and reflected a shift from centralized computing towards distributed and federated infrastructures.

Europe was one of the principal proving grounds where these concepts were translated from theoretical models into sustained scientific practice. By the early 2000s, European research communities relied on cross-border collaboration, particularly in particle physics, climate research, materials science, and computational chemistry. CERN in Geneva played a central role in this evolution. The experimental program of the Large Hadron Collider generated unprecedented data volumes that required continuous analysis distributed across several countries. This necessity led to the development of the Worldwide LHC Computing Grid, widely documented as the first large-scale and long-term scientific Grid infrastructure [13, 32]. The system linked Tier-1 centers such as INFN-CNAF in Bologna, CC-IN2P3 in Lyon, GridKA at the Karlsruhe Institute of Technology, the Rutherford Appleton Laboratory in the UK, and DESY in Hamburg, together with an extended network of European

universities and laboratories [17]. These deployments demonstrated that distributed systems could operate as a coherent scientific platform and support sustained research involving many independent organizations. European collaborations also shaped the organizational structures that allowed such infrastructures to function effectively. Concepts articulated in early Grid literature, including the Virtual Organization, became operational mechanisms through projects such as the DataGrid initiative and the EGEE (Exploiting Grids for eScience in Europe) program. These efforts showed how federated identity management, coordinated resource access, and shared governance could support large-scale cooperation across institutional and national boundaries [71, 14]. The Worldwide LHC Computing Grid incorporated these principles into everyday scientific workflows and provided one of the most advanced examples of a Virtual Organization functioning in practice. Projects such as European DataGrid and EGEE developed middleware frameworks, including gLite, while European Grid Infrastructure also integrated technologies such as UNICORE [47, 72]. These systems provided authentication, workload scheduling, monitoring, and distributed data management and complemented earlier Grid toolkits by offering stable solutions for long-term scientific operations. Their development involved coordinated contributions from many European laboratories and universities and helped establish a consistent approach to interoperability across scientific infrastructures in Europe.

The experiences and innovations from grid computing in the 2000s fed directly into the rise of cloud computing in the 2010s. Many architectural principles that became central to modern cloud platforms were first tested and refined at grid scale: virtualization, distributed job scheduling, workflow automation or data locality [31]. CERN contributed further to this continuity by adopting and developing large-scale open source cloud platforms and by collaborating with industry partners in initiatives reported in studies of scientific cloud deployment [49, 83]. These developments showed how operational practices established in Grid environments informed the design and operation of modern cloud ecosystems.

Building on this technical continuity, European initiatives on cloud governance later advanced similar concerns about openness, accountability, and interoperability. Policies such as the General Data Protection Regulation, projects including GAIA-X or the European Open Science Cloud (EOSC) reflect a broader European orientation toward transparent data management and federated digital infrastructures, extending principles pioneered in large-scale scientific Grids [28, 25].

In this broader perspective, Grid computing holds an important place in the history of distributed systems. Emerging within an international scientific context, it reached operational maturity in Europe, where long-term collaborations demanded coordinated computational structures. Its influence extended beyond science: governance models, federated access mechanisms, and secure data-sharing practices developed in European Grid projects anticipated approaches now used in healthcare, environmental monitoring, public digital services, and industrial data platforms. These patterns continue to shape Europe's vision for responsible, interoperable digital ecosystems. Without the technical and collaborative models established in Grid environments, cloud computing would not have taken its present form [9].

5 The Architecture of Interoperability: European Innovations in Specification and Testing for Telecommunications

Ina Schieferdecker

WE have all quickly become accustomed to using telecommunication and mobile devices on a daily basis. The communication networks underlying these devices are so complex that we often forget the incredible engineering feat they enable: connecting any device to any network anywhere in the world, regardless of the manufacturer or operator, and securely and reliably transporting any type of data, including text, documents, voice, audio, pictures and video. These networks support services such as messaging, signalling, emergency calls, mobile banking, remote surgery and global internet access, to name but a few. All these services are underpinned by communication protocols developed through protocol engineering to enable reliable, secure and interoperable communication. From the earliest days of mobile telephony, the vision has been of global connectivity and interoperability.

The creation of the Short Message Service (SMS) is one of the most famous examples of curiosity-driven research in protocol engineering. In 1984, the Groupe Spécial Mobile (GSM) approached two men to ask if it would be possible to send text messages using mobile networks. SMS, which today sees roughly 23 billion messages sent daily, originated from Friedhelm Hillebrand’s curiosity about human communication patterns. This eventually led to the 160-character protocol constraint, designed to fit existing signalling capacities. This innovation revolutionised personal communication and paved the way for future messaging services worldwide. Previously, only a few researchers had envisaged the potential of such a service [38].

Less well known is the fact that the quest for global interoperability in mobile communications was also driven by curiosity-driven research in Europe. To enable Nokia phones to communicate with Ericsson base stations, pioneering protocol engineers such as Ivar Jacobson—who would later co-invent the Unified Modeling Language—developed graphical representations of complex protocol behaviour as ‘state machines’ to prevent logic errors in switching systems [41]. It laid down basic concepts for a mathematical language for protocol engineers to describe communication protocols without ambiguity.

Later on, many of the further foundational technologies and standards enabling global interoperability were developed through European initiatives. Indeed, the transition from 1G analogue systems to 2G digital mobile telecommunications involved more than just a change in radio frequencies and modulation techniques. While 1G was primarily a hardware-centric public voice service, 2G introduced the need for encrypted communication, error detection and authorisation of channels to be shared by a large number of users simultaneously [6]. In response to this complexity, European leadership, channelled through the European Telecommunications Standards Institute (ETSI), pioneered a shift from a ‘hardware-first’ to a “*specification-first*” *engineering paradigm*. Supported by European Commission research frameworks, such as the European Strategic Program for Research in Information Technology (ESPRIT) initiative, this approach ensured interoperability was built into the

standards via formal technical, syntactical, and semantic validation, as a preliminary step before physical systems deployment [75]. Modern mobile networks (2G–5G) owe their global success to this Europe-led informatics framework, which formed the backbone of the mobile revolution.

A cornerstone of this specification-first approach was the development of the Abstract Syntax Notation One (ASN.1), which provides a vendor-, platform-, and language- independent notation for specifying data structures. Developed in the 1980s by the International Organization for Standardization (ISO) and the International Telecommunication Union—Telecommunication Standardization Sector (ITU-T), with significant input from Europe, ASN.1 allowed designers to focus on application semantics rather than the underlying electrical or optical signals used to represent bits. It became the ‘lingua franca’ for global telecommunications, forming the basis of X.400 messaging, Signaling System No. 7 and the radio access networks of 3G and subsequent systems. To address the extreme bandwidth constraints of early cellular networks, European engineers developed the Packed Encoding Rules (PER). Unlike the more verbose Basic Encoding Rules (BER), PER reduces the number of bits by up to 50% by removing redundant octets and utilising PER-visible constraints to statically determine field lengths, thus demonstrating the direct translation of algorithmic efficiency in informatics to industrial scalability [46, 6].

While ASN.1 formalised complex dynamic data structures used in communication messages, the complexity of digital switching necessitated a concurrent shift from ambiguous natural language descriptions to formal, executable models of behaviour as prepared by Ivar Jacobson’s Phd thesis. The Specification and Description Language (SDL) was developed by the ITU-T with significant contributions from Europe to model reactive, concurrent, and distributed systems. Much of SDL’s formal power was refined through the Research and Development in Advanced Communications Technologies in Europe (RACE) programme. Specifically, the Specification and Programming Environment for Communication Software (SPECS) project sought to provide a rigorous formal semantics for telecommunications software. Its primary achievement was establishing a rigorous formal semantics for specification languages such as SDL, thereby transforming them from simple diagrams into mathematically verifiable models. By developing a Common Representation (CR) for heterogeneous systems and tools for automated analysis, the SPECS project laid the essential groundwork for specification-based protocol engineering and testing, as well as for the generation of reliable code in modern digital networks [16].

Based on its theoretical foundation of communicating extended finite state machines, SDL allowed European telecom giants like Ericsson and Nokia to describe control behaviour in telecommunication clearly and realistically. This legacy has enabled engineers to simulate and verify the implementation-independent specifications of mobile communication protocols, drastically reducing time-to-market and increasing the reliability of the global switching infrastructure.

However, formalising the specification was only half the battle. Ensuring that implementations from various vendors matched those specifications required a new gold standard of validation. Testing methodology has evolved from the early Tree and Tabular Combined Notation (TTCN) used in 2G, to the modern Testing and Test Control Notation (TTCN-3), which has been under development at ETSI since 1999 and has been updated and ex-

tended annually ever since. TTCN-3 is a unique, specialised test design and programming language—like ASN.1 or SDL a domain-specific language (DSL) for telecommunications—designed specifically for the black-box testing of distributed, communicating systems. Its development was greatly influenced by European informatics research at institutions such as universities in Twente, Budapest, Berlin and Göttingen [36].

Today, TTCN-3 is a critical tool for conformance and interoperability testing in mobile communications. It ensures that handsets from one manufacturer work with base stations from another, guaranteeing interoperable, secure and reliable communication. This rigorous validation process remains a prerequisite for the global interoperability and roaming success of 3rd Generation Partnership Project (3GPP) standards, including protocols from 2G to 6G and beyond [75].

The rigours of telecommunications specifications eventually provided the ultimate stress that somehow forced software engineering and general software engineering to mature. This profoundly influenced the development of the Unified Modelling Language (UML), which was co-developed by Ivar Jacobson. UML is a global industry standard for designing and graphically modelling software systems. Overseen by the Object Management Group (OMG), it provides a standardised set of notations for describing, visualising and documenting the architecture, data and behaviour of complex software systems. It uses various diagram types, such as class, sequence and state machine diagrams, to represent the static structure and dynamic behaviour of systems.

Although UML began primarily as a general-purpose design tool, the evolution of UML 2.0 was heavily influenced by the formal rigour of telecommunications engineering. This allowed UML to support more precise semantics and specialised extensions, such as the UML Testing Profile (UTP).

With the UTP, European research enabled the rise of model-driven testing for general software systems by exporting concepts matured in TTCN-3, such as formal test architectures and verdicts, and extending them to include arbiters, for example, and bringing them to the broader IT world. This enables the wider IT sector to benefit from the same ‘telecom-grade’ robustness and reliability that was originally developed for cellular networks [11, 67].

As the industry moves towards 6G, these European innovations are evolving to handle network virtualisation, network slicing, and AI-driven management [6]. While other regions often dominate the global application layer, the ‘integrity layer’ of the digital world—comprising model-driven protocol, software engineering and testing to ensure global connectivity and interoperability—remains a profoundly European achievement. And so, many years of curiosity-driven basic Informatics research, supported by ongoing European funding, have made mobile communication one of the most reliable and interoperable systems in human history.

6 Amazon Web Services: How an online bookstore became the top cloud service provider

Marco Aiello

AMAZON WEB SERVICES (AWS) represents one of the most profound technological and economic transformations of the early twenty-first century. While AWS is today a core component of an American multinational corporation, its intellectual roots are deeply European. The conceptual foundations of large-scale distributed computing on which AWS is built can be traced back to the academic work of Werner Vogels, who was educated at the Vrije Universiteit Amsterdam. Under the supervision of Andrew S. Tanenbaum, whose operating systems research led to MINIX and indirectly to Linux (see Section 3). Vogels explored the challenges of scalability, fault tolerance, and consistency in distributed systems, long before the notion of “cloud computing” existed.

After completing his Ph.D. in Amsterdam, Vogels continued his academic career in Portugal, where he collaborated with research groups studying reliable distributed object systems. This experience further deepened his understanding of replication and fault recovery in heterogeneous networked environments.

In the 1990s, distributed systems were largely the domain of academic inquiry. Researchers examined replication, distributed consensus, and failure recovery as abstract problems of coordination, often with little expectation of immediate industrial use. Vogels’ doctoral work and subsequent research focused on the reliable operation of networked services and the trade-offs between availability and consistency. These investigations anticipated the very challenges that would later define cloud computing: how to build reliable systems out of unreliable components. In retrospect, this line of research provided the conceptual blueprint for what would become one of the most powerful infrastructures in modern computing.

When Vogels joined Amazon in the early 2000s, the company was primarily an online retailer struggling to scale its IT systems. Drawing on his academic understanding of distributed architectures, he helped transform Amazon’s internal infrastructure into a set of modular, reusable services. The insight that these services could be externalized for other organizations to use, marked the birth of Amazon Web Services. The launch of Simple Storage Service (S3) and Elastic Compute Cloud (EC2) in 2006 introduced a new paradigm: computation and data storage as elastic, on-demand resources. Vogels played a central role in defining this architecture and later became Amazon’s Chief Technology Officer, shaping its overall technological vision and external research engagement strategy.

A key milestone in this transformation was the publication of the Dynamo paper in 2007, which described the distributed key-value storage system used internally at Amazon [20]. Co-authored by Vogels, it became one of the most influential papers in systems engineering, inspiring the design of NoSQL databases and large-scale fault-tolerant storage systems across industry and academia.

The societal and economic consequences have been immense. AWS today powers much of the Internet’s backend, providing infrastructure for governments, startups, and multinational corporations alike. It has enabled entirely new industries, lowered the barriers to

technological innovation, and contributed decisively to the global shift toward data-driven business models. Remarkably, Amazon now earns a substantially greater share of its profit from cloud services than from retail operations. Although AWS represents only about 15% of Amazon's total net sales, it consistently generates more than 60% of the company's operating income, underscoring the transformative economic impact of a technology that originated from research in distributed systems.

Yet, as with the Web, none of this was foreseen when Vogels and his colleagues were studying the reliability of distributed systems in Amsterdam. Their work was motivated by intellectual curiosity about coordination and consistency, not by the ambition to reshape global computing. In hindsight, the rise of AWS demonstrates how Europe's deep academic contributions to computer science, particularly in systems design and architecture, can generate lasting global transformations.

Part II

Programming Languages: The European Side of Babel

Petre Sora

THE *Online Historical Encyclopaedia of Programming Languages* (HOPL) contains, by its own account, information on close to 9000 programming languages [61]. In contrast, *Ethnologue*, an authoritative database on worldwide living natural languages, identifies 7170 languages as being currently alive [24]. One might wonder: Why are there so many programming languages and what is their use? Of course, the above comparison is made for effect and is not quite fair as it stands. It is important to point out that HOPL’s definition of what constitutes a programming language is obviously broader than it is usually the case, as it contains “programming languages” from the eighteenth century onwards (not many though). Furthermore, one must be aware of the fact, that many programming languages are arguably rather dialects (this is, of course, also a question which concerns natural languages), and most of the nearly 9000 languages cannot be considered “alive,” many of them were never even used on a significant scale. But even so, the question remains.

The proliferation of programming languages started soon after computers became available and is indeed considerable. Already in 1961 did the January issue of the *Communications of the ACM*, an important trade journal with a wide reach, figure on its cover an image of a tower of Babel made out of bricks inscribed with names of programming languages; the image of “Babel” has since accompanied discussions on programming languages (see [57, 66]). The reasons for the proliferation of programming languages are multiple and vary with the passage of time, as the multiplication incessantly continues. Among these reasons are the comparative ease of creating a new language and the comparative ease of being able to publish about it. Economic competition and differentiation also play a role. There are also “good” reasons though for the creation of new languages, which point to a certain necessity for this proliferation.

Definitions of what a programming language is, usually stress their tool-character, their communicational aspects, and their expressive or descriptive powers (see, e.g., [7, 40]). Programming languages are made to solve problems: the better a problem can be described and handled, the easier and safer it can be solved. Of course, not all problems are alike, different *domains* call for different approaches. Programming is not only dependent on the domain, but is also a technical activity influenced by scientific progress: New concepts are sometimes integrated in existing languages, but it is often easier to test their viability within new languages. The central concept of a programming language, or its general approach defines the so-called *programming paradigm* the language belongs to (many older and most modern languages support multiple paradigms). The most important programming paradigms are:

- Procedural: Programs are structured as *procedures* which act upon data.

- Object-Oriented: Programs consist of *objects*, which interact with each other.
- Functional: Programs are structured as compositions of *functions*.
- Logic: Programs consist of the application of *formal logic* to sets of facts.

While the European role with respect to the topics of infrastructure or artificial intelligence is secondary to the American one, with respect to contributions on programming languages the United States and Europe are arguably on par. Little known is the fact that the first high-level programming language originated in Germany, just like the first computer a creation of Konrad Zuse's he called *Plankalkül* (Plan Calculus). Wolfgang Giloi points out admiringly in the abstract of his historical piece on this early programming language that “Plankalkül was not only the first high-level programming language but in some aspects conceptually ahead of the high-level languages that evolved a decade later” [34]. ALGOL 60, short for ALGOritmic Language, a joint effort of American and European computer scientists with very important contributions from the latter, proved to be of lasting importance, as many of the most widely used of today's programming languages are its descendants. Simula 67, a highly original Norwegian effort from 1967 based on ALGOL, is acknowledged to be the first object-oriented programming language. The list could easily go on as further highly influential European languages like Prolog, Pascal, and Python among others were developed during the ensuing decades.

One might wonder about the reasons for these preeminent and sustained European accomplishments. This success is largely attributable to exceptionally talented individuals (several of them were later honored with the Turing Award) and the institutions these people were involved with. Some important differences in mentality and culture with respect to the United States might have also played a decisive role, though. There is a certain noteworthy European focus on simple solutions—Niklaus Wirth, one of the most versatile computer scientists and a highly prolific programming language developer, stands as a symbol for this tradition [15]. There is also a less stronger focus on direct return and immediate applicability in the European mentality, e.g., it has been noted (e.g., [45]) that there is a difference between the so-called Scandinavian school of object-orientation, which emphasized the design and modeling aspects of the technology (design and modeling are essentially a form of understanding), and the American one, which highlights the reuse aspects of object-oriented programming (an economic trait)—see for a longer discussion ([60], p. 259–ff.). These European features have often not led to an immediate success with respect to programming languages, but the success was a lasting one nonetheless. The following sections aim to shed more light on these important European contributions to the field of programming languages starting with the Scandinavian language Simula.

7 SIMULA: Europe’s Groundbreaking Contribution to Object-Oriented Programming

Kim Mens

SIMULA marked one of the most influential milestones in the history of contemporary programming languages. Developed in the 1960s by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center in Oslo, Simula was originally intended to model complex systems through simulation. In doing so, it introduced a set of revolutionary programming concepts that would later define the object-oriented programming (OOP) paradigm.

Syntactically, Simula was based on ALGOL 60. Simula I was developed around 1962, and its more influential successor, Simula 67, was finalized in 1967. Whereas Simula I was primarily designed for simulation, Simula 67 was designed to be a general-purpose programming language and provided the framework for many of the features of object-oriented languages today [81].

Simula was originally designed for discrete event simulation, which is why the object-oriented model was so fitting: it allowed software to be structured around real-world entities with state and behaviour. More specifically, Simula 67 was the first language to introduce classes, objects, inheritance, and dynamic (or late) binding. These concepts laid the foundation for later object-oriented languages such as C++ (developed by Danish computer scientist Bjarne Stroustrup at Bell Labs), Smalltalk, and Java.

Although Simula was not widely adopted as a general-purpose language, its conceptual impact was profound. Bjarne Stroustrup, creator of C++, Alan Kay, the principal creator of Smalltalk, and James Gosling, creator of Java, have all acknowledged Simula as a direct and major source of inspiration for their work. Simula transformed how software systems could be designed, making them more modular, reusable, and maintainable. In recognition of their pioneering contributions, Dahl and Nygaard were awarded the ACM Turing Award in 2001, “*for ideas fundamental to the emergence of object-oriented programming, through their design of the programming languages Simula I and Simula 67*” [5].

Simula remains a shining example of how European research, particularly from Scandinavia, has shaped the global trajectory of programming language design. Its legacy continues to influence contemporary programming languages, software engineering practices, and computer science education worldwide.

8 PROLOG: The European Roots of Logic Programming

Kim Mens

PROLOG is a pioneering logic programming language that emerged from European research in the early 1970s. Developed by Alain Colmerauer and Philippe Roussel at the University

of Aix-Marseille, Prolog was initially conceived as a tool mainly for natural language processing [18]. Roussel coined the name “Prolog” (as an abbreviation for “PROgrammation en LOGique”) in 1972, and together with Colmerauer, implemented the first Prolog interpreter that same year [80].

The logical foundations of Prolog trace back to the resolution principle introduced by John Alan Robinson in 1965, which provided a complete inference mechanism for first-order predicate logic. Robinson’s resolution method unified two key ideas: Herbrand’s theorem—developed by Jacques Herbrand in the 1930s during his time in Paris, and unification, a process of making logical expressions identical by finding suitable substitutions for variables. Robinson, who had studied in Paris and was influenced by Herbrand’s work, formulated resolution as a general proof procedure for automated theorem proving. This mechanism became the logical engine underlying Prolog, created in the early 1970s by Alain Colmerauer and Philippe Roussel in Marseille, and Robert Kowalski in Edinburgh.

To make resolution-based reasoning computationally viable, an efficient method for unification was essential. The first linear-time unification algorithm was presented by Alberto Martelli and Ugo Montanari from Pisa in their seminal paper “An Efficient Unification Algorithm” (1982) [51].

This Franco-British collaboration with later Italian additions laid the groundwork for logic programming as a distinct paradigm, emphasizing declarative problem-solving through facts and rules rather than imperative instructions.

After the invention of Prolog in Marseille, the next major European contribution was compiling Prolog, which really put Prolog on the map. The first Prolog compiler was developed by David H. D. Warren at the University of Edinburgh in 1975. Edinburgh Prolog defined the syntax and basic primitives that influenced all subsequent Prolog systems. David Warren went on to develop the Warren Abstract Machine in 1983, which became the basis for most efficient Prolog implementations later.

Prolog quickly became central to early artificial intelligence research. Its ability to represent knowledge and perform automated reasoning made it ideal for expert systems, theorem proving, and natural language understanding. The language later played a key role in Japan’s ambitious Fifth Generation Computer Systems project (FGCS) in the 1980s, which aimed to build intelligent machines using logic programming and massively parallel architectures [78]. The FGCS project failed for a variety of reasons including hardware limitations, overambitious goals, and competition from other paradigms like object-oriented programming and neural networks. Another technical reason for its failure was its inability to unify search-based and concurrent Prolog. This synthesis of the search-based and concurrent variants of Prolog was another European advance achieved by Sverker Janson at the Swedish Institute of Computer Science in 1994.

European contributions to Prolog’s evolution continued through various implementations and dialects, including those from universities in Belgium, France, Germany, Hungary, the Netherlands, Poland, Portugal, Spain, Sweden, and United Kingdom [70]. The ISO standardisation of Prolog in the 1990s further solidified its place in the global computing landscape [44].

While Prolog is no longer central in mainstream AI, its legacy endures. Especially in symbolic AI, constraint logic programming, and explainable systems, it offers a high level

of abstraction for reasoning tasks. The recent resurgence of interest in interpretable and rule-based AI has renewed appreciation for the foundational work of Colmerauer, Roussel, Kowalski, Warren and their European peers.

Notable enduring European implementations of Prolog still widely used today include SWI-Prolog (The Netherlands), widely used for research and web applications; SICStus Prolog (Sweden), known for its performance and industrial use; and CIAO Prolog (Spain), which emphasizes modularity and integration with modern programming paradigms.

9 Teaching the World to Program: The Pascal-Modula Revolution

Simona Motogna

PASCAL was the first programming language I learned, and so was for lots of other programming enthusiasts. Pascal introduced a clean, structured, and strongly typed approach to programming at a time when programming languages were low-level, with a lot of syntactical constraints and source code was difficult to comprehend. Wirth's philosophy *Algorithm + Data Structures = Programs* [82] established the foundational principles that the efficiency and fineness of an algorithm implementation depend on the employed data structures and that the structure and simplicity lead to more reliable and maintainable programs. Modula and Modula 2 were built following Pascal's distinctive features and extended them with the concept of module, information hiding, and co-routines dedicated to concurrent processes [79].

Developed by Niklaus Wirth and his team at ETH Zürich, Pascal was greatly influenced (similarly to Simula) by ALGOL 60, and then ALGOL W (created by Nicholas Wirth and C.A.R. Hoare). Important contributions were made in terms of compiler construction (P-code [58]), programming and testing of early versions for Pascal and Modula.

Pascal rapidly became one of the most influential teaching languages in computer science, shaping the education of millions of programmers during the 1970s–1990s. Its simplicity and structure eased the introduction of programming in high schools. The design of Pascal has been the starting point in proposing other languages such as Ada, Modula-3, Oberon (also by Wirth), and also influencing design decisions in Java and C#.

Modula-2 provided an early example of language-level modules, influencing research in encapsulation, modularity, and concurrent systems.

In a broader way, in the 1980s Pascal standardized the teaching of programming, algorithm design, and data structures, and enforced the structured programming principles. Programming grew from a complicated and tedious task to an accessible and easy-to-understand activity, greatly contributing to professionalizing software development.

10 Eiffel - Contract, Clarity and Code

Simona Motogna

EIFFEL is more than a programming language proposal, but an approach to build software that is reliable, maintainable, and reusable by introducing the concept of *Design by Contract (DbC)* [53]. Meyer developed Eiffel with the goal of unifying all phases of software development, analysis, design, implementation, testing, and maintenance within one coherent framework. This stands in contrast to existing fragmented workflows that rely on separate tools for each stage.

Eiffel was developed under Bertrand Meyer’s leadership at Interactive Software Engineering (ISE), California. However, Meyer’s academic affiliation with ETH Zürich served as a hub for research, teaching, and formalization of Eiffel concepts, compilers, frameworks, and libraries.

There were three major drivers behind Eiffel’s creation: in the early 1980s, the software systems were becoming more and more complex, object-oriented principles were already looking promising and had been adopted by researchers and practitioners, and Meyer’s interest in software modularity, correctness, and reusability.

In a short time, Eiffel and the Design by Contract principle have been adopted and influenced in a significant manner in different computer science research topics, including object-oriented language design, such as Java assertion mechanisms, .NET Code Contracts, or decorator-based contracts in Python. Design by Contract became a foundational concept studied in programming language theory and static analysis. Software engineering methodologies have been notably transformed [52], with major impact on curricula and textbooks, but also reshaping industry practices.

Overall, Eiffel’s impact is broader than its user base: it contributed a vocabulary and methodology that increased the rigor and maturity of the software engineering discipline. Design by Contract profoundly reshaped industry practices by influencing testing frameworks, API design, and reliability engineering. Many modern software development tools, including unit testing, contract checking, and interface specification systems, draw clear inspiration from Meyer’s principles. Although Eiffel is not a mainstream language, it continues to serve in specific domains such as finance and avionics, where correctness and robustness are essential. Beyond the language itself, Meyer’s influential textbooks, especially *Object-Oriented Software Construction* [52], have educated generations of developers. Through this work, he helped professionalize object-oriented software engineering and established rigorous methods that continue to guide software development.

11 Python

Ana C. R. Paiva

PYTHON is a programming language that incorporates significant scientific contributions, placing great emphasis on readability, simplicity, and ease of use [1]. It features a clean syntax that enables the development of clear and easy-to-maintain code, which has significantly influenced software development practices. Python’s design is based on a philosophy known as “The Zen of Python” which emphasizes clarity and elegance [59]. This philosophy contributes to making Python a versatile tool in various fields, including web development, data analysis, machine learning, and scientific computing.

Python was created by Guido van Rossum in the late 1980s and early 1990s while he was working at the Centrum Wiskunde & Informatica (CWI) in the Netherlands. This was also the institution where ABC, a teaching and prototyping language, that greatly influenced the creation of Python, had previously been developed. Although ABC was innovative for its time, it faced distribution challenges and had limited success [4].

The creation of Python was motivated by the need for a language that could serve as an effective scripting tool while also supporting rapid development. With this need in mind, Van Rossum devised a language that could bridge the gap between low-level programming and high-level scripting tasks. This characteristic makes Python suitable for a wide range of users and applications.

Python has had a significant impact on the scientific community because its rich ecosystem of libraries and frameworks enables the implementation of complex algorithms, data analysis, and prototyping without requiring in-depth knowledge of computer science. This accessibility has fostered interdisciplinary collaborations and accelerated innovation in areas such as computer science, bioinformatics, and machine learning.

Beyond its impact on research, Python has also had a strong influence on society. It is a widely popular programming language, ranked among the top ten most popular languages by the TIOBE Programming Community Index since 2003 [2], and used by both educators and professionals in various fields. Furthermore, the establishment of the Python Software Foundation (PSF) has helped formalize its development and governance within the community, fostering an inclusive culture that welcomes contributions from different backgrounds [3]. It has also played a significant role in making programming education more accessible, empowering people to learn programming and develop software in an approachable and effective way.

Part III

Artificial Intelligence: The European Part of the Story

Petre Sora

IN a much-noticed article by the community of historians of computing, the historian Michael S. Mahoney cautions that “computers and computing [...] have always been surrounded by hype [...], and hype hides history” ([50], page 120). Salient examples of hyped technologies discussed so far in this essay are the World Wide Web and object-oriented programming. This last part of the essay is dedicated to what is probably the currently most hyped sub-field of computing: artificial intelligence (AI). The grandiose and often-cited optimism of AI pioneers like Herbert Simon and Marvin Minsky during the sixties or the high aims and promises of the the Japanese fifth-generation computing project during the eighties (for more information see [26], see also Section 8) makes the approach of “[h]istorians [who] prefer to judge revolutions in retrospect, after the dust has settled” ([50], page 120) appear wise. Nonetheless, although both Simon’s prediction from 1965 that “machines will be capable, within twenty years, of doing any work that a man can do” (Simon, as quoted in [23], page 129) and Minsky’s from 1967 that

within a generation ... few compartments of intellect will remain outside the machine’s realm—the problem of creating “artificial intelligence” will be substantially solved (Minsky, as quoted in [23], page 146)

remain unfulfilled to this day, the progress made during these last six to seven decades in the field of AI and the degree to which this progress has permeated everyday matters, is both fascinating and worrisome.

The “birthplace” of AI is often considered to be the *Dartmouth Summer Research Project on Artificial Intelligence*, a summer workshop held at Dartmouth College in 1956 which counted as attendants, among others, John McCarthy, Minsky and Simon. Seeing the Dartmouth summer workshop as the “birthplace” of AI is somewhat of a misleading simplification though. Claude Shannon, probably best known for his paramount contributions to information theory, himself an attendant of the workshop, had built already in 1950 an electromechanical mouse he called Theseus, which was able to navigate a maze and find its way out [43]. Furthermore, one year earlier Shannon had already presented at the National IRE Convention an important early paper on the automation of chess playing subsequently published as *Programming a Computer for Playing Chess* [69]. Around the same time Alan Turing published his famous paper *Computing Machinery and Intelligence* in which he described a “game” or test to determine whether a machine “thinks” [74]. Turing called this test “The Imitation Game,” a phrase greatly popularized by director Morten Tydlum’s homonymous movie from 2014 – currently the test is known simply as the *Turing test*. Norbert Wiener’s

seminal book *Cybernetics* (first edition in 1948) also cannot be omitted from this enumeration. One could, of course, go even further back in time and highlight the Spanish engineer Leonardo Torres y Quevedo’s chess playing automaton from the early 1910s, “the first *real* chess player” ([84], p. 598, my emphasis), but it should be obvious by now, that a specific “birthplace” (or even time) is impossible to be determined. There is rather a century-old fascination with the creation of human-like approximations, as legends like the one about the Golem, and stories like E.T.A. Hoffmann’s *The Sandman* (“Der Sandman,” 1816) or Mary Shelley’s *Frankenstein* (1818) show. It is only that this fascination does not manifest itself nowadays solely in the artistic field.

If one would try to plot the history of AI as a function on a three-dimensional coordinate system with the y-axis as the temporal dimension, the x-axis would stand for the predominant approach, and the z-axis for the degree of hype (with depressions usually being called “AI winters”). The two large competing approaches in AI are the symbolic and the connectionist ones or differently put, the historically opposing camps were *Logical Versus Analogical or Symbolic Versus Connectionist or Neat Versus Scruffy*, as an article of Minsky’s, in which he argues for cooperation instead of opposition, points in its title [55] (see, e.g., [35] and [62] for more context). Currently neural networks, i.e., connectionists, dominate the AI landscape. Along the temporal axis, the critique which often enough strongly targeted the field of AI and its practitioners would have to be documented. Some of the critique was agitated and strident interspersed with visions of impending doom, some was sensible and genuinely worried or constructive. Probably the two most prominent critics were the philosopher Hubert Deryfus with his two well-known publications *Alchemy and Artificial Intelligence* (a RAND technical report from 1956) and *What Computers Can’t Do* (first edition in 1972), and the computer scientist Joseph Weizenbaum, an important AI researcher himself, turned later AI critic with his very influential book *Computer Power and Human Reason* (1976).

The history of AI is to a large extent dominated by the United States, the reasons why AI has made such progress in the past two decades have important European roots though. The third part of this essay is about them.

12 Deep Learning: From European Curiosity to a Global Technological Shift

Jean-Marc Jézéquel

DEEP LEARNING has become one of the defining technologies of the twenty-first century, underpinning advances in computer vision, speech recognition, natural language processing, robotics, and autonomous systems. Its rapid diffusion across industry and society can give the impression of a sudden breakthrough. In reality, the foundations of deep learning were laid decades earlier through curiosity-driven research, much of it conducted in academic environments with little immediate prospect of application. A central figure in this trajectory is Yann LeCun, whose early work, following his doctoral studies in Paris, played a decisive role in shaping modern artificial intelligence.

LeCun received his Ph.D. in 1987 from Université Pierre et Marie Curie (Paris VI), where he worked under the supervision of Maurice Milgram, in a research environment influenced by European traditions in mathematics, signal processing, and theoretical computer science. His doctoral work focused on neural networks and learning algorithms at a time when connectionist approaches were viewed with skepticism by much of the artificial intelligence community. Symbolic AI dominated the field, while neural networks were often dismissed as biologically inspired curiosities lacking theoretical rigor and practical scalability.

In the late 1980s and early 1990s, LeCun pursued a line of research that combined convolutional architectures, gradient-based optimization, and end-to-end learning. His work on convolutional neural networks (CNNs) introduced a principled way to exploit spatial structure and translation invariance in visual data. This approach departed from handcrafted feature extraction pipelines, instead allowing systems to learn hierarchical representations directly from raw inputs. At the time, this methodology was computationally expensive and difficult to deploy, and its potential impact remained largely confined to specialized domains such as optical character recognition.

A notable early success of this research was the deployment of neural-network-based handwriting recognition systems in the 1990s, most prominently for reading handwritten digits on bank checks. These systems, developed at Bell Labs after LeCun moved to the United States, demonstrated that learning-based methods could outperform carefully engineered symbolic solutions in real-world tasks. Yet, despite these successes, neural networks remained a niche approach for many years, constrained by limited computational resources, small datasets, and a prevailing belief that learning-based systems could not scale to complex reasoning tasks.

The broader transformation began only decades later, when increases in computational power, the availability of large datasets, and algorithmic refinements converged. The resurgence of neural networks in the 2010s—now often referred to as the deep learning revolution—can be traced directly to the architectural and conceptual principles developed in this earlier period. Convolutional networks became the backbone of modern computer vision; gradient-based learning emerged as a unifying paradigm across perception and language; and representation learning replaced manual feature engineering as the dominant methodology in artificial intelligence.

The societal impact of this transformation has been profound. Deep learning systems now enable automatic image and speech recognition at human-level accuracy in many settings, power large-scale translation systems, support medical diagnosis, and form the basis of modern generative models. These technologies have reshaped industries ranging from healthcare and transportation to media and scientific research. Importantly, this impact was not the result of a targeted industrial program, but rather the delayed consequence of fundamental research pursued in academic settings without a clear path to commercialization.

LeCun's career is an excellent example of the fact that long-term investment in basic research, even when its utility is uncertain or contested, can yield transformative societal outcomes. The ideas that underpin deep learning were not optimized for immediate impact, nor were they developed in response to market demand. They emerged from sustained intellectual exploration, informed by mathematics, neuroscience, and computer science, and supported by research institutions willing to tolerate risk and delayed reward.

In retrospect, the rise of deep learning appears inevitable, much like the emergence of the Web or object-oriented programming. Yet this inevitability is visible only after the fact. At the time of their conception, the ideas developed by LeCun and his contemporaries occupied the margins of mainstream artificial intelligence research. Their eventual success reinforces the argument that Europe’s contribution to global computing lies not only in technological artifacts, but in research cultures that value depth, patience, and intellectual freedom.

13 The Transformer Revolution in Artificial Intelligence: From Recurrence to Global Attention

Ina Schieferdecker

THE history of artificial intelligence (AI) can be understood as a sequence of paradigm shifts, reflecting evolving ideas about how intelligence can be modelled and implemented in machines [65]. While foundational research in deep learning and earlier recurrent models made significant strides in processing language [19, 39], the development of the Transformer was driven by a fundamental shift in how researchers approached sequence modelling. This breakthrough was not merely the result of incrementally improving existing architectures, but of asking radical “what if?” questions about inherited assumptions [77].

Before the introduction of the Transformer architecture, the standard for processing language relied on models that evaluated information sequentially, such as Recurrent Neural Networks (RNNs) [48]. This step-by-step structure meant that a system could not fully process the end of a sentence until it had finished the beginning. Consequently, this sequential nature created a bottleneck, making training slow and preventing researchers from utilising the full power of modern parallel computing hardware [10, 33]. The curiosity that drove the breakthrough was therefore radical: researchers asked whether recurrence was actually necessary to model sequences at all [77]. Challenging the dominant mental model that language must be processed in strict order, they set out to test whether a system could instead look at an entire input sequence simultaneously and decide dynamically which information matters most.

The decisive shift occurred when researchers reframed the concept of “attention”. Originally, attention was introduced as an auxiliary mechanism—a way to let models “peek” at relevant parts of the input data to assist sequential recurrent processes [10]. The transformative leap happened when the Google Brain team including team members the German Research Center for Artificial Intelligence (DFKI) asked: what if attention is all you need? [77]. This reframing flipped the architecture inside-out. Instead of relying on sequential hidden states, the new model used self-attention to instantly weight the relevance of every word to every other word [77]. This allowed for a representation-first philosophy, where the structure of the data is separated from the computation itself. Rather than baking word order into the maths of the network structure, researchers injected it explicitly as data via positional encodings, keeping the core computation simple and highly parallelisable.

Crucially, this research was deeply driven by the physical realities of modern hardware. While sequential models were ill-suited to the architecture of modern graphics processing

units (GPUs), the Transformer was designed to be hardware-native, embracing massive parallelism. This approach meant that models could be trained much faster and scaled to vastly larger datasets than ever before [21, 63]. Although the original research focused on machine translation, the architecture proved remarkably general. Curiosity about whether attention could serve as a universal interface for learning quickly led to its rapid adoption across other modalities, including images, audio, and video [22].

Ultimately, the Transformer did not emerge solely from a desire to beat a translation benchmark or meet a commercial requirement. It was the result of questioning inherited assumptions, elevating auxiliary mechanisms to core components, and letting hardware realities influence theoretical design. The impact of this curiosity-driven work was immediate and far-reaching. It redirected global AI research away from complex recurrent structures towards simpler, highly scalable attention-based models, while also contributing to greater model interpretability by allowing researchers to inspect attention patterns [77].

Today, the societal implications of this research are profound. The Transformer serves as the foundational architecture of contemporary generative AI, enabling the emergence and rapid growth of influential organisations such as OpenAI, Anthropic, Hugging Face, and Stability AI. Furthermore, the authors' decision to release their code as open-source software [76] facilitated rapid worldwide adoption. These advances have led to faster, more accurate, and widely accessible AI technologies that continue to reshape modern ways of life.

14 Conclusions

Petre Sora

IT is easy to extol the virtues of curiosity-driven research. After all, who would not want to pursue freely his or her pet project leaving others to wonder about its usefulness and applicability? But then again, what do usefulness and applicability mean and how does one assess them? Especially if the assessment has to be done in advance. Even if one has the feeling of having stumbled upon “something big,” the impression might be confirmed by future events or – more often – will be not. It is only “after the dust has settled,” that an assessment—a real assessment, namely one that is not tainted by hopes, interests, and other biases, one that furthermore also takes into account the technology’s impact upon society – becomes possible. In this sense – in the historian’s sense – too much of the dust stirred by many of the technologies discussed here, is probably still in the air for a real assessment.

The important point this essays aims to make is not tied to the concrete technologies discussed though. They merely exemplify it. As the introduction already pointed out, examples of technologies or scientific breakthroughs borne out of curiosity-driven research could have been equally chosen from other geographies and such examples from other fields and from other historical periods abound. The important point is, that despite developments often appearing as necessary in retrospect, from the present’s point of view the future proves largely unpredictable. Too many aspects conspire together for it to be otherwise and furthermore, there is a certain “additive” quality in science and technology which lets previous discoveries and inventions influence future ones, sometimes in highly unexpected ways.

There is a very important case to be made for incremental research (e.g., [54]). There is also a very important case to be made for applicability and return on investment. But, as this essay hopes to have proven, there is also a very important case to be made for curiosity-driven research in Europe.

References

- [1] General Python FAQ. <https://docs.python.org/3/faq/general.html>. Accessed: 2025-12-02.
- [2] Python (programming language). [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)). Accessed: 2025-12-03.
- [3] Python Software Foundation. <https://www.python.org/psf-landing/>. Accessed: 2025-12-02.
- [4] The story of Python and how it took over the world—Python: The Documentary. <https://youtu.be/GfH4QL4VqJ0>. Accessed: 2025-12-03.
- [5] ACM. ACM Turing Award citation – Ole-Johan Dahl and Kristen Nygaard. https://amturing.acm.org/award_winners/dahl_6917600.cfm, 2025. Accessed: 2025-10-27.
- [6] Joyce Ayoola Adebusola, Adebisi Marion Olubunmi, Adebisi Ayodele Ariyo, Okesola Olatunji Julius, and Okeyinka Aderemi Elisha. An overview of 5G technology. *Department of Computer Science, Landmark University*, 2020.
- [7] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Pearson/Addison-Wesley, Boston, MA, USA, 2 edition, 2007.
- [8] Marco Aiello. *The Web Was Done by Amateurs: A Reflection on One of the Largest Collective Systems Ever Engineered*. Springer, 2018.
- [9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A Berkeley view of Cloud Computing. Technical report, UC Berkeley, 2009. Accessed: 2025-11-29.
- [10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.
- [11] Paul Baker, Zhen Ru Dai, Jens Grabowski, Øystein Haugen, Serge Lucio, Eric Samuelson, Ina Schieferdecker, and Clay E. Williams. The UML 2.0 testing profile. *OMG Official Standard*, 2004.
- [12] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. Harper San Francisco, 1999.
- [13] Ian Bird. Computing for the Large Hadron Collider. *Annual Review of Nuclear and Particle Science*, 61:99–118, 2011.
- [14] Daniele Bonacorsi et al. Running CMS software on grid testbeds. *arXiv:physics/0306038*, 2003. Accessed: 2025-11-29.

- [15] László Böszörményi, Jürg Gutknecht, and Gustav Pomberger, editors. *The School of Niklaus Wirth: The Art of Simplicity*. dpunkt.verlag, Heidelberg, Germany, 2000.
- [16] Rolv Bræk. *SDL basics*. Sintef Delab, 1993.
- [17] CERN. Technical design report for the LHC Computing Grid (LCG). Technical report, CERN, 2005. Accessed: 2025-11-29.
- [18] Alain Colmerauer and Philippe Roussel. The birth of Prolog. In *History of Programming Languages—II*, pages 331–367. ACM, 1996.
- [19] Daniel Crevier. *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, 1993.
- [20] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon’s highly available key-value store. *ACM SIGOPS operating systems review*, 41(6), 2007.
- [21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.
- [22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
- [23] Hubert L. Dreyfus. *What Computers Still Can’t Do: A Critique of Artificial Reason*. MIT Press, Cambridge, MA, USA, 1992.
- [24] David M. Eberhard, Gary F. Simons, and Alison J. Robinson, editors. *Ethnologue: Languages of the World*. SIL Global, Dallas, TX, USA, twenty-ninth edition, 2026. Accessed: Feb. 28, 2026. Online version: <https://www.ethnologue.com/>.
- [25] European Commission. European Open Science Cloud (EOSC) Strategic Implementation Plan. https://research-and-innovation.ec.europa.eu/strategy/strategy-research-and-innovation/our-digital-future/open-science/european-open-science-cloud-eosc_en, 2015. Accessed: 2025-11-29.
- [26] Edward A. Feigenbaum and Pamela McCorduck. *The Fifth Generation: Artificial Intelligence and Japan’s Computer Challenge to the World*. Michael Joseph, London, UK, 1984.

- [27] Abraham Flexner. *The Usefulness of Useless Knowledge*. Princeton University Press, 2017.
- [28] Gaia-X European Association for Data and Cloud AISBL. Gaia-X Architecture Document. <https://docs.gaia-x.eu/technical-committee/architecture-document/21.06/>, 2021. Accessed: 2025-11-29.
- [29] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
- [30] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid. *International Journal of High Performance Computing Applications*, 15(3), 2001.
- [31] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Proceedings of the Grid Computing Environments Workshop (GCE'08)*, pages 1–10, 2008.
- [32] Neil Geddes. The Large Hadron Collider and Grid computing. *Philosophical Transactions of the Royal Society A*, 370:965–977, 2012.
- [33] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.
- [34] Wolfgang K. Giloi. Konrad Zuse’s Plankalkül: The First High-Level, “non von Neumann” Programming Language. *IEEE Ann. Hist. Comput.*, 19(2):17–24, Apr./Jun. 1997.
- [35] Ashok K. Goel. Looking back, looking ahead: Symbolic versus connectionist AI. *AI Mag.*, 42(4):83–85, Dec. 2021.
- [36] Jens Grabowski, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, and Colin Willcock. An introduction to the testing and test control notation (TTCN-3). *Computer Networks*, 42(3):375–403, 2003.
- [37] Godfrey Harold Hardy. *A mathematician’s apology*. Cambridge University Press, 1992.
- [38] Friedhelm Hillebrand. *GSM and UMTS: The Creation of Global Mobile Communication*. John Wiley & Sons, Chichester, UK, 2002. Reference work on the history of GSM standardization and the development of SMS.
- [39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [40] Ellis Horowitz. *Fundamentals of Programming Languages*. Springer, Berlin, Germany, 1983.
- [41] Ivar Jacobson. *Object-oriented development in an industrial environment*. PhD thesis, Stockholm University, Department of Computer Science, Stockholm, Sweden, 1985.

- [42] Jeremy Johnson. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6), 1976.
- [43] Daniel Klein. Mighty mouse. *MIT News Magazine*, Dec. 2018.
- [44] P. Körner, M. Leuschel, J. Barbosa, et al. Fifty years of Prolog and beyond. *Theory and Practice of Logic Programming*, 22(6):776–858, 2022.
- [45] Bent Bruun Kristensen, Ole Lehrmann Madsen, and Birger Møller-Pedersen. The when, why and why not of the BETA programming language. In Barbara G. Ryder and Brent Hailpern, editors, *Proceedings of the Third ACM SIGPLAN History of Programming Languages Conference (HOPL-III)*, pages 1–57. ACM, 2007.
- [46] John Larmouth. *ASN.1 Complete*. Open Systems Solutions, 1999.
- [47] E. Laure, S. M. Fisher, A. Frohner, C. Grandi, and P. Kunszt. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.
- [48] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.
- [49] Javier Castro León and CERN Cloud Infrastructure Team. Advanced features of the CERN OpenStack Cloud. *EPJ Web of Conferences*, 214:07026, 2019.
- [50] Michael S. Mahoney. The histories of computing(s). *Interdisciplinary Science Reviews*, 30(2):119–135, 2005.
- [51] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982.
- [52] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., 1st edition, 1988.
- [53] Bertrand Meyer. Applying “design by contract”. *Computer*, 25(10):40–51, October 1992.
- [54] Bertrand Meyer. Incremental Research vs. Paradigm-Shift Mania. *Commun. ACM*, 55(9):8–9, Sep. 2012.
- [55] Marvin L. Minsky. Logical Versus Analogical or Symbolic Versus Connectionist or Neat Versus Scruffy. *AI Mag.*, 12(2):34–51, Jun. 1991.
- [56] Glyn Moody. *Rebel code: Linux and the open source revolution*. Perseus Books, USA, 2001.
- [57] David Nofre. A Compelling Image: The Tower of Babel and the Proliferation of Programming Languages During the 1960s. *IEEE Ann. Hist. Comput.*, 47(1):22–35, Jan./Mar. 2025.

- [58] Steven Pemberton and Martin Daniels. *Pascal Implementation: The P4 Compiler and Interpreter*. Ellis Horwood, Chichester, UK, 1986.
- [59] Tim Peters. PEP 20—The Zen of Python. <https://peps.python.org/pep-0020/>. Accessed: 2025-12-02.
- [60] Tomas Petricek. *Cultures of Programming: The Development of Programming Concepts and Methodologies*. Cambridge Univ. Press, 2026.
- [61] Diarmuid Pigott. Online Historical Encyclopaedia of Programming Languages. Accessed: Feb. 17, 2026. [Online]. Available: <https://hop1.info/>.
- [62] Lindsay Poirier. Neat Versus Scruffy: How Early AI Researchers Classified Epistemic Cultures of Knowledge Representation. *IEEE Ann. Hist. Comput.*, 47(2):7–19, Apr./Jun. 2025.
- [63] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [64] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. 21(2), 1978.
- [65] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4 edition, 2021.
- [66] Jean E. Sammet. *Programming Languages: History and Fundamentals*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1969.
- [67] Ina Schieferdecker, Zhen Ru Dai, Jens Grabowski, and Axel Rennoch. The UML 2.0 testing profile and its relation to TTCN-3. In *Testing of Communicating Systems (Test-Com 2003)*. Springer, 2003.
- [68] Oshani Seneviratne and James Hendler, editors. *Linking the World’s Information: Essays on Tim Berners-Lee’s Invention of the World Wide Web*, volume 52. Association for Computing Machinery, New York, NY, USA, 1 edition, 2023.
- [69] Claude E. Shannon. Programming a Computer for Playing Chess. *Philosophical Mag.*, 41(314):256–275, Mar. 1950.
- [70] Software Preservation Group. Prolog Project Archive. <https://www.softwarepreservation.org/projects/prolog/>, 2025. Accessed: 2025-10-27.
- [71] Harald Stockinger et al. Grid data management in action: Experience in running and supporting data management services in the eu datagrid project. *arXiv:cs/0306011*, 2003. Accessed: 2025-11-29.
- [72] Achim Streit et al. Unicore—from project results to production Grids. *arXiv:cs/0502090*, 2005. Accessed: 2025-11-29.

- [73] Andrew S. Tanenbaum. *Operating systems: design and implementation*. Prentice-Hall, Inc., USA, 1987.
- [74] A. M. Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, Oct. 1950.
- [75] Hans van der Veer and Anthony Wiles. Achieving technical interoperability - the ETSI approach. Technical Report White Paper No. 3, European Telecommunications Standards Institute (ETSI), 2008.
- [76] Ashish Vaswani, Samy Bengio, Eugene Brevdo, et al. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*, 2018.
- [77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [78] Wikipedia contributors. Fifth Generation Computer Systems — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Fifth_Generation_Computer_Systems, 2025. Accessed: 2025-10-27.
- [79] Wikipedia contributors. Modula-2. <https://en.wikipedia.org/wiki/Modula-2>, 2025. Accessed: 2025-11-26.
- [80] Wikipedia contributors. Prolog—Wikipedia, the free encyclopedia. <https://en.wikipedia.org/wiki/Prolog>, 2025. Accessed: 2025-10-27.
- [81] Wikipedia contributors. Simula—Wikipedia, the free encyclopedia. <https://en.wikipedia.org/wiki/Simula>, 2025. Accessed: 2025-10-31.
- [82] Niklaus Wirth. *Algorithms + Data Structures = Programs*. Prentice Hall, Englewood Cliffs, NJ, 1976.
- [83] Worldwide LHC Computing Grid. CERN WLCG Overview. <https://home.cern/science/computing/grid>, 2025. Accessed: 2025-11-29.
- [84] Heinz Zemanek. Central European Prehistory of Computing. In Nicholas C. Metropolis, Jack Howlett, and Gian-Carlo Rota, editors, *A History of Computing in the Twentieth Century: A Collection of Essays*, pages 587–609. Academic Press, 1980.
- [85] Konrad Zuse. *The Computer—My Life*. Springer, Berlin, Germany, 1993.

About the Authors

The authors include members of the board of Informatics Europe, with the addition of Petre Sora, who helped uniform and improve the overall document.

For enquiries and feedback about this report,
please contact communications@informatics-europe.org



www.informatics-europe.org
© Informatics Europe, 2026
CC BY-SA 4.0

