Theoretical Questions of Computer Science, Computational Mathematics, Computer Science and Cognitive Information Technologies

https://doi.org/10.25559/SITITO.021.202503.391-397
UDC 004

Constructive Computer Science at ETH Zuerich

Jürg Gutknecht

Swiss Federal Institute of Technology, Zurich, Switzerland

Address: 101 Rämistrasse, Zurich 8092, Switzerland

gutknecht@gmail.com

Original article

Abstract

The article examines the emergence and development of constructive computer science at ETH Zürich as a distinctive tradition in which computer science is understood through the design and implementation of real programming languages, compilers, workstations, operating systems, and integrated software environments. The discussion begins with the early computing activities of the Institute of Applied Mathematics, including the use of the Zuse Z4, the construction of ERMETH, and the role of ALGOL 60 in introducing structured programming concepts and formal language description. The article then follows the work of Niklaus Wirth from Pascal and the portable P-code machine to Modula-2, M-code, and the Lilith workstation, where hardware, system software, device drivers, applications, and the user interface were conceived as a coherent modular system. A further stage is represented by Ceres and Oberon, the introduction of type extension, the Gadgets graphical framework, and the hardware description language Lola. The central argument is that the ETH Zürich tradition was important not merely because it produced influential languages and machines, but because it demonstrated a disciplined engineering philosophy based on simplicity, strong typing, portability, modular decomposition, and close alignment between programming language design and computer construction. The article therefore presents constructive computer science as both a historical achievement and a methodological model for safe, comprehensible, and integrated system development.

Keywords: constructive computer science, ETH Zuerich, systems construction, programming languages, Niklaus Wirth

Conflict of interests: The author declares no conflict of interests.

For citation: Gutknecht J. Constructive Computer Science at ETH Zuerich. *Modern Information Technologies and IT-Education.* 2025;21(3):391-397. https://doi.org/10.25559/SITITO.021.202503.391-397

Теоретические вопросы информатики, прикладной математики, компьютерных наук и когнитивно-информационных технологий

Конструктивная информатика в ETH Zurich

Ю. Гуткнехт

Швейцарская высшая техническая школа Цюриха (ETH Zürich), г. Цюрих, Швейцария

Адрес: 8092, Швейцария, г. Цюрих, Ремиштрассе, д. 101

gutknecht@gmail.com

Оригинальная статья

Аннотация

Статья посвящена становлению и развитию конструктивной информатики в ETH Zürich как особого направления, в котором компьютерная наука рассматривается прежде всего через создание работающих языков программирования, компиляторов, аппаратно-программных систем и пользовательских сред. В центре внимания находится историческая линия от ранних вычислительных проектов Института прикладной математики, включая использование Zuse Z4 и создание ERMETH, к языку ALGOL 60 и последующим работам Никлауса Вирта. Особое внимание уделяется переходу от Pascal и P-code к Modula-2, M-code и рабочей станции Lilith, где программное обеспечение, операционная система и прикладные компоненты были организованы как однородная сеть модулей. Далее рассматривается модернизация этой линии в паре Ceres-Oberon, введение расширения типов, развитие графической среды Gadgets и применение языка описания аппаратуры Lola в учебных и исследовательских проектах. Аннотация подчёркивает, что ключевой вклад ETH Zürich состоял не только в отдельных языках или машинах, но и в целостной инженерной культуре простоты, строгой типизации, переносимости, модульности и тесной связи между языком, компилятором, архитектурой и системой. Работа показывает, что конструктивная традиция ETH оказала существенное влияние на академическую информатику, образование программистов и практику системного проектирования.

Ключевые слова: конструктивная информатика, ETH Zurich, построение систем, языки программирования, Никлаус Вирт

Конфликт интересов: автор заявляет об отсутствии конфликта интересов.

Для цитирования: Гуткнехт Ю. Конструктивная информатика в ETH Zurich // Современные информационные технологии и ИТ-образование. 2025. Т. 21, № 3. С. 391-397. https://doi.org/10.25559/SITITO.021.202503.391-397

First Constructive Phase

Computer Science at ETH and in particular the Department of Informatics grew out of Numerical Mathemathics in the 1950s. In 1950 a member at the renowed "Institute of Applied Mathematics", professor Eduard Stiefel and staff decided to acquire on loan one of the very first computers worldwide, a German Zuse Z4. No real programming language was available at this time just something called "Plankalkül" instead. As a proof of concept, Stiefel used Plankalkül to demonstrate the Z4´s power by a calculation of the Euler number "e" up to a very large number of positions.

Soon after, however, Stiefel and his colleague Heinz Rutishauser came up with the idea of building their own computer, which they did and called it ERMETH, see Figure 1. ERMETH became operational in 1957. Its relais based architecture was composed of roughly 1500 tubes and diodes with an attached disk storage of a capacity of 10000 words and a weight of 1.5 tons. This time, the institute's proof of concept was computing "pi". In 1958 an intercontinental team of scientists gathered at ETH with the aim of developing a more advanced programming language called Algol (for algorithmic language). The first result was Algol 58, followed by Algol 60 [1]. Although strongly geared towards mathematical applications, Algol introduced some key programming concepts like structured programming, recursive functions, and a meta language called BNF (Backus Naur form) for formally defining syntaxes [1], [11].

F i g. 1. The Institute of Applied Mathematics and its ERMETH computer

Source: ETH Library Zurich, picture archive.

After Algol 60 the path of language design split in two. On the one hand the Algol group continued the Algol development up to a version Algol68 that was not received well because of its excessive complexity and clumsiness [12]. On the other hand, a young engineer started to develop a lightweight version of Algol, first called Algol W and then Pascal. The name of this engineer was Niklaus Wirth, and Pascal was finally released together with a virtual P-code machine acting as a frontend target for compilers [2], [9]. This was a significant innovation at the time that (1.) simplified any compiler writer's life and (2.) made compiler frontends inherently portable. As a consequence, a number of Pascal compilers like UCSD Pascal from the University of California at San Diego became quickly available, and Pascal spread world-wide [13]. In 1984 Wirth won the prestigious Turing Award [7].

The P-code embodies a model of a simple stack oriented architecture that releives compiler frontends from tasks like register management, memory management, and the like. Here is an example of an arithmetic expression and its translation into P-code:

```
• Arithmetic expression (a + b*(x - y) / b
• P-code LD a LD b LD x LD y - * + LD b /
  where LD means loading a value onto the stack
```

However, Pascal compilers are not just comparatively simple pieces of software, they are strong in terms of early recognition of programming errors, mainly due to Pascal's "strong typing" principle that does not allow data of one type to be treated as data of another type [10], [22].

Second Constructive Phase

The second and up to now last constructive phase of Computer Science at ETH started in 1980 after Niklaus Wirth returned from a sabbatical at the famous research lab Xerox PARC in California. It was the time when "personal computer", if at all, was identified with keyboard and a 24 x 80 screen as its primitive user interface. In Palo Alto, Wirth learnt about the Xerox Alto computer, the first representative of a radically new generation of personal workstations. Bitmap based displays and "mouse-like" input devices meant a substantial upgrade of the user interface, and a significantly increased level of computing power allowed for totally new applications like advanced text processing, graphic designs, and photo editing, to name just a few. The underlying motto was WYSIWYG, "what you see is what you get" [17], [21].

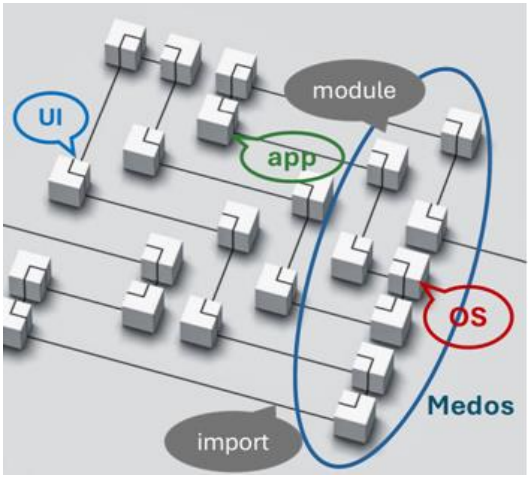
In any case, Wirth was fascinated by the new generation of computing workstations and he became eager to build one of his own, however, following his trademark, at just a fraction of the Alto's complexity [7], [8]. The final result of this endeavor was Lilith, the first personal workstation of its kind in Europe. Several editions of ever smaller versions of the Lilith were finally built, see Figure 2 [16], [24]. What was Wirth's and his team's scientific approach?

F i g. 2. The Lilith workstation (final version)

Source: Wexelblat R. L. History of Programming Languages. Academic Press, 1981. 758 p. https://doi.org/10.1016/C2013-0-11690-X

First, both the Pascal language and the associated P-code had to be upgraded carefully from a language for specifying algorithms to a language for specifying systems, or, put differently from a language for programming in the small to programming in the large, while preserving most of Pascal's characteristics. The main upgrade of the new language called Modula and later Modula-2 against Pascal was the module construct [3]. Modules are building blocks of a software system [18]. They come in pairs of implementation and definition, where the latters are subject to be imported from other modules.

Second, the P-code was upgraded to an M-code that, as a technical innovation, was microcoded in hardware, more precisely in 16 bit AMD 2901 processor hardware. Thanks to this implementation of the M-code directly in hardware, any kind of a "low level" system software cushion between hardware and application software, typically implemented in an unsafe assembler language was abandoned with the Lilith's system design [23]. In the end the entire software from the operating system (with name Medos) and device drivers all the way up to applications and user interface presented itself quite uniformly as one homogeneous network of Modula-2 modules, see Figure 3 [3], [25].



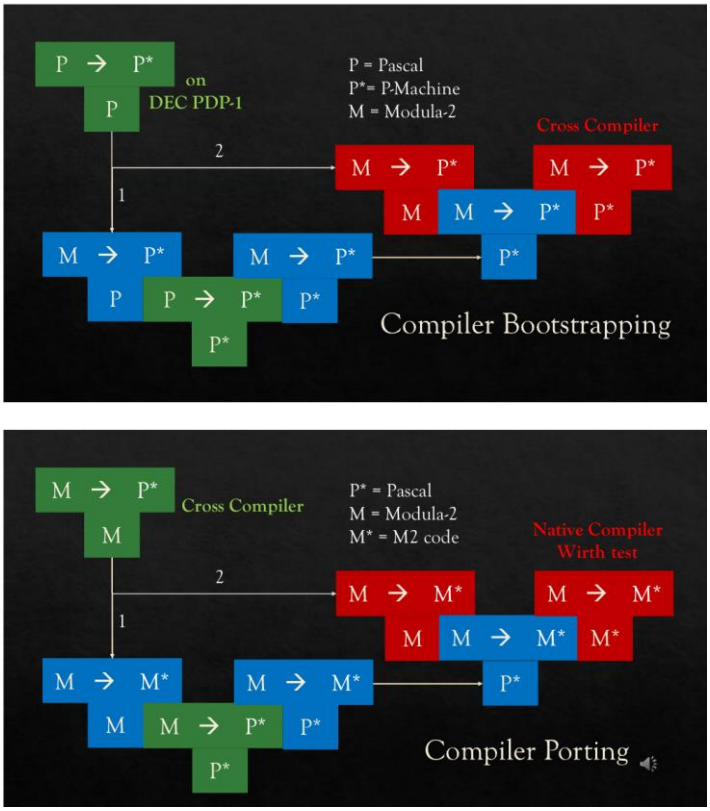
F i g. 3. The homogeneous Modula-2 Software System

Source: Compiled by the author based on S.E. Knudsen, Medos-2: A Modula-2 Oriented Operating System for the Personal Computer Lilith. ETH Zürich, 1983.

Thanks to the M-code, writing Modula-2 compilers was a relatively easy exercise [3]. Nevertheless, it had to be done, and it was done in two steps: (1.) generating a Modula-2 crosscompiler running on a different platform, and (2.) porting the crosscompiler to the Lilith platform. Figure 4 illustrates these two steps in terms of so-called "Tombstone diagrams" also called "T diagrams". Notably the first T diagram in the development of the cross compiler represents an existing Pascal compiler running on a DEC PDP-1 computer, while the final diagram in the porting series allows the "Wirth test" to be performed, namely compiling the compiler with itself and comparing the two involved object codes (the two codes should be equal) [7].

Modula-2 was made available on several platforms, and it was used internationally in a diversity of mostly safety-critical applications [3]. However, it did not match Pascal in its comprehensive worldwide spread. Also, an attempt to commercialize Lilith by a company called DISER (Digital Image and Sound) was not successful. The commercial time was simply outrunning academia.

However, the academic time was not stopping either. Both hardware and software developed quickly. Hardware according to Gordon Moore's law stating that (simplified a bit) hardware performance doubles every two years. (Looking back from today this law cannot be confirmed in terms of processor performance that drags behind but has significantly been outdone in terms of memory capacity.) In contrast, software started to slow down mostly because of increasing complexity such as object-oriented programming, a situation bearing a sarcastic slogan "Hardware is getting faster slower than software is getting slower" [8].



F i g. 4. The two-phased Modula-2 Compiler Construction

Source: Compiled by the author based on L.B. Geissmann, Separate Compilation in Modula-2 and the Structure of the Modula-2 Compiler on the Personal Computer Lilith, ETH Zürich, 1983, and Astrobe's historical materials on Lilith/Modula-2.

In the year 1986 the Wirth group decided to replace the outdated pair (Lilith, Modula-2) with a modernized pair (Ceres, Oberon), see Figure 5 [5], [19]. Ceres was put on a 32 bit NS2032 processor architecture replacing the bitsliced 16 bit AMD 2901 architecture used for Lilith. A noticeable progress language-wise from Modula-2 to Oberon was the introduction of a very lightweight version of object-orientation called type-extension [4], [14]. Type-extension allows new structures (record types) to be derived from existing ones, without giving up strong type-checking at compile time [4], [6]. For example,

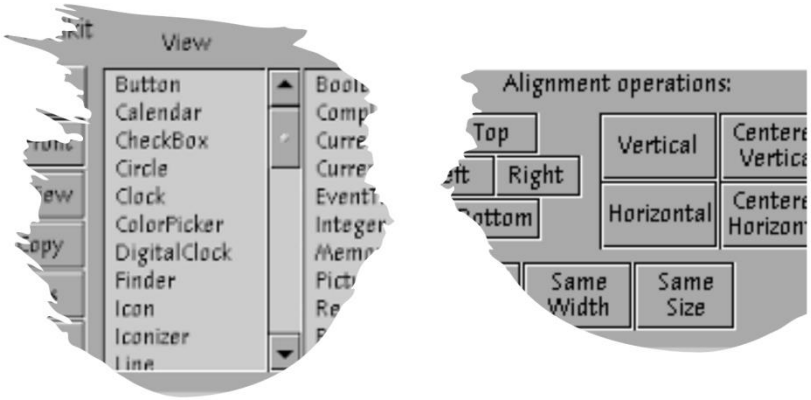
```
TYPE
    T = RECORD a : ... ; b : c : ... END ;
    T1 = RECORD(T) c : ,,, ; d: ... END ;
```

in Oberon means that type T1 is derived ("extended") from type T, inheriting all its infrastructure and being type-compatible with T.

F i g. 5. Niklaus Wirth and the Ceres Workstation

Source: Heinz Nixdorf MuseumsForum.

As a proof of concept, type extension was heavily used and tested in a project for developing a modern graphical user interface (GUI). The outcome was Gadgets, a framework that would not have to hide itself still today, s. Figure 6 [20]. In Gadgets, each component like "menu", "textfield", "slider", "checkbox" etc. is derived via type extension from a base type gadget. The essential point here is that, thanks to type extension, new components can smoothly be integrated into the Gadgets system at any time. As a matter of fact, new GUI components were developed and neatly integrated into Gadgets years after its development [20], [21].



F i g. 6. The Gadgets GUI Framework

Source: Fischer A., Mössenböck H. The Oberon Companion: A Guide to Using and Programming Oberon System 3. vdf Verlag der Fachhochschulen, Zurich, 1997. Fig. 3.6-3.7.

A final and last step towards integrated system construction concluding the second phase is based on FPGA (Field Programmable Gate Arrays) technology. Wirth developed a hardware description language called Lola with a syntax and style along the lines of Modula-2 and Oberon but with a completely different semantics [7], [15].

For example, the Lola excerpt below describes no actions but rather a composition of hardware elements such as bits an registers :

```
TYPE Counter(N) ;
VAR
    ci : BIT ; co : BIT ;
    c : [N] BIT ; q [N] BIT ;
BEGIN
    q.0 := REG(q.0 - ci) ; c.0 := q.0 * ci ;
    FOR i := 1.. N-1 DO
        q.i := REG(q.i - c[i - 1]) ; c.i :=
q.0 * c[i - 1]
    END ;
    co : = c[N - 1]
END Counter
```

The Lola language was used in student labs and, quite recently, by a developer outside of ETH for recreating the Ceres hardware.

Summary

We can identify two phases of constructive computer science at ETH, the first phase growing out of applied mathematics in around the 1950s and 1960s, and a second phase representing a pioneering effort towards making personal workstations and their concept known and available in Europe. In both phases the development of simple, safe and powerful programming languages attracting an international community of users was a principal issue and result [15], [16], [24].

References

1. Backus J.W., Bauer F.L., Green J. et al. Report on the algorithmic language ALGOL 60. *Communications of the ACM.* 1960;3(5):299-311. https://doi.org/10.1145/367236.367262
2. Wirth N. The programming language pascal. *Acta Informatica.* 1971;1(1):35-63. https://doi.org/10.1007/bf00264291
3. Wirth N. Modula: A language for modular multiprogramming. *Software: Practice and Experience.* 1977;7(1):1-35. https://doi.org/10.1002/spe.4380070102
4. Wirth N. The programming language oberon. *Software: Practice and Experience.* 1988;18(7):671-690. https://doi.org/10.1002/spe.4380180707
5. Wirth N. From modula to oberon. *Software: Practice and Experience.* 1988;18(7):661-670. https://doi.org/10.1002/spe.4380180706
6. Wirth N., Gutknecht J. The oberon system. *Software: Practice and Experience.* 1989;19(9):857-893. https://doi.org/10.1002/spe.4380190905
7. Wirth N. From programming language design to computer construction. *Communications of the ACM.* 1985;28(2):160-164. https://doi.org/10.1145/2786.2789
8. Wirth N. A plea for lean software. *Computer.* 1995;28(2):64-68. https://doi.org/10.1109/2.348001
9. Wirth N. Program development by stepwise refinement. *Communications of the ACM.* 1971;14(4):221-227. https://doi.org/10.1145/362575.362577

10. Hoare C.A.R. An axiomatic basis for computer programming. *Communications of the ACM.* 1969;12(10):576-580. https://doi.org/10.1145/363235.363259
11. Dijkstra E.W. Letters to the editor: go to statement considered harmful. *Communications of the ACM.* 1968;11(3):147-148. https://doi.org/10.1145/362929.362947
12. Hoare C.A.R. The emperor's old clothes. *Communications of the ACM.* 1981;24(2):75-83. https://doi.org/10.1145/358549.358561
13. Wirth N. 50 years of Pascal. *Communications of the ACM.* 2021;64(3):39-41. https://doi.org/10.1145/3447525
14. Dotzel G., Skulski W., Dubois P.F. Oberon-2, A High-Performance Alternative To C++. *Computers in Physics.* 1997;11(1):81. https://doi.org/10.1063/1.4822520
15. Tkachov F.V. Less is more. Why Oberon beats mainstream in complex applications. *Journal of Physics: Conference Series.* 2014;523:012011. https://doi.org/10.1088/1742-6596/523/1/012011
16. Severance C. The Art of Teaching Computer Science: Niklaus Wirth. *Computer.* 2012;45(7):8-10. https://doi.org/10.1109/mc.2012.245
17. Kay A., Goldberg A. Personal Dynamic Media. *Computer.* 1977;10(3):31-41. https://doi.org/10.1109/c-m.1977.217672
18. Parnas D.L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM.* 1972;15(12):1053-1058. https://doi.org/10.1145/361598.361623
19. Wirth N. Oberon: A system for workstations. *Microprocessing and Microprogramming.* 1988;24(1-5):3-8. https://doi.org/10.1016/0165-6074(88)90017-8
20. Gutknecht J. Oberon, gadgets, and some archetypal aspects of persistent objects. *Information Sciences.* 1996;93(1-2):65-86. https://doi.org/10.1016/0020-0255(96)00061-8
21. Swinehart D.C., Zellweger P.T., Beach R.J., Hagmann R.B. A structural view of the Cedar programming environment. *ACM Transactions on Programming Languages and Systems.* 1986;8(4):419-490. https://doi.org/10.1145/6465.6466
22. Hoare C.A.R., Wirth N. An axiomatic definition of the programming language PASCAL. *Acta Informatica.* 1973;2(4):335-355. https://doi.org/10.1007/bf00289504
23. Reimer M. Implementation of the database programming language modula/R on the personal computer lilith. *Software: Practice and Experience.* 1984;14(10):945-956. https://doi.org/10.1002/spe.4380141005
24. Garfinkel S., Spafford E.H. In Memoriam: Niklaus Wirth. *Communications of the ACM.* 2024;67(3):20-20. https://doi.org/10.1145/3641309
25. Geissmann L. Der Lilith-Debugger. Ein modernes Werkzeug zur Fehlersuche in Modula-2 Programmen/ The Lilith Debugger. A modern tool for debbugging Modula-2 Programs. *it – Information Technology.* 1985;27(1-6):95-106. https://doi.org/10.1524/itit.1985.27.16.95

Submitted 12.08.2025; approved after reviewing 06.09.2025; accepted for publication 24.09.2025.

Поступила 12.08.2025; одобрена после рецензирования 06.09.2025; принята к публикации 24.09.2025.

About the author:

Jürg Gutknecht, Professor of Computer Science, Ph. D. in Mathematics, Swiss Federal Institute of Technology (101 Rämistrasse, Zurich 8092, Switzerland), gutknecht@gmail.com

The author has read and approved the final manuscript.

Об авторе:

Гуткнехт Юрг, профессор, Ph. D. in Mathematics, Швейцарская высшая техническая школа Цюриха (8092, Швейцария, г. Цюрих, Рамиштрассе, д. 101), gutknecht@gmail.com

Автор прочитал и одобрил окончательный вариант рукописи.