In Memoriam
Edsger Wybe Dijkstra
(1930–2002)

Contents

Edsger Wybe Dijkstra, Professor Emeritus in the Computer Sciences Department at the University of Texas at Austin, died of cancer on 6 August 2002 at his home in Nuenen, the Netherlands. Dijkstra was a towering figure whose scientific contributions permeate major domains of computer science. He was a thinker, scholar, and teacher in renaissance style: working alone or in a close-knit group on problems of long-term importance, writing down his thoughts with exquisite precision, educating students to appreciate the nature of scientific research, and publicly chastising powerful individuals and institutions when he felt scientific integrity was being compromised for economic or political ends. In the words of Professor Sir Tony Hoare, FRS, delivered by him at Dijkstra's funeral:

> Edsger is widely recognized as a man who has thought deeply about many deep questions; and among the deepest questions is that of traditional moral philosophy: How is it that a person should live their life? Edsger found his answer to this question early in his life: He decided he would live as an academic scientist, conducting research into a new branch of science, the science of computing. He would lay the foundations that would establish computing as a rigorous scientific discipline; and in his research and in his teaching and in his writing, he would pursue perfection to the exclusion of all other concerns. From these commitments he never deviated, and that is how he has made to his chosen subject of study the greatest contribution that any one person could make in any one lifetime.

Throughout his scientific career, Dijkstra formulated and pursued the highest academic ideals of scientific rigor untainted by commercial, managerial, or political considerations. Simplicity, beauty, and eloquence were his hallmarks, and his uncompromising insistence on elegance in programming and mathematics was an inspiration to thousands. He judged his own work by the highest standards and set a continuing challenge to his many friends to do the same. For the rest, he willingly undertook the role of Socrates, that of a gadfly to society, repeatedly

goading his native and his adoptive country by remarking on the mistakes inherent in fashionable ideas and the dangers of time-serving compromises.

0 Early Years

Dijkstra was born in 1930 in Rotterdam, The Netherlands, the son of a chemist father and a mathematician mother. He was the third of four children. His father was a teacher of chemistry in a high school, and later its superintendent. His father achieved prominence in his field, becoming the president of the Dutch Chemical Society. His mother had a lasting influence on his approach to mathematics and his emphasis on elegance. Though she never held a formal job, "she had a great agility in manipulating formulae and a wonderful gift for finding very elegant solutions" [19].

Dijkstra had considered a career in law during his last years in high school. He had hoped to represent the Netherlands in the United Nations. His performance in his final exam was extraordinary; he graduated in 1948 with the highest possible marks in mathematics, physics, chemistry, and biology. So his teachers and parents counseled him to seek a career in the sciences. He decided to join the University of Leyden, to study mathematics and physics during the first years, and theoretical physics later on.

In the early 1950s, electronic computers were a novelty. Dijkstra stumbled on his career quite by accident. His father had seen an advertisement for a three-week course in programming for the electronic computer EDSAC, to be given at Cambridge University, and he advised his son to attend the course. Dijkstra felt that computers might become an important tool for theoretical physicists, and he decided to attend this course in September 1951. When he asked his supervisor, prof. dr. A. Haantjes, for a letter of recommendation, it turned out that prof. Haantjes knew A. van Wijngaarden, the director of the Computation Department at the Mathematical Center in Amsterdam, and he knew that van Wijngaarden had taken the same course in 1950. Prof. Haantjes phoned van Wijngaarden, who invited Dijkstra to visit the Mathematical Center and wrote the letter of recommendation. Van Wijngaarden also offered Dijkstra a job, which he accepted, becoming officially the Netherlands' first "programmer" in March 1952.

For some time Dijkstra remained committed to physics, working on it in Leyden three days out of each week. With increasing exposure to computing, however, his focus began to shift. As he recalled:

> After having programmed for some three years, I had a discussion with A. van Wijngaarden, who was then my boss at the Mathematical Center in Amsterdam, a discussion for which I shall remain grateful to him as long as I live. The point was that I was supposed to study theoretical physics at the University of Leyden simultaneously, and as I found the two activities harder and harder to combine, I had to make up my mind, either to stop programming and become a real, respectable theoretical physicist, or to carry my study of physics to a formal completion only, with a minimum of effort,

and to become ..., yes what? A programmer? But was that a respectable profession? For after all, what was programming? Where was the sound body of knowledge that could support it as an intellectually respectable discipline? I remember quite vividly how I envied my hardware colleagues, who, when asked about their professional competence, could at least point that they knew everything about vacuum tubes, amplifiers and the rest, whereas I felt that, when faced with that question, I would stand empty-handed. Full of misgivings I knocked on van Wijngaarden's office door, asking him whether I could speak to him for a moment; when I left his office a number of hours later, I was another person. For after having listened to my problems patiently, he agreed that up till that moment there was not much of a programming discipline, but then he went on to explain quietly that automatic computers were here to stay, that we were just at the beginning and could not I be one of the persons called to make programming a respectable discipline in the years to come? This was a turning point in my life and I completed my study of physics formally as quickly as I could [doing his doctoraal examen in Leyden in March 1956]. One moral of the above story is, of course, that we must be very careful when we give advice to younger people: sometimes they follow it! [9]

When Dijkstra married Maria (Ria) C. Debets in 1957, he was required as a part of the marriage rites to state his profession. He stated that he was a programmer, which was unacceptable to the authorities, there being no such profession at that time in the Netherlands.

1   Mathematical Center, Amsterdam

The early computers were built out of vacuum tubes. They occupied large amounts of space, consumed power voraciously, and failed regularly. As Dijkstra recalls, "In retrospect one can only wonder that those first machines worked at all, at least sometimes. The overwhelming problem was to get and keep the machine in working order."[9]

He worked closely with Bram J. Loopstra and Carel S. Scholten, who had been hired to build a computer. Their mode of interaction remains a model of disciplined engineering: They would first decide upon the interface between the hardware and the software, by writing a programming manual. Then the hardware designers would have to be faithful to their part of the contract, while Dijkstra, the programmer, would write software for the nonexistent machine. Two of the lessons he learned from this experience were the importance of clear documentation, and that program debugging can be largely avoided through careful design.

One of his earliest algorithms was for the Shortest Path problem. The problem is as follows: given a number of cities on a map and distances between some of them, what is the shortest path between two designated cities? The problem can be set in a more abstract fashion: Find a shortest path between two given vertices in a directed network in which each edge has a non-negative length;

the lengths need not satisfy the triangle inequality. This is a problem of considerable practical importance, and the best known algorithms had running times which were cubic in the size of the network. The running time of Dijkstra's algorithm grew only as the square of the network size.

Dijkstra formulated and solved the Shortest Path problem for a demonstration at the official inauguration of the ARMAC computer in 1956, but —because of the absence of journals dedicated to automatic computing— did not publish the result until 1959. The algorithm was not generally known for several years after that (in their classic book published in 1962, Ford and Fulkerson [25] cite several inferior algorithms for this problem). In a recent communication concerning the shortest-path paper, Prof. Douglas McIlroy remarks on the modernity of the algorithm's expression, noting that it "is given nondeterministically, which was distinctly unusual for the time." He continues, "None of the thousands of retellings of the original nugget has been able to improve upon it." Dijkstra also invented a very efficient algorithm for the Minimum Spanning Tree (with a running time which grew as the square of the network size), and he published both algorithms in a single paper [4]. This paper is a Citation Classic in the Web of Science database during the period 1945–2000.

While at the Mathematical Center, Dijkstra worked on the very important "real-time interrupt" problem; this became the topic of his Ph.D. thesis [3], which he completed under van Wijngaarden's supervision in Amsterdam in1959. The problem is ubiquitous in the design of modern operating systems. It arises when several computing devices operate asynchronously, and one device may affect the behavior of another by interrupting the latter's execution sequence. This was his first experience with non-determinacy, i.e., the possibility that the results of a computation need not be identical in two different runs. Several computer manufacturers of the day were facing the same problem in systems where the Central Processing Unit (CPU) and the peripheral devices operated nearly asynchronously, but they had not approached the problem with the same rigor that Dijkstra had applied to it. In particular, his doctoral thesis [3] discussed buffering techniques for communications among devices whose speeds differed by several orders of magnitude. Nondeterminacy was later formalized by Dijkstra in a landmark paper [11] in 1975.

Programming as a professional activity was poorly understood in those years. In Dijkstra's own words:

> What about the poor programmer? Well, to tell the honest truth: he was hardly noticed. For one thing, the first machines were so bulky that you could hardly move them and besides that, they required such extensive maintenance that it was quite natural that the place where people tried to use the machine was the same laboratory where the machine had been developed. Secondly, his somewhat invisible work was without any glamour: you could show the machine to visitors and that was several orders of magnitude more spectacular than some sheets of coding. But most important of all, the programmer himself had a very modest view of his own work: his work derived all its significance from the existence of that wonderful machine. Because that was a unique machine, he knew only too well that his program

had only local significance and also, because it was patently obvious that this machine would have a limited life-time, he knew that very little of his work would have a lasting value. Finally, there is yet another circumstance that had a profound influence on the programmer's attitude towards his work: on the one hand, besides being unreliable, his machine was usually too slow and its memory was usually too small, i.e., he was faced with a pinching shoe, while on the other hand its usually somewhat queer order code would cater for the most unexpected constructions. And in those days many a clever programmer derived an immense intellectual satisfaction from the cunning tricks by means of which he contrived to squeeze the impossible into the constraints of his equipment. [9]

At the Mathematical Center, Dijkstra and his colleague J. A. Zonneveld developed a compiler for the programming language Algol-60; it had a profound influence on his later thinking on programming as a scientific activity. Algol-60 was a high-level programming language, designed by an international committee in 1960. Earlier, the programming language Fortran had shown that programmers can be more productive by programming in a high-level language, and that the efficiency of execution need not suffer as a result of using such a language. Algol-60 was an effort at a more systematic design of a programming language. Its official report included several great innovations, notably a formal method for the description of syntax (currently known as the Backus Naur Form, named after two members of the design committee) and the explicit introduction of recursion. Dijkstra appreciated the importance of recursion, and was instrumental in getting it incorporated into the definition of Algol-60. He was probably the first to introduce the notion of a "stack" for translating recursive programs, reporting this seminal work in a short article [5]. He and Zonneveld had completed the implementation of the first Algol-60 compiler by August 1960, more than a year before a compiler was produced by another group. The terms "vector" (for a sequence of consecutive locations in memory) and "stack" have now entered the Oxford English Dictionary in a computing context and are attributed to Dijkstra.

Dijkstra often credits the publication of the Algol-60 report as the moment of birth of Computer Science as a discipline. In coding its compiler, he saw first-hand how systematic design can help tame a mass of intricate details into a set of separable concerns. The prevailing ideas about programming sound quaint today; as he recalls,

Two opinions about programming date from those days. ... The one opinion was that a really competent programmer should be puzzle-minded and very fond of clever tricks; the other opinion was that programming was nothing more than optimizing the efficiency of the computational process, in one direction or the other.

The latter opinion was the result of the frequent circumstance that, indeed, the available equipment was a painfully pinching shoe, and in those days one often encountered the naive expectation that, once more powerful machines were available, programming would no longer be a problem, for then the struggle to push the machine to its limits would no longer be necessary and

that was all programming was about, wasn't it? But in the next decades something completely different happened: more powerful machines became available, not just an order of magnitude more powerful, even several orders of magnitude more powerful. But instead of finding ourselves in the state of eternal bliss of all programming problems solved, we found ourselves up to our necks in the software crisis! How come?"[9]

2 Eindhoven University of Technology

Dijkstra was appointed Professor of Mathematics at the Eindhoven University of Technology in 1962. The university did not have a separate computer science department (no universities then did) and the culture of the mathematics department did not particularly suit him. Dijkstra tried to build a group of computer scientists who could collaborate on solving problems. This was an unusual model of research for the Mathematics Department. In spite of his colleagues' reservations , he built the THE operating system (named for the university, then known as Technische Hogeschool te Eindhoven), which has influenced the designs of all subsequent operating systems. Dijkstra remarks in a later note: "It was the first operating system conceived (or partitioned) as a number of loosely coupled, explicitly synchronized, co-operating sequential processes, a structure that made proofs of absence of the danger of deadlock and proofs of other correctness properties feasible."[19] It introduced a number of design principles which have become part of the working vocabulary of every professional programmer: levels of abstraction, programming in layers, the semaphore, and co-operating sequential processes. His original paper on the THE operating system was reprinted in the 25th Anniversary issue of *Communications of the ACM*, in January 1983. By way of introduction, the Editor-in-Chief says, "This project initiated a long line of research in multilevel systems architecture — a line that continues to the present day because hierarchical modularity is a powerful approach to organizing large systems."

He had earlier formulated the Mutual Exclusion Problem [6], which was also reprinted in the same 25th Anniversary issue of *Communications of the ACM*, in January 1983. It is introduced with this explanation of its significance:

> This short paper marks the beginning of a long era of interest in expressing the synchronization of concurrent processors using ordinary programming languages. This paper considers the problem of implementing an indivisible operation by mutual exclusion of critical sections of code. Later papers by Dijkstra introduced the concepts of semaphores; one of the motivations of semaphores was reducing the complexity of the programming required to implement mutual exclusion...

In 1966, Dijkstra published a very short letter to the editor in *Communications of ACM*, titled "Go To statement considered harmful"[7]. He argued that the programming statement GO TO, found in many high-level programming languages, is a major source of errors, and should therefore be eliminated. This letter caused a giant commotion in the computing community, with combatants

taking positions on all sides of the issue. Some went to the length of equating good programming with the elimination of GO TO. Dijkstra was understandably upset with the ideological tone of the debate and lamented that there are some who believe that a simple syntactic trick will solve all programming problems. He refused to mention the debate, or even the GO TO statement, in his seminal article, "Notes on Structured Programming"[1]. The debate has long since died down; programming languages provide alternatives to the GO TO, few programmers today use it liberally, and most never use it at all.

Dijkstra formulated some of his early ideas about programming as a mathematical discipline starting around the time of publication of this letter. He continued to point out that software productivity is closely related to rigor in design, which eliminates software bugs at an early stage. He was particularly impressed by the dimension of the software problem when he attended the famous 1968 NATO Conference on Software Engineering, which was the first conference devoted to the problem. He became convinced that programming methodology should become a scientific discipline, and he decided to study how to avoid complexity in software designs.

He sent "Notes on Structured Programming", later to be one of his seminal works, to a few friends soliciting their comments. This note became a sensation, and major corporations initiated programs based on his ideas to integrate rigorous practices into their programming projects. Dijkstra had advocated certain design principles, which have now become completely accepted in the computer science community: Large systems should be constructed out of many smaller components; each component should be defined only by its interface and not by its implementation; smaller components may be designed following a similar process of decomposition, thus leading to a top-down style of design; the design should start by enumerating the "separable concerns" and by considering each group of concerns independently of the others; and mathematical logic is and must be the basis for software design. This work has had far-reaching impact on all areas of computer science, from teaching of the very first course in programming to the designs of complex software. Mathematical analysis of designs and specifications have become central activities in computer science research.

In 1972 Dijkstra won the ACM Turing Award, the most prestigious scientific award in Computer Science. His acceptance speech for this award, titled the "The humble programmer", includes a vast number of observations on the evolution of programming as a discipline and prescriptions for its continued growth; it is must-reading for any aspiring computer scientist. In it he says:

> The vision is that, well before the seventies have run to completion, we shall be able to design and implement the kind of systems that are now straining our programming ability, at the expense of only a few percent in man-years of what they cost us now, and that besides that, these systems will be virtually free of bugs. These two improvements go hand in hand. In the latter respect software seems to be different from many other products, where as a rule higher quality implies higher price. Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result the programming process will become cheaper. If

> you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with. In other words: both goals point to the same change.
>
> Such a drastic change in such a short period of time would be a revolution, and to all persons that base their expectations for the future on smooth extrapolation of the recent past —appealing to some unwritten laws of social and cultural inertia— the chance that this drastic change will take place must seem negligible. But we all know that sometimes revolutions do take place! And what are the chances for this one? [9]

Dijkstra goes on to describe a number of economic reasons why this change must necessarily take place, and suggests how it can be accomplished. In retrospect, his expectation of a revolution happening by the end of the seventies was too optimistic. Ironically, the revolution's failure to occur even by the end of the millennium is due in part to Dijkstra's own contributions. Sir Tony Hoare has observed in a private communication,

> As Dijkstra predicted, the development in the power of computers and their widespread distribution have led to a vast increase in programmers' ambition and in the size of the programs they write. This software is still written in inadequate languages, and is seriously afflicted by the curse of compatibility. And it is still far from reliable in any absolute sense. Nevertheless, the increase in the size of software has not been brought to a halt by the predicted accumulation of errors in its many parts. The viability of large-scale software development is due largely to improvements in programming languages, in structured programming practices, in the hierarchical structuring of interfaces, and in the numerous checks and warnings issued by compilers and other modern program analysis tools. All of these improvements are based (often directly) on the same programming theories, ideals, and disciplines propounded by Dijkstra. And even now, the only clear way forward is to follow still more closely the strict mathematical disciplines that Dijkstra discovered, developed, and ably taught us.

3   Burroughs Corporation

Dijkstra joined Burroughs Corporation as its Research Fellow in August 1973. His duties consisted of visiting some of the company's research centers a few times a year and carrying on his own research, which he did in the smallest Burroughs research facility, namely, his study on the second floor of his house in Nuenen. He was already very famous by that time, and he received a large number of invitations to lecture throughout the world. He used these visits to interact with other computer scientists, mentor younger scientists, and sharpen his skills as an English speaker. He also established the Eindhoven Tuesday Afternoon Club (ETAC), a group of researchers who met every Tuesday afternoon in the campus of the university to work on a scientific problem or discuss a recently published paper.

One of his significant contributions from this period is the development of a theory of nondeterminacy [11], a concept outside traditional mathematics. It is often believed that since computers behave predictably, the results of their executions are deterministic: We obtain the same result no matter how many times we run a program. Dijkstra was the first to observe that nondeterminacy is central in computations that involve asynchronous interacting components; two runs could produce significantly different results. Additionally, nondeterminacy could be used as an effective tool in simplifying program design and reasoning about programs, even when no asynchrony is involved.

His earlier work with the THE multiprogramming system had prepared him to undertake an abstract and theoretical study of nondeterminacy. In connection with nondeterminacy, he formulated the basic principle of *fairness* (though he did not coin the term): In an asynchronous system of components, it can be assumed that each component proceeds at a non-zero but finite speed, but no assumption about the exact speed can be (or ought to be) made in the software design. This design principle is the key to building systems in which some of the components could later be replaced by faster (or slower) components without affecting the correctness of the design. He was very proud of this notion, as an abstraction that retains the relevant properties and discards the irrelevant. He recalled that his colleagues were horrified that his design of the THE multiprogramming system did not take into account the differing speeds of the peripheral devices, such as the printer and the punching machine; he had responded that the punching machine also makes more noise, and he was not about to take that into consideration in his software design either.

His other major contribution during this period was the design of "predicate transformers" as a tool for defining program semantics and as a basis for derivations of programs. His ideas refined the earlier ideas of C.A.R. Hoare (now Sir Tony Hoare) for an axiomatic basis of computer programming. He expounded these ideas along with nondeterminacy in the book, *A Discipline of Programming* [12], which has been identified as a "Citations Classic" by the Science Citation Index.

The Burroughs years saw him at his most prolific in output of research articles. He wrote nearly 500 documents in the EWD series (described below), most of them technical reports, for private circulation within a select group. In fact, many of his "small" ideas, such as *deadlock* [8] (evocatively called *deadly embrace* by Dijkstra), on-the-fly-garbage-collection [14], and self-stabilizing systems [10] have started new subareas of computer science.

4 Austin and The University of Texas

Dijkstra accepted the Schlumberger Centennial Chair in the Computer Science Department at the University of Texas at Austin in 1984. He had been visiting the Burroughs Research Center in Austin since the late 1970s and had become quite familiar with the Department of Computer Science and its faculty. As he says in an autobiographical note, "... when the University of Texas at Austin offered me

the Schlumberger Centennial Chair in Computer Sciences, the offer was actually welcome. The offer was not so surprising, nor my acceptance, for I knew Austin, I knew UT and they knew me."[19]

Dijkstra remained prolific in research during his years at Austin. He had embarked on a long-term project on "Streamlining Mathematical Arguments". He explains this area of research:

> As a matter of fact, the challenges of designing high-quality programs and of designing high-quality proofs are very similar, so similar that I am no longer able to distinguish between the two: I see no meaningful difference between programming methodology and mathematical methodology in general. The long and short of it is that the computer's ubiquity has made the ability to apply mathematical method[s] more important than ever. [20]

While at Austin, he co-authored a book on predicate calculus [16], in which he advocated a "calculational proof style" for mathematical arguments: One proves a theorem by calculating that a formula's value is *true*. This proof style is beginning to have considerable impact. Dijkstra continued applying his method in a number of diverse areas: coordinate geometry, linear algebra, graph theory, designs of sequential and distributed programs, and many others.

The years in Austin saw Dijkstra at his best as a teacher and mentor for a generation of students, both undergraduate and graduate. From his days in the Eindhoven University of Technology, he had thought deeply about how computer science should be taught. He had sharpened his thinking on education over a number of years, and Austin provided him the opportunity for trying out his ideas. He enjoyed the experience, appreciating "... brilliant students who made it a challenge and a privilege to lecture for them"[22]. He urged universities not to shrink from the challenge of teaching radical novelties:

> Teaching to unsuspecting youngsters the effective use of formal methods is one of the joys of life because it is so extremely rewarding. Within a few months, they find their way in a new world with a justified degree of confidence that is radically novel for them; within a few months, their concept of intellectual culture has acquired a radically novel dimension. To my taste and style that is what education is about. Universities should not be afraid of teaching radical novelties; on the contrary, it is their calling to welcome the opportunity to do so. Their willingness to do so is our main safeguard against dictatorships, be they of the proletariat, of the scientific establishment, or of the corporate elite. [17]

His approach to teaching was unconventional. At the beginning of each semester he would take a photo of each of the students, in order to memorize their names. He never followed a text book, with the possible exception of his own [16] while it was under preparation. He never used overhead projectors, instead writing on a blackboard. He invited the students to suggest ideas, which he then explored, or refused to explore because they violated some of his tenets. He assigned challenging homework problems, and would study his students' solutions thoroughly; often, his comments were longer than the text to which the comments applied. He conducted his final examinations orally, over a whole week. Each

student was examined in Dijkstra's office or home, and an exam lasted several hours. At the end the student was offered a beer (provided it was age-wise and place-wise legal), and Dijkstra would give the student a signed copy of the photo he had taken at the beginning of the semester, and chat with him or her about the educational experience. For many students, he defined the notion of a stimulating professor; for all those contemplating a career in education, he represented the ideal, and many have attributed their appreciation of computer science to his influence.

He took great pains to educate a select group of graduate students in the Austin Tuesday Afternoon Club (known as ATAC), which was modeled after the similar group, ETAC, which he had established earlier in Eindhoven. This is where explorations of research ideas, writing and presentation styles, and critical reading of scientific articles were undertaken. Most of these privileged students ultimately developed a crisp style of thinking and writing.

Dijkstra saw teaching not just as a required activity but as a serious research endeavor. He often mentioned that a major goal of research should be to create teachable material.

> For me, the first challenge for computing science is to discover how to maintain order in a finite, but very large, discrete universe that is intricately intertwined. And a second, but not less important challenge is how to mould what you have achieved in solving the first problem, into a teachable discipline: it does not suffice to hone your own intellect (that will join you in your grave), you must teach others how to hone theirs. The more you concentrate on these two challenges, the clearer you will see that they are only two sides of the same coin: teaching yourself is discovering what is teachable." [15]

Dijkstra enjoyed the natural beauty of Austin and the surrounding hill country. He and his wife had a fondness for exploring state and national parks in their Volkswagen bus, dubbed the Touring Machine, in which he wrote many of his technical papers. They had a constant stream of visitors in their home, for whom he enjoyed playing Mozart on his Bösendorfer piano.

Although Dijkstra's favorite composer was Mozart, his musical tastes ranged as far back as Tartini and Corelli, and in the last couple of decades he developed an enthusiasm for Dvorak. Most of the music he liked was instrumental; he was suspicious of singing, and especially of the female voice (one notable exception was Kathleen Ferrier). He had no use for opera (for him music could not be improved by the addition of a story and scenery), and Wagner and Tschaikovsky he detested wholeheartedly.

A longtime supporter of Austin's classical music radio station, KMFA, Dijkstra did not always agree with the station's selections, and he had his favorites among the station's announcers (he attached considerable importance to clarity of diction). One of KMFA's business supporters was Mathematics, Inc., which devotees of the EWD series recognized as Dijkstra's long-running spoof of the software industry.

On the occasion of Dijkstra's 60th birthday in 1990, the Department of Computer Sciences organized a two-day seminar in his honor. Speakers came from all over the US and Europe, and a group of computer scientists contributed research articles which were edited into a book [24].

Dijkstra retired from active teaching in November 1999. To mark the occasion and to celebrate his forty-plus years of seminal contributions to computing science, the Department of Computer Sciences organized a symposium, which took place on his 70th birthday in May 2000. The symposium was attended by a large number of prominent computer scientists as well as present and former students. Dijkstra gave a farewell lecture in which he fondly recalled his years at UT:

> A second reason for considering myself fortunate was my transfer to UT Austin, where the Department of Computer Sciences was most accommodating. As specific examples I mention the Year of Programming, initiated by K. M. Chandy and J. Misra, but mostly organized by H. Richards, and the sabbatical years that W. H. J. Feijen, W. H. Hesselink, C. A. R. Hoare and D. Gries spent at the Department, visits that greatly enriched both our lives in all sorts of ways. Add to that helpful staff, whose continued assistance I greatly appreciated, brilliant students who made it a challenge and a privilege to lecture for them, and the clear, blue sky of Texas, for which I had been waiting for 50 years. [22]

5 Awards and Honors

Dijkstra received almost all of the major awards for which he was eligible. Among these are the ACM Turing Award in 1972; the introduction given at the awards ceremony is a tribute worth repeating:

> The working vocabulary of programmers is studded with words originated or forcefully promulgated by E. W. Dijkstra: display, deadly embrace, semaphore, go-to-less programming, structured programming. But his influence on programming is more pervasive than any glossary can possibly indicate. The precious gift that this Turing Award acknowledges is Dijkstra's style: his approach to programming as a high, intellectual challenge; his eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness; and his illuminating perception of problems at the foundations of program design. He has published about a dozen papers, both technical and reflective, among which are especially to be noted his philosophical address at IFIP, his already classic papers on cooperating sequential processes, and his memorable indictment of the go-to statement. An influential series of letters by Dijkstra have recently surfaced as a polished monograph on the art of composing programs. We have come to value good programs in much the same way as we value good literature. And at the center of this movement, creating and reflecting patterns no less beautiful than useful, stands E. W. Dijkstra.

Among his other honors and awards are

> Member of the Royal Netherlands Academy of Arts and Sciences (1971)
>
> Distinguished Fellow of the British Computer Society (1971)
>
> AFIPS Harry Goode Memorial Award (1974),
>
> Foreign Honorary member of the American Academy of Arts and Sciences (1975)
>
> Doctor of Science Honoris Causa from the Queen's University of Belfast (1976)
>
> Computer Pioneer Award from the IEEE Computer Society (1982)
>
> ACM/SIGCSE Award for outstanding contributions to computer science education (1989)
>
> ACM Fellow(1994)
>
> Honorary doctorate from the Athens University, Greece (2001).
>
> ACM Influential Paper Award for his paper, "Self-stabilizing systems in spite of distributed control"[10] (2002).

In 2002, the C&C Foundation of Japan recognized Dijkstra "for his pioneering contributions to the establishment of the scientific basis for computer software through creative research in basic software theory, algorithm theory, structured programming, and semaphores." Dijkstra was alive to receive notice of the award, but it was accepted by his family in an award ceremony after his death.

6 Written Works

As in everything else, Dijkstra thought deeply about the problems of communication through writing.

> At a given moment, the concept of polite mathematics emerged, the underlying idea of which is that, even if you have only 60 readers, it pays to spend an hour if by doing so you can save your average reader a minute. By inventing an idealized "average reader", we could translate most of the lofty human goal of politeness into more or less formal criteria we could apply to our texts. [23]

His refusal to condescend to his audience was manifested in the extraordinary care he took as the author or co-author of nine books, twelve book chapters, forty journal articles, thirty-four contributions to conference proceedings, and twenty-two other publications. Nor do these statistics do justice to his vast writings, many of which were written as a series, called EWDs, for private circulation. The articles in the series, which include scientific reports, trip reports and essays, reached number 1318 at the time of Dijkstra's death. Many of his published works first appeared as EWDs. The dissemination of these manuscripts represents a unique form of publication via an informal distribution tree. Each EWD manuscript went to a small group of people, each of whom acted as a new distribution point to send copies to additional people, all over the world. The

Department of Computer Sciences at The University of Texas at Austin has digitally scanned these writings, along with other writings of Dijkstra, and makes them available at `http://www.cs.utexas.edu/users/EWD/`.

Almost all articles in this series appearing after 1972 are hand-written. Having invented much of the technology of software, Dijkstra eschewed the use of computers in his own work for many decades. Even after he succumbed to his UT colleagues' encouragement and acquired a Macintosh computer, he used it only for e-mail and for browsing the World Wide Web. He had no use for word processors, believing that one should be able to write a letter or article without rough drafts, rewriting, or any significant editing. He would work it all out in his head before putting pen to paper, and once mentioned that when he was a physics student he would solve his homework problems in his head while walking the dark streets of Leyden. He preferred the spontaneous Mozart to the laborious Beethoven. His editing of his own work was so light that he could get by with strips of paper and paste.

Dijkstra's favorite writing instrument was the Montblanc Meisterstück fountain pen. He repeatedly tried other pens, but none ever displaced the Montblanc. How many Montblancs he possessed over the years no one knows, but at any one time he had perhaps half a dozen. Some lived in a glass-topped wooden case on his desk, while he kept a couple in a leather wallet in his shoulder bag. And often one was in the shop for maintenance or repair.

He also experimented with inks. He was acutely sensitive to the interaction between ink and paper, and carried out long-term tests of inks' resistance to fading in direct sunlight.

Dijkstra took particular care with his handwriting, and over the years it improved. One winter in Nuenen he slipped on some ice, fell, and broke his right wrist. Rather than take a few weeks off from writing, he trained himself to write with his left hand. After his right hand became operational again, he would make sure to spend some time each week writing left-handed to maintain his ambidexterity.

7 Prophet and Sage

Dijkstra remained till the end a prophet and sage, warning his colleagues about the dangers of mixing scientific questions with political and economic considerations. Throughout his careeer he would rebuke powerful institutions when he felt that scientific principles were being abandoned.

He wrote a large number of essays on the nature of computer science research and education, on its relationship to the industry and society, and even on the manner in which scientific articles should be written and presented. He spoke out against the culture of "write-only journals" and "speak-only conferences" which ignore the difficulties faced by the intended audience.

He was often severe with prominent computer scientists for ideas that he felt were socially pleasing but scientifically unsound. One computer scientist, on

reading a Dijkstra trip report, said, "Dijkstra is right, but you don't say such things." Dijkstra said them.

One of Dijkstra's most acerbic essays is his rebuttal, "A Political Pamphlet from the Middle Ages"[13], to an article by three well-known computer scientists, Richard A. DeMillo, Richard J. Lipton and Alan J. Perlis [2]. His rebuttal begins:

> This note concerns a very ugly paper [...]. Its authors seem to claim that trying to prove the correctness of programs is a futile effort and, therefore, a bad idea. To quote from the opening sentence: "program verification [...] is bound to fail in its primary purpose: to dramatically increase one's confidence in the correct functioning of a particular piece of software". As rendered above, this statement is obviously wrong ....

And, Dijkstra says later:

> ... [they] accuse without substantiation that "a large segment of the computer science community" accepts this nonsense as a fact of life, and then try to sell the great message that the computer science community has been misguided. That is what I call the style of a political pamphlet.

The following quote expresses his views on the shortcomings of the American style of computer science education:

> ... prevailing industrial attitude exerts a strong pressure on the university not to indulge in such hobbies as scientific education, but to confine itself to vocational training of some sort or another. … Deprived of what is generally considered computing's core challenge, American Computing Science is the big loser, and we can not blame the universities, for when the industry most in need of their scientific assistance is unable to face that they are in a high-technology business, even the best university is powerless. Universities should be more enlightened than their environments, and they can [be], but not much (that is, not openly). In the current political climate, it is unlikely that things will improve soon. [20]

His paper "On the Cruelty of Really Teaching Computing Science"[17] was a strong indictment of the current educational standards and industrial practices:

> Needless to say, this vision of what computing science is about is not universally applauded. On the contrary, it has met widespread —and sometimes even violent— opposition from all sorts of directions. I mention as examples
>
> 0. the mathematical guild, which would rather continue to believe that the Dream of Leibniz is an unrealistic illusion
> 1. the business community, which, having been sold the idea that computers would make life easier, is mentally unprepared to accept that they only solve the easier problems at the price of creating much harder ones
> 2. the subculture of the compulsive programmer, whose ethics prescribe that one silly idea and a month of frantic coding should suffice to make him a life-long millionaire
> 3. computer engineering, which would rather continue to act as if

it is all only a matter of higher bit rates and more flops per
second

4. the military, which is now totally absorbed in the business of
using computers to mutate billion-dollar budgets into the
illusion of automatic safety

5. all soft sciences for which computing now acts as some sort of
interdisciplinary haven

6. the educational business that feels that if it has to teach formal
mathematics to CS students, it may as well close its schools.

And, from the same article:

I must draw attention to the astonishing readiness with which the suggestion
has been accepted that the pains of software production are largely due to a
lack of appropriate "programming tools". (The telling "programmer's
workbench" was soon to follow.) Again, the shallowness of the underlying
analogy is worthy of the Middle Ages. Confrontations with insipid "tools" of
the "algorithm-animation" variety has not mellowed my judgement: on the
contrary, it has confirmed my initial suspicion that we are primarily dealing
with yet another dimension of the snake-oil business.

Dijkstra had strong (and, some might say, unfounded) opinions about Artificial
Intelligence:

Finally, to correct the possible impression that the inability to face radical
novelty is confined to the industrial world, let me offer you an explanation of
the continuing popularity of artificial intelligence. You would expect people
to feel threatened by the "giant brains or machines that think". In fact, the
frightening computer becomes less frightening if it is used only to simulate a
familiar noncomputer. I am sure that this explanation will remain controvers-
ial for quite some time for artificial intelligence, as a mimicking of the human
mind, prefers to view itself as being at the front line, whereas my explanation
relegates it to the rearguard. (The effort of using machines to mimic the
human mind has always struck me as rather silly. I would rather use them to
mimic something better.) [17]

He was not kind to the practitioners in his own discipline:

It is time to unmask the computing community as a Secret Society for the
Creation and Preservation of Artificial Complexity. And then we have
software engineers, who only mention formal methods in order to throw
suspicion on them. In short, we should not expect too much support from the
computing community at large. [21]

He was not kinder to the mathematicians:

And from the mathematical community I have learned not to expect too much
support either, as informality is the hallmark of the Mathematical Guild,
whose members —like poor programmers— derive their intellectual
excitement from not quite knowing what they are doing and prefer to be
thrilled by the marvel of the human mind (in particular their own ones). [21]

In the end he was an optimist. He believed in the generations of computer

scientists to come, in the power of education, and in the importance of the educational enterprise for society. He concludes, "In the next fifty years, Mathematics will emerge as The Art and Science of Effective Formal Reasoning, and we shall derive our intellectual excitement from learning How to Let the Symbols Do the Work." [21]

8 Some Memorable Quotations

Dijkstra was famous for his wit, eloquence, and way with words, such as in his remark, "The question of whether computers can think is like the question of whether submarines can swim"; his advice to a promising researcher, who asked how to select a topic for research, "Do only what only you can do"; and his remark in his Turing Award acceptance speech,

> In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind. [9]

Compiled below are a few of Dijkstra's other memorable epigrams.

> Computer Science is no more about computers than astronomy is about telescopes.

> A formula is worth a thousand pictures.

> Always design your programs as a member of a whole family of programs, including those that are likely to succeed it.

> Separate Concerns.

> A Programming Language is a tool that has profound influence on our thinking habits.

> The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague. [9]

> Progress is possible only if we train ourselves to think about programs without thinking of them as pieces of executable code.

> Program testing can at best show the presence of errors but never their absence.

> ... if 10 years from now, when you are doing something quick and dirty, you suddenly visualize that I am looking over your shoulders and say to yourself, Dijkstra would not have liked this, well that would be enough immortality for me.

> As a matter of fact the still often repeated requirement that axioms should be self evident strikes me as a medieval relic: to the extent that they take philosophy seriously, it is impossible for me to take logicians seriously. (Again this may be a cultural difference: it seems there are societies in which philosophers still have some intellectual standing.)

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense.

Being abstract is something profoundly different from being vague.

The problems of the real world are those that remain when you ignore their known solutions.

The prisoner falls in love with his chains. (in reference to programmers using inadequate tools)

I pray daily that more of my fellow programmers may find the means of freeing themselves from the curse of compatibility. [9]

Brainpower is by far our scarcest resource. [9]

My daughter and I taking a shower with equal frequency is a frightening thought for both of us. (February 21,1984; commenting on a mutual exclusion algorithm in which the two components are alternately granted access to the critical section.)

Nothing is as expensive as making mistakes.

If you carefully read its literature and analyze what its devotees actually do, you will discover that software engineering has accepted as its charter, "How to program if you cannot." [17]

We must give industry not what it wants, but what it needs.

Waiting is a very funny activity: you can't wait twice as fast.

Do not try to change the world. Give the world the opportunity to change itself.

So-called natural language is wonderful for the purposes it was created for, such as to be rude in, to tell jokes in, to cheat or to make love in (and Theorists of Literary Criticism can even be content-free in it), but it is hopelessly inadequate when we have to deal unambiguously with situations of great intricacy, situations which unavoidably arise in such activities as legislation, arbitration, mathematics or programming. (foreword to *Teaching and Learning Formal Methods*, edited by C. N. Dean and M. G. Hinchey, Academic Press, 1996)

The traditional mathematician recognizes and appreciates mathematical elegance when he sees it. I propose to go one step further, and to consider elegance an essential ingredient of mathematics: if it is clumsy, it is not mathematics. [16]

Don't compete with me: firstly, I have more experience, and secondly, I have chosen the weapons. (During first lecture in Capita Selecta, August 29, 1996)

Maintaining a large range of agilities mental and physical requires regular exercise [..]. That is why the capable are always busy. (Lecture, Capita Selecta, October 10, 1996)

Mathematicians are like managers; they want improvement without change. (During a meeting of the Austin Tuesday Afternoon Club, Fall 1996)

... I had already come to the conclusion that in the practice of computing, where we have so much latitude for making a mess of it, mathematical elegance is not a dispensable luxury, but a matter of life and death. [15]

_____
Larry R. Faulkner, President
The University of Texas at Austin

_____
John R. Durbin, Secretary
The General Faculty

This Memorial Resolution was prepared by a special committee consisting of Professor Jayadev Misra (chair) and Dr. Hamilton Richards, who acknowledge with gratitude the comments and contributions of Prof. Robert S. Boyer (The University of Texas at Austin), Prof. David Gries (Cornell University), Prof. M. Douglas McIlroy (Dartmouth College), and Prof. Sir Tony Hoare, FRS (Oxford University and Microsoft Research).

References

1. O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare. *Structured Programming*. Academic Press, 1972.

2. Richard A. DeMillo, Richard J. Lipton, and Alan J. Perlis. Social processes and proofs of theorems and programs. *Communications of the ACM 22*, 5 (1979), 271–280.

3. Edsger W. Dijkstra. Communication with an Automatic Computer. PhD thesis, University of Amsterdam, 1959.

4. Edsger W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik 1* (1959), 83–89.

5. Edsger W. Dijkstra. Recursive programming. *Numerische Mathematik 2* (1960), 312–318.

6. Edsger W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM 8*, 9 (1965), 569.

7. Edsger W. Dijkstra. Go To statement considered harmful. *Communications of the ACM 11*, 3 (1968), 147–148. Letter to the Editor.

8. Edsger W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*, 43–112. Academic Press, 1968.

9. Edsger W. Dijkstra. The humble programmer. *Communications of the ACM 15*, 10 (1972), 859–866. Turing Award lecture. Also `http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.PDF`.

10. Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM 17*, 11 (1974), 643–644.

11. Edsger W. Dijkstra. Guarded commands, nondeterminacy, and the formal derivation of programs. *Communications of the ACM 18*, 8 (1975), 453–457.

12. Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

13. Edsger W. Dijkstra. A political pamphlet from the Middle Ages, EWD638. `http://www.cs.utexas.edu/users/EWD/ewd06xx/EWD638.PDF`, 1977. Circulated privately.

14. Edsger W. Dijkstra et al. On-the-fly garbage collection: An exercise in cooperation. *Communications of the ACM 21*, 11 (1978), 966–975 .

15. Edsger W. Dijkstra. My hopes of computing science. In *Proc. 4th Int. Conf. on Software Engineering*, Munich, 442–448. IEEE, September 1979. Also `http://www.cs.utexas.edu/users/EWD/ewd07xx/EWD709.PDF`.

16. Edsger W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1989.

17. Edsger W. Dijkstra. On the cruelty of really teaching computing science. *Communications of the ACM 32*, 1 (1989), 398–414. Also `http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF`.

18. Edsger W. Dijkstra. On the economy of doing mathematics, EWD 1130. `http://www.cs.utexas.edu/users/EWD/ewd11xx/EWD1130.PDF`, 1992. Circulated privately.

19. Edsger W. Dijkstra. From my life, EWD 1166. `http://www.cs.utexas.edu/users/EWD/ewd11xx/EWD1166.PDF`, 1993. Circulated privately.

20. Edsger W. Dijkstra. Why American Computing Science seems incurable, EWD 1209. `http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1209.PDF`, 1995. Circulated privately.

21. Edsger W. Dijkstra. The next fifty years, EWD1243a. `http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1243a.PDF`, 1996. Circulated privately.

22. Edsger W. Dijkstra. Under the spell of Leibniz's Dream, EWD 1298. *Information Processing Letters 77*, 2–4 (2001), 53–61. Also `http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1298.PDF` .

23. Edsger W. Dijkstra. The notational conventions I adopted, and why, EWD 1300. *Formal Aspects of Computing 14*, 2 (2002), 99–107. Also `http://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1300.PDF`, 2000.

24. W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and Jayadev Misra (editors). *Beauty is our Business*. Texts and Monographs in Computer Science. Springer-Verlag, 1990.

25. L.R Ford, Jr. and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.