Under the spell of Leibniz's Dream

Recently, when it dawned upon an undergraduate student that I had been introduced to automatic computing in 1951, he reacted with "Wow! Then even my parents were not born yet.", thus driving home the message that, indeed, the outstanding feature of my professional career could very well be its length. It has been a long time indeed, and if I had to characterize those almost 50 years, the first characterization that came to my mind was "Half a century of change.". My personal experience of change has, of course, been intensified by my switch from one continent to another, from one civilization to a very different one, thus almost doubling the set of experiences I had to integrate.

You should realize that I grew up in a very simple world. During the German occupation, the manifest lawlessness made that the distinction between the Good Guys and the Bad Guys was no problem at all. Life remained simple for about two decades after the Liberation by the Allied Forces (to whom I shall remain grateful for the rest of my life and beyond), because WWII had left my country in such a complete shambles that everyone agreed that restoration of our infrastructure was the first thing to be done. Before that task had been completed, a devastating flood showed the need for what became known as "the Delta Works". The urgency and scope of these undertakings gave to many endeavours a unique sense of purpose which in retrospect should be viewed as a luxury: we never needed to wonder what to do next. On the contrary, for each of the new machines that were built at the Mathematical Centre in Amsterdam, urgent work was waiting before the machines were operational. For the first X1 I changed one of the first programs because one of the ferrite cores in memory was broken and could store only one value, but the show had to go on!

An important side-effect of the hard times was the creation of a spiritual climate in which the distinction between pure and applied science had vanished: of all the things you could do, you just did the most urgent one, and the development of some urgently needed theory was often the most practical thing to do. It was my privilege to grow up in that spiritual climate, and the conviction that what is theoretically beautiful tends to be eminently useful has been an article of my faith ever since: in the design of sophisticated digital systems, elegance is not a dispensable luxury but a matter of life and death, being a major factor that decides between success and failure.

In 1969, at the NATO Conference on Software Engineering Techniques in Rome, I called the distinction between practical and theoretical people "obsolete, worn out, inadequate and fruitless", angrily adding that I absolutely refused to consider myself as either impractical or not theoretical. But times have changed. These days there is so much obsession with application that, if the University is not careful, external forces, which do make the distinction, will drive a wedge between "theory" and "practice" and may try to banish the "theorists" to a ghetto of separate departments and separate buildings. A simple extrapolation will tell us that in due time the isolated practitioners will have little to apply; this is well-known, but has never prevented the financial mind from killing the goose that lays the golden eggs. The worst thing with institutes explicitly devoted to applied science is that they tend to become institutes of second-rate theory.

But I am rushing ahead. During WWII, the Universities had been closed, when I enrolled in 1948, there was still a lot of catching up to be done, and I remember that Leyden University was a very crowded place. The ideal to extend higher education to a larger fraction of the graduating high school population existed, but for a decade or so it did not really materialize, just for lack of space and lack of facilities. It was only after my graduation that the Universities stopped being the elite institutions that they used to be.

The envisaged growth of the university system was viewed with grave suspicion. I remember how, in 1955, one of my great-aunts felt that starting a new technological university in Eindhoven was a stupid idea, for she did not see how they were ever going the find faculty for it that was sufficiently bright and decent. A decade later, I was on the faculty of that university myself, and I must admit that, indeed, the old lady's misgivings had not been misplaced.

In the 50s, the university population really began to grow, the faculty, the student body and the goals of the university, they all became more diversified and it was <u>not</u> a happy mix. I was regularly reminded of a remark by Antoine de Saint-Exupéry that it is Man's predicament to have to plan for tomorrow in yesterday's language: the outstanding feature of faculty meetings was the general dishonesty, because linguistically we were still pretending to strive for perfection and to conduct a first-rate enterprise while we all knew that mediocrity was taking over.

The students were not happy either. While faculty complained that they could not maintain standards, students were frustrated by demands they were not used to, such as clarity of diction and correctness of spelling and grammar. They also felt cheated: going to University had been presented to them as the privilege of being admitted to an exclusive group, but as a result of their sheer numbers, the group was no longer exclusive.

Around 1968, the time bomb exploded and worldwide the student revolutions broke loose, buildings were occupied and on campus, mob rule reigned. The revolution was successful in the sense that, when the dust had settled, the old university was no more. Under slogans like "Education for the People by the People", maintaining intellectual standards was presented as something immoral and striving for perfection —traditionally the raison d'être of the academic enterprise-- was discarded as "elitist" and hence "undemocratic". For a lecture course on a formal topic like program correctness, the climate had become less than ideal, for the whole preposterous idea that programs should meet formal specifications was clearly only pushed by an older generation that wanted to stay in power and to that end tried to prolong the circumstances in which their better education gave them the upper hand. Political sloganeering knew no limits; half a decade later, an Australian columnist would relate mathematical discipline to fascism and would explain that from Wirth and Dijkstra, coming from Germanic countries, you could expect no better than sympathy for such stiffling rigour, but that he totally failed to understand how the freedom-loving Anglo-Saxon Hoare could join that crowd.

But I digress. Within two student generations, i.e., about 10 years, the revolutionary zeal had petered out, and it looked as if on campus life was back to normal, but that appearance was very misleading. The whole turmoil had caused major shifts in the power structure, in the role of money, and in the intellectual quality of the whole academic enterprise.

I grew up with an advice of my grandfather's: "Devote yourself to your most genuine interests, for then you will become a capable man and a place will be found for you.". We were free not to try to become rich (which for mathematicians was impossible anyhow). When Ria and I had to decide whether we should try to start a family, we felt that we had to choose between children and a car; Ria had a Raleigh, I had a Humber, we decided that they were still excellent bicycles and chose the children. Today's young academics, whose study is viewed as an investment, should envy

us for not having grown up with their financial distractions. The other day I saw a telling symptom of how the business world is encroaching on our peace of mind. For a number of years I now give each semester one lecture to a class of software engineering students, who then write down their comments on my performance, and these are revealing; for instance, they used to refer to my contributions to computer science, but the last time they just referred to my contributions to "the industry". This was a telling but innocent symptom, but there are much more serious ones, such as the retention problem. For some undergraduates the major computer science problem seems to be in which month to drop out so that they can become a billionaire and in the more pragmatic graduate schools it becomes very difficult to attract Ph.D. students because their target audience is just as happy to go to industry directly. About 10 years ago I tried to make up my mind on the question whether Computing Science could save the computer industry, and my conclusion was negative; since then I felt it my duty to try to prevent the computer industry from killing Computing Science, but I doubt that I have been successful....

But, worried as I am by the monetary distortion of our academic value system, I am even more worried by another development over the last fifty years, and that is the general loss of respect for each other, and if that is the price to be paid for the growth of the academic enterprise, we may ask indeed whether that price was not too high. Observing in Leyden our fellow students, we quickly saw that some shouldn't have enrolled, the others soon coalesced into a support group for which mutual trust and respect provided the glue, and it is quite possible that I have learned more from my fellow students than from my professors. Observing our professors, we soon decided that a few appointments had been mistakes, for the others we had the greatest admiration and I remember that we felt proud to be their students. Conversely, our professors treated us with respect: we were treated as the next generation that had to provide their successors.

But things have changed. These days it is quite common that students don't even know the name of the person who is lecturing to them, let alone that they feel proud to be his student! And it is equally common that faculty talk about students, in particular undergraduates, as if they were animals from another planet. Students are no longer seen as part of the solution, but as part of the problem, and textbooks and lecturing techniques have become so condescending that, if I were a student, I would take offence. I don't think that I could stand a lecturer that assumes that my

attention span is no more than 7 minutes or who feels obliged to feed me a cartoon every 5 foils, and subjects me to multiple-choice tests because I am supposed to be functionally illiterate. I invite you to read carefully the catalogues of publishers of mathematical textbooks: obviously, colour math is better than B&W math, most books are recommended for being intuitive instead of formal, for being chatty instead of crisp, for being vague and sloppy instead of rigorous. It is clearly an article of the New Faith that teaching the real thing would be counter-productive: just as our students are supposed to live on junk food, they are supposed to thrive on junk science.

The loss of mutual respect has affected more than just the educational process, it has corroded publication as well. In 1975 I received a letter that objected to the style in which I had written an article for the Communications of the ACM (= Association for Computing Machinery). The complaint was that, by separating my concerns more strictly than usual, I had addressed my intended audience in a style they were not used to. The writer continued with the well-known quotation from P.T.Barnum that "No one ever got broke by underestimating the intelligence of the American people." and urged me to bear that in mind whenever I wrote for the programming community. So, 25 years ago, the rot had already set in; at the end of the century it would lead to an endless series of fat, yellow books titled "Such and such for dummies". Allow me to quote in contrast from "The elements of Style" by Strunk and White, because it reflects a much more inspiring spirit; "No one can write decently who is distrustful of the reader's intelligence, or whose attitude is patronizing.".

This disrespectful, almost contemptuous attitude has not only affected teaching and publishing, it has affected the academic research agenda as well, but in spite of all that, Computing Science was conceived, was born and started to grow up quite successfully.

It definitely was not born yet when, in the early 50s, I got involved in automatic computing. At the time I was being groomed to become a very good theoretical physicist, but in 1955 that training was aborted when I decided to become a programmer instead. Not suffering from excessive modesty, I had what I considered a very good reason: I had concluded that programming presented

a greater challenge than theoretical physics. I traded theoretical physics, a discipline highly respected because it had been good for a number of Dutch Nobel prizes, for what at the time was hardly a profession and certainly was without academic respectability. It was a difficult decision, which I would never have taken without the support of my parents and the encouragement of A.van Wijngaarden, my then boss at the Mathematical Centre in Amsterdam. I am grateful to all three.

Things changed in 1960, when a transatlantic cooperation created the programming language ALGOL 60, an event that my colleague F.E.J.Kruseman Aretz for good reasons has chosen to mark the birth of Computing Science. The reasons were good because ALGOL 60 introduced a handful of deep novelties, all of which have withstood the test of time. Let me review them briefly.

Firstly there is the introduction and use of Backus-Naur-Form (or BNF for short), a formal grammar to define the syntax of the programming language. More than anything else, ALGOL 60's formal definition has made language implementation a topic worthy of academic attention, and this was a crucial factor in turning automatic computing into a viable scientific discipline.

Secondly there was the block structure and the concept of the local variable, with the beautiful synthesis of lexicographic scope with dynamically nested lifetimes. The lexicographic scope led to a unified notation for mathematical quantification, while the nested lifetimes opened the door to the next novelty.

This was recursion. Logicians knew it, LISP had incorporated it, but ALGOL 60 gave it a firm place in imperative programming. These days, that place is quite natural, but at the time it was a shocking novelty because FORTRAN, which did not accommodate recursive definitions, had created the widely held opinion that recursion was too fancy and hence misplaced in a "real" programming language. In defence of the status quo, ALGOL 60 has been blamed for one of its major virtues. A few years later a five-year old son would show me how smoothly the idea of recursion comes to the unspoilt mind. Walking with me in the middle of town he suddenly remarked to me "Daddy, not every boat has a lifeboat, has it?" I said "How come?" "Well, the lifeboat could have a smaller lifeboat, but then that would be without one."

In the fourth place I would like to mention the introduction of the type boolean as a first class citizen. It emerges when we view statements such as "2 + 2 = 4" or "It's Friday" as expressions that have a value, the two possible values being denoted by "true" and "false". These so-called "truth values" were introduced in 1854 by George Boole, but they hardly got citizenship in the world of mathematics. From the fact that, thanks to ALGOL 60, now each programmer is familiar with them, I expect a profound and radical influence on mathematics as a whole, an influence which is only beginning to be felt. This expectation is not based on gazing in a crystal ball but on my mother's inability to really accept the concept. She was a brilliant mathematician, but by definition a generation older, and she could not really accept 3 + 4 = 9 as an expression: for us it is a complicated way of writing "false", for her, 3 + 4 = 9 was just wrong. For her, boolean expressions were statements of fact: she had to interpret them.

Let me try to give you a simple example of what interpretation can lead to. Consider 10 pigeonholes with pigeons in them, and look now at the following two statements.
First statement: When the number of pigeons exceeds 10, at least 1 hole contains at least 2 pigeons.
Second statement: If each hole contains at most 1 pigeon, there are at most 10 pigeons.
The first statement evokes in the usual interpretation a picture of housing shortage, of holes with too many pigeons crammed into them. The second statement evokes a picture of housing luxury, with each pigeon having his own hole , and perhaps even some free holes! Yet in Boolean logic, the two statements are equivalent, they mean the same thing in exactly the same way as "Jack loves Jill" and "Jill is loved by Jack" mean the same thing. In other words, the pictorial interpretations blur the issue by introducing spurious distinctions.

ALGOL 60's fifth novelty was its machine independence. FORTRAN's purpose had been to ease programming for the IBM 709, while now it became the computer's task to execute ALGOL programs, and this was a major paradigm shift. We saw in the difficulty of implementing ALGOL 60 on IBM machines a confirming symptom of their poor design; Alan Perlis, however, judged ALGOL 60 "an inefficient language".

In the decade that followed, I passed a few personal milestones. My programming experience culminated in the ALGOL implementation with J.A.Zonneveld at the Mathematical Centre in Amsterdam and the THE Multiprogramming System with a handful of people at the Technical University Eindhoven. Both projects were at the time very ambitious, we successfully completed them by keeping them as clean as possible, by today's standards both were a mess and both times we knew we were making history. Each evening, for instance, Jaap Zonneveld and I would take our own copy of the documentation home so that the compiler could not get lost in a fire at the Mathematical Centre.

The ALGOL implementation was one of my proudest achievements. Its major significance was that it was done in 8 months at the investment of less than 3 man-years. The rumour was that IBM's FORTRAN implementation had taken 300 man-years and thus had been a project in size way beyond what a university could ever hope to undertake. By requiring only 1 % of that labour, our compiler returned language implementation to the academic world, and that was its political significance. The THE Multiprogramming System was technically significant: it has introduced concepts that pervade computing science to this very day.

The advent of more powerful and --what people sometimes forget-- much more reliable computers invited many ambitious projects at all sorts of places. Some were successful, some were overambitious, all of them drove home the message that such systems could easily get so complicated as to become intellectually unmanageable and, hence, no longer trustworthy. These were serious concerns, which led to a number of observations and conclusions. I mention a few.

(i) When exhaustive testing is impossible --i.e., almost always-- our trust can only be based on proof (be it mechanized or not).

(ii) A program for which it is not clear why we should trust it, is of dubious value.

(iii) A program should be structured in such a way that the argument for its correctness is feasible and not unnecessarily laborious.

(iv) Given the proof, deriving a program justified by it, is much easier than, given the program, constructing a proof justifying it.

Considerations like the above gave rise to my "Notes on Structured Programming (from 1969) and C.A.R.Hoare's "Notes on Data Structuring" (from 1970). It is hard to quantify influence, but I think they have been seminal papers. People don't refer to them anymore, many haven't even read them, but the message has not been forgotten, it has been absorbed. [For your information, the book containing both texts is still in print; for 1999, our worldwide sales were 4 copies.)

Before going on I would like to point out that the recognition of programming as an intellectual challenge was at the time highly controversial.

Firstly, there was the matter of accuracy. A program would consist of, say, a few thousand instructions of half a dozen characters each, and thus would be a text of ten thousand characters, in which not a single typing error was permitted. The competent programmers unavoidably learned the clerical skills needed for the flawless manipulation of such long character strings: they became routine like the surgeon's precautions in the operating room. The misunderstanding this created in the minds of others was that programming was merely a matter of accuracy, and that therefore it should be done by clerically trained personnel, and, if more persons were involved, under proper management. In short, programming was viewed as a perhaps painful but simple, low-level task. Needless to say, this confusion between the score and the composition led to an underestimation of the intellectual challenges programming presents.

Secondly, there was the widely spread belief that, once we had the right, sufficiently powerful programming language, programming would be so easy that everybody could do it. The older ones among us still remember the famous ad in a Datamation issue of 1968, showing in full colour a beaming Susie Mayer, telling the reader that she had solved all her programming problems by switching to PL/I. IBM never published her picture 4 years later.

The forces to play down the difficulty of programming were just too strong. I mentioned the sense of "popular justice" of the democratization movement that felt that everybody should be able to program. Then there was the religion at Xerox PARC that it should all be so natural that toddlers would love to do it. Then there was COBOL's creed that if your programming language was similar

enough to English, even officers would be able to program. A major force was IBM, which wanted to sell hardware and hence was eager to present its machines as solutions rather than as the source of new problems. Finally, the prejudices of management did not allow industrial programming to be viewed as difficult: in the first half of the century, it had become a management goal to organize for the sake of stability the industrial enterprise in such a fashion that it would be as independent as possible of the competence of individual employees, with the result that in the second half of the century the mere suggestion that even industrial programming could require brains was blasphemy. For a while, the "deskilling of the programmer" was a hot topic.

The prevaling attitude was reflected in the creation of two literary figures --admittedly of rather poor literature, but nevertheless of great paralyzing power--, viz. "the average programmer" and "the casual user". Up to these days, academic research in programming methodology has been supposed to respect the severe intellectual limitations of these fictitious morons: consequently any proposal that required any further education of the programming person was out. Academia has suffered, for programming methodology could not flourish as a research topic in a world in which it was considered irrelevant. For many a university, this has been a great loss; here and there the situation might be improving, but it is a slow process.

By the end of the 60s, The University of Oxford had its PRG, i.e., its Programming Research Group, and IFIP, the International Federation for Information Processing, had its Working Group 2.3 on "Programming Methodology". Both flourished and, as time went on, they were joined by the newly founded NATO Summer School in Marktoberdorf, as that became more focussed on the methodological aspects of design and got more and more prestigious in the process. So the word spread, and that was fine, but how slowly it did so is clearly illustrated by the date of the 1st conference on the Mathematics of Program Construction --or MPC for short--: this was organized in the Netherlands by Roland C.Backhouse and Jan L.A. van de Snepscheut in 1989, essentially 20 years after the scientific merit and potential of the topic had been recognized. The next 3 MCP Conferences took place in the 90s, in the United Kingdom, in Germany, and in Sweden, respectively, and if I were an American, I would consider the fact that the first 4 of these conferences all took place in Europe a very alarming symptom indeed.

On the other hand we should be glad that the gospel of design by derivation rather than by trial and error is still preached. For me personally it is very gratiftying to see that the area of application has been extended by people whom I consider in moments of greater vanity as my pupils: I have in mind A.J.Martin in connection with chip design, and W.H.J.Feijen and A.J.M. van Gasteren, who forged the Gries-Owicki theory into a methodology for deriving multi-programs. They all have done me proud.

Back to personal history. After 10 years at the Mathematical Centre in Amsterdam, I went to the Department of Mathematics at the Eindhoven University of Technology, where I stayed from 1962 until 1973. During that decade, Computing Science or "Informatics", as the topic became known on the continent, became a much more substantial discipline than my mathematical colleagues had envisaged or intended and, perhaps not totally without justification, part of the blame for this undesired development was heaped upon me. Life in Eindhoven became a little difficult, but fortunately, Burroughs Corporation offered to rescue me.

I stayed with Burroughs Corporation for 11 wonderful and very productive years, my primary charter being to do my own thing. I had one major disappointment: I had hoped to lead the life of a scientific "man of letters", but the frequency with which my letters were answered was, on the average, too low to maintain a lively interchange, and so I ended up travelling more than I had intended and was comfortable with. But apart from that, they were, as said, wonderful years. It was too good to last, and it did not. After the company had changed its CEO, it rapidly lost interest in science and technology, and the groups with which I had built up my relations disbanded the one after the other. But it was not only Burroughs that changed during that decade, for I changed as well; my mathematical interests widened and eventually I was ready to return to the University Campus, and was fortunate that UT Austin offered me the opportunity to do so.

During my Burroughs years I changed very much as a mathematician. By the end of the 60s, R.W.Floyd and C.A.R.Hoare had shown us how to reason in principle about programs, but it took a few years before their findings penetrated through my skull. In the early 70s we studied Tony Hoare's correctness proof of the routine "FIND" and my friends still remember that at the end of that

study I declared in disgust that "that meaningless ballet of symbols was not my cup of tea"; only gradually, my resistance to that kind of formula manipulation faded, until eventually, I began to love it.

Circumstances forced me to become a very different mathematician. I had invented predicate transformers as a formal tool for programming semantics and they served my purpose, but I lacked at the time the mathematical background needed to understand what I was doing, so much so that I published a paper with a major blunder. I was helped by C.S.Scholten, with whom I had worked since the 50s, and together we provided the mathematical background that had been lacking. In doing so we developed a much more calculational style of doing mathematics than we had ever practised before. I stayed away from where I had blundered, but that hurdle the younger and mathematically better equipped R.M.Dijkstra took in his stride after he had extended our apparatus from predicates to relations.

Let me try to sketch my mathematical change. For centuries, mathematics has been what it says in the dictionary: the abstract science of space, number and quantity. In this conception, mathematics is very much about things, such as triangles, prime numbers, areas, derivatives and 1-1 correspondences. Accordingly, such mathematics is done in a language in which nouns prevail and which each time is tied rather closely to the subject matter at hand. In its orientation towards <u>things</u> it invited pictures and visual representations. This has gone so far that the Greeks introduced, for instance, the hardly needed notion of the "triangular numbers" 1, 3, 6, 10, etc., purely because such numbers of dots admitted triangular arrangements:

But at least since the 18th Century --personally I am aware of Augustus de Morgan and George Boole from Great Britain-- there has emerged another view, representing a shift from the "quod" to the "quo modo", from the "what" to the "how": it views mathematics primarily as the art and science of effective reasoning.

This was refreshing because the methodological flavour gave it a much wider applicability. Take a sophisticated piece of basic software such as a program-

ming language implementation or an operating system; we cannot see them as products engendered by the abstract science of space, number and quantity, but as artefacts they are logically so subtle that the art & science of effective reasoning has certainly something to do with their creation: in this more flexible conception of what mathematics is about, programming is of necessity a mathematical activity. History has shown that the mathematical community needs a constant reminder of this more methodological conception of its trade, as many mathematicians invest so much of their time and energy in the study of a specific area that they tend to identify mathematics with its [always specific] subject matter.

As far as I know, Gottfried Wilhelm Leibniz, who lived from 1646 to 1716, has been the first to tackle effective reasoning as a technical problem. As a youngster of 20 years of age he conceived, possibly inspired by the work of Descartes, a vision of reasoning as applying a calculus. Like modern computing scientists, he invented impressive names for what had still to be inventend, and, for good reasons not overly modest, he called his system no more and no less than "Characteristica Universalis". And again like modern computing scientists, he grossly underestimated the time the project would take: he confidently prophesied that a few well-chosen men could do the job in five years, but the whole undertaking was at the time of such a radical novelty that even the genius of Leibniz did not suffice for its realization, and that it was only after another two centuries that George Boole began to realize something similar to the subsystem that Leibniz had called the "calculus raticinator".

Let me quote from E.T.Bell young Leibniz's description of what he was aiming at:

> "a general method in which all truths of the reason would be reduced to a kind of calculation. At the same time this would be a sort of universal language or script, but infinitely different from all those projected hitherto; for the symbols and even the words in it would direct reason; and errors, except those of fact, would be mere mistakes in calculation."

I think it absolutely astounding that he foresaw how "the symbols would direct the reasoning", for how strongly they would do so was one of the most delightful

discoveries of my professional life.

Around 1900, the Dream of Leibniz came closer to realization as the necessary formalisms became available and the great German mathematician David Hilbert promoted the project. Hilbert made clear that the total calculation had to be achieved with the aid of manipulations from a well-defined repertoire. One could argue with the repertoire, and the Dutchman L.E.J.Brouwer vigorously argued against Hilbert's choice, but instead of arguing what is the "right" repertoire, one can turn the concept of proof from an absolute concept to one relative to the repertoire of admitted manipulations. In passing we note that, having abolished the notion of "the true repertoire", we still are free to have our preferences. Hilbert's revolution was in any case to redefine "proof" to become a completely rigorous notion, totally different from the psycho/sociological "A proof is something that convinces other mathematicians.". A major shortcoming of the latter view is that it gives no technical guidance for proof design and makes it very difficult to teach that kind of mathematics. Yet, or perhaps for this reason, many of the more conservative mathematicians still cling to this form of consensus mathematics; they will even vigorously defend their informality.

Here I must admit that I am a poor historian or a poor psychologist, for I see two things that I cannot reconcile with each other. On the one hand Hilbert, generally recognized as the greatest mathematician of his generation, has stated very explicitly that rigour in proof is not the enemy of simplicity, and, also, that the effort for rigour leads us to the discovery of simpler and more general methods of proof. He confirms the prediction Leibniz made, that "the symbols would direct reason". On the other hand the world of mathematics seems to continue as if these things have never been said. They don't even take the trouble of explaining why Hilbert was wrong or misguided, it looks more as if he is just ignored. [As soon as I wrote this down, I took a test and checked the index of "Descartes' Dream: The World According to Mathematics" by the mathematical philosophers Philip J.Davis and Reuben Hersch: after "Heinlein" comes "Hofstadter". Hilbert is not mentioned. I should add that the blurb characterizes the book as "A passionate plea against the use of formal mathematical reasoning as a method for solving mankind's problems...."]

Sure, mathematicians manipulated formulae prior to Hilbert, but with the exception of the most familiar cases --such as doing arithmetic--, people manipulated interpreted formulae, i.e., they justified their manipulations via the perceived properties of the things denoted by their symbols. Hilbert the formalist showed that such interpretation was superfluous because which manipulations were permissible could be defined in terms of the symbols themselves. By leaving the formulae uninterpreted, their manipulations becomes much simpler and safer. As my initial protest against the "meaningless ballet of symbols" illustrates, the preferences of the formalists are an acquired taste, but since Hilbert we know that the taste is worth acquiring.

During Hilbert's life, Leibniz's Dream, by and large, just stayed a dream. People viewed formal proofs as an interesting theoretical possibility or an unrealistic idealization, and they would regard their own proof as a usually sufficient sketch of a formal argument. They would even assure you that, if you insisted, they could formalize their informal argument, but how often that claim was valid is anybody's guess.

In the 2nd half of the 20th Century, things shifted with the advent of computers, as more and more people began to adopt formal techniques. Parts of Leibniz's Dream became reality, and it is quite understandable that this happened mostly in Departments of Computing Science, rather than in Departments of Mathematics. Firstly, the computing scientsts were in more urgent need of such calculational techniques because, by virtue of its mechanical interpretability, each programming language is eo ipso a formal system to start with. Secondly, for the manipulation of uninterpreted formulae, the world of computing provided a most sympathetic environment because we are so used to it: it is what compilers and theorem provers do all the time! And, finally, when the symbol manipulation would become too labour-intensive, computing science could provide the tools for mechanical assistance. In short, the world of computing became Leibniz's home; that it was my home as well was my luck.

Most circumstances conspired to make me very fortunate. In 1981, while I was still with Burroughs Corporation, the BP Venture Research Unit, under the enlightened guidance of D.W.Braben, gave a grant titled "The Taming of Complexity", from which Netty van Gasteren's work could be supported; a subsequent

grant for "The Streamlining of the Mathematical Argument" would enable the young British logician L.A.Wallen to visit us in Austin as a "post doc". All this assistance is gratefully acknowledged. In those years we learned that the most effective way of avoiding unmastered complexity is not to introduce that complexity in the first place. And when we focussed on that laudable goal we learned to exploit and appreciate the power of well-designed calculi.

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 more than 50 Years!

I thank everybody in general for everything, and you in particular for your attention.

Austin, 22 April 2000

Prof.dr Edsger W.Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712 - 1188
U.S.A.