Dijkstra, Edsger W.:
Selected writings on
computing: a personal
perspective

Texts and Monographs in
Computer Science
New York - Heidelberg -
Berlin: Springer-Verlag
XVII, 362 p. DM.79.00;
$ 31,60 (1982)

AMS Codes : 68 - XX

68B05 68B10

68-02 68B20

Key words:

PROGRAMMING THEORY
DIJKSTRA'S PHYLOSOPHY
EFFICIENT ALGORITHMS
GENERAL CORRECTNESS THEORY

N.B.Red.: Gewünscht von
S.Istrail (2124)

The present volume comprises a collection of 66 papers whose selection was subject to a personal perspective. They are presented in the hystorical order of their completion, giving an accurate picture of the problems with which the author was confronted, his solutions, his methods, as well as his phylosophy, aesthetics and simplicity underlying the "discovering procedure".

To enhance the joy and excitement accompanying, as well-established pattern, the reading of Dijkstra's papers, non-technical papers are also included in volume. To have a better picture of them, special literary-like constructs can be applied to them such as: essay, parable, fairy-tale, pedagogical pretext, warning, pleading, pamphlet, comprehensive explanation, open letter, aphorism, thesis. However, they require a certain kind of complex "label", which might be better defined as a "co-operation" between usually "sequential" creative "processes". This perfect mixing of the ingredients into a coherent whole, flourishing on the frontier between technical computing science and the phylosophy substantiating its essentially distinguished development, for the enormous benefit of both, is what we can call the Dijkstranian Style. And in this respect, our classification of these papers as "non-technical" becomes a very unfair manner of speaking.

The themes of reflection cover, inter alia, favourite (we think!) pedagogical topics, ranging from the didactics of programming (i.e. ≪the teaching of thinking≫!) and the sociological impact of it, to the psychology of stepping into a new programming statute (should we say dignity?) and an analysis of the obstacles such new approaches are likely to come up against from general intellectual inertia.

In this respect, the paper EWD480 : "Craftsman or Scientist?" is as eloquent as representative. The craftsman - like manner of teaching programming is the way in which ≪ the students should be taught how to solve the problems of "the real world" and that therefore, the curriculum should pay as little attention as possible to "abstract subjects"≫. This is one extreme of teaching programming, with a pronounced anti-intellectualistic flavour.

At the other end, we have the pure scientist-like manner. But ≪ his beautiful and formal apparatus, indispensable as it may be, does not necessarily suffice ≫.

As the extremes are not good, a certain "blending" is advisable; such a blending, however, is not a ≪one-dimensional question≫ . As opposed to the ≪disastrous blending viz. that of the technology of the craftsman with the pretence of the scientist≫ , substantial evidence is given of what we should try to blend: ≪the technology of the scientist with the pretence of the craftsman≫.

The first alternative is disastrous, because ≪ the craftsman has no conscious formal grip on his subject matter, he just "knows" how to use his tools. If this is combined with the scientist's approach of making one's knowledge explicit, he will describe what he knows explicitly, i.e. his tools, instead of describing how to use them! >>.

The latter alternative for blending, the advisable one, is that which comprises the teaching of << facts about systems, machines, programming languages, etc. - and it is very easy to be explicit about them, but the trouble is that these facts represent about 10 percent of what has to be taught: the remaining 90 percent is problem solving and how to avoid unmastered complexity, in short: it is the teaching of thinking, no more and no less >> .

As a highly intellectual activity, programming has introduced remarkable refinements into << how to discover the unexpected >> , and on its way toward effectiveness brought with its useful lessons on << how to avoid unmastered complexity >> and, even more, on << how to reduce the demands made on our quantitatively limited power of reasoning >> .

Somewhat corresponding to the above two views on the teaching of programming, are two views on programming and programmers. They are reported no less beautifully than instructively in

EWD512 : Comments at a Symposium

<< There are two views of programming. In the old view, the purpose of our programs is to instruct our machines; in the new one, it is the purpose of our machines to execute our programs. In the old view a programmer's expertise is proportional to his knowledge of all the funny properties of the equipment against which he has to fight a continuous battle. In the new view a programmer's competence is displayed by his good taste and the justification with which he rejects inelegant implementations and clumsy interfaces >> .

There are, included in the book, 11 Trip Reports reflecting the author's impressions recorded from 11 International Computing

Science Conferences. As it is abundantly clear, they are « more revealing about their author than about the people and places visited ».

The opportunity provided by the completion of a Trip Report turns out to be the perfect place to debate problems like the ethics of conference participation or the severe evaluation of the claims made by some participant vs. the true content (if any!) of its paper; in this latter respect, Dijkstra's keen comments are often orientated toward the ridiculization of the lack of understanding and/or confusion hidden behind the "technical language".

The reports provide without exception "all in one breath"-readings, describing in his «crystal clear» manner the facts revealed as the author's fine-toothed comb went through the conference events.

To complete the picture, a "Non-Trip Report" (EWD561) is included, where details about regular dayly work occur.

Another series of papers is devoted to "Mathematics Inc.", a delightful pretext of talking about Mathematics and making Mathematics, in lively business-like terms. It is here the place where the author's humour «floreat et crescat» , making from this series a rare intellectual delight.

To record some other themes of the non-technical papers, we shall give their titles, which are, all by themselvs, as provocative (or explanatory) as suggestive:

EWD611 : On the Fact that the Atlantic Ocean Has Two Sides.

EWD618 : On Webster, Users, Bugs and Aristotle.

EWD636 : Why Naire Program Transformation Systems Are Unlikely to Work.

EWD637 : The Three Golden Rules for Successful Scientific Research.

EWD648 : Why is Software so Expansive? An Explanation to the Hardware Designer.

EWD498 : How Do We Tell Truths that Might Hurt?

And related to the last paper, containing «computing's misery captured in a dozen of easily remembered maxims» two such "truths" are representative:

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

«APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums».

If the series "Mathematics Inc." would be found delightful by the mathematician, two other papers

EWD594 : A Parable, and

EWD678 : A Story that Starts with a Very Good Computer will suit even the finest taste of the computing scientist.

Regarding the technical paper, included in the volume, they are selected following the guiding principles: «<u>not published elsewhere</u> and as varied and as representative as possible».

Four technical papers

EWD227 : Stepwise Program Construction.

EWD376 : Finding the Maximum Strong Components in a Directed Graph.

EWD482 : Exercises in Making Programs Robust.

EWD622 : On Making Solutions More and More Fine-Grained.

are representative of the author's fundamental view of developing algorithms of a high degree of naturalness, hand in hand with its proof of correctness. And this can be interpreted as the author's understanding of "good" : an algorithm is good if it possesses a reasonable, "not <u>too</u>-hard-to-find" - proof of correctness. To make the above vague "<u>too</u>" as clear as possible, he provides plenty of evidence of the way in which the algorithms must be developed: as natural as possible, i.e. achieving the lack of "resistance" as opposed by its proof of correctness, in the process of deriving it.

More specifically, starting from the problem specifications (formulated in a clear and good notation - as opposed to the usual «hair-raising» one!) the «deepest simplicities» must be discovered, avoiding the (initial) «temptations that should be resisted» ; then, our «separation of concerns» will act efficiently to point out the basic goals, towards which we shall direct our programming effort, viewed as a "goal oriented activity". And in this process, proving correctness is not an a posteriori task; the correctness formulas are the very "passports" for introducing the programming fragments into scene.

An impressive series of papers is devoted to parallelism, concurrence and communication:

EWD338 : Parallelism in Multi-Record Transactions.

EWD386 : The Solution to a Cyclic Relaxation Problem.

EWD391 : Self-Stabilization in Spite of Distributed Control.

EWD464 : A New Elephant Built from Mosquitoes Humming in Harmony.

EWD465 : Monotonic Replacement Algorithms and Their Implementation.

EWD554 : A Personal Summary of Gries-Owicki Theory.

EDW607 : A Correctness Proof for Communicating Processes: A Small Exercise.

EWD622 : On Making Solutions More and More Fine-Grained.

EWD623 : The Mathematics Behind the Banker's Algorithm.

EWD629 : On Two Beautiful Solutions Designed By Martin Rem.

EWD643 : A Class of Simple Communication Patterns.

Among the subjects covered by the above papers we mention the

the following: logical problems involved by the carrying of parallel transactions; the study of self-stabilizing systems; « systematic ways for finding the algorithms ensuring some desired form of co-operation between a set of loosely coupled sequential processes» ; « systematic methodology for elephant design » similarly to the existing one for the design of sequential programs, virtual storage implementation; the Gries-Owicki nonoperational approach to concurrency; the (successful) investigation, in isolation, of mosquitoes communitating à la Hoare; proving termination of intricate elephants endowed with bags of pebble; « smoothing the correctness proof » and giving strong evidence of « the feasibility of departing from the usual operational arguments in which one tries to visualize classes of computational histories » ; methods for deadlock prevention; a P/V - implementation of conditional critical regions; avoiding the danger of deadlock or starvation.

Two papers deserve special attention:

EWD462 : A Time-Wise Hierarchy Imposed upon the Use of a Two-Level Store.

EWD508 : A Synthesis Emerging?

The first one, EWD462, though enthusiastically accepted for publication, was never published before. The most suggestive presentation of the paper is formulated in the first phrases of the referee's conclusions, fairly included by the author in the commented contents of the book;

"The paper formulates and illustrates some fundamental principles of software engireering which have been shamefully and disastrously neglected in the past; for this reason its publication is to be highly recommended".

The second paper, EWD508, « records the highlights of a discussion » together with C.A.R.Hoare, which led the latter to design his famous "Communicating Sequential Processes".

Some other subjects are covered by some other papers, e.g. shunting monitors, error recovery vs. basically unreliable storage techniques, small vs. ≪sizable≫ programs, program inversion, weak and strong termination of programs, continuity and bounded non-determinacy.

Last but neither least nor accidental we have to say some words about EWD538 : A Collection of Beautiful Proofs. In an effort to isolate what "elegance in proof" means, the author presents this raport as a draft chapter to a possible book titled "On the Nature and Role of Mathematical Elegance". The problems included in the Collection, are chosen so that they could be appreciate by the ≪generally educated≫ and that ≪all mathematicians will agree that they are beautiful≫ . They support strong evidence that mathematical elegance is not an elusive concept at all.

"The fighting against software inelegance" is now a movement, whose exponent is E.W.Dijkstra. And in this respect, his words are representative:

≪We have to fight chaos, and the most effective way of doing that is to prevent its emergence. We have to learn to avoid all forms of combinatorial complexity generators that, when active, rapidly tax our ability to carry out a caseanalysis far beyond the limits of our power of reasonong. To recognize the emergence of a combinatorial complexity generator long before it has poisoned your design beyond salvation requires constant vigilance, a vigilance that can and should be taught.

To circumvent such emerging complexity generators may very well be a tough problem, the solution of which I can only describe as mathematical invention≫ .

Regarding programming environment, rejecting inelegant solutions is not an aesthetic demand only, but the very base of reliable design. And in this context, "elegance finding" is not everybody's business.

Can this be put into a more persuasive shape than the following: ≪ Don't blame me for the fact that competent programming, as I view it as an intellectual possibility will be too difficult for "the average programmer" - you must not fall into the trap of rejecting a surgical technique because it is beyond the capabilities of the barber in his shop arround the corner ≫.

The author's strong demand for elegance is based on his essential formation as ≪ pragmatic industrial mathematician ≫, and all his papers are a collection of beautiful proofs that "Elegance emergence" is the very business of the true Computing Scientist.
His writings, profound and beautiful, with strong influence on our patterns of thinking, seem to unveil those facets of the design process which are equally natural, efficient and elegant, i.e. (quoting G.Pólya) its magic.

In every fibre of our bodies we agree that placed in that way, our effort will be in the right perspective.

It is our fortune that his discourse about "magic", i.e. Dijkstra's "meta-magic", is magic itself.

S.Istrail