STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

PROGRAMMING PROVERBS REVISITED

M. FRENŢIU and B. PÂRV*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received February 17, 1994
MS subject classification 68N05

REZUMAT. - Proverbe ale programării revăzute. În lucrare se prezintă metode, principii şi reguli considerate importante în activitatea de programare Se subliniază importanţa acestora în orice curs de învăţare a programării

Computer programming is still in a state of crisis, at least for two reasons the hardware changes, and the appearance of new problems which can be solved by computer The complexity of programs is increasing continuously, and it generates major changes in program design techniques The notion of "good program" can be considered from two different points of view programmer's view, and user's one From the user's point of view one can distinguish 10 so-called "external quality factors" [10] correctness, robustness, extensibility, reusability, compatibility, efficiency, portability, verificability, integrity, and ease of use From programmer's viewpoint, one can enumerate two major criteria for a good program modularity, and complete documentation Of course, the external quality factors must be taken into account as final goals in the software development process

All these quality criteria must find their place in the formation of new programmers There is a continuous need to teach programming for obtaining a better productivity, i e to

* *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

teach the students the methods that allow us to obtain correct programs from the first execution As Floyd [4] pointed out in his Turing Award Lecture, there "is possible to explicitly teach a set of systematic methods for all levels of program design" Methods, principles, and rules considered important in programming are given below Also the bibliographical source is indicated in the brackets

1 *Define the problem completely* [7, 9] One cannot write a correct program if the problem to be solved is not known exactly By this we mean to write the specifications of the problem As it is known [11], this is not an easy problem, but a very serious one Often the beginners start to write the program but they do not know what are the results that must be obtained

2 *Think first, program later* [9] This may be interpreted to design the algorithms correctly Think to them, try to prove their correctness, and write the program later, when you are sure that everything is correct

3 *Use Top-Down Design* [4, 7, 9] This is a very well known, and important programming paradigm [4] It is also met as *step-wise refinement* method [13], or *Divide and conquer* principle [7]

4 *Use Modularity as much as possible* [9] A function, a procedure, a Turbo-Pascal unit, a Modula module, or an Ada package are considered modules Each module of a program is more understandable than the entire program Also, using modules, the logical structure of the program is improved Build up libraries of your modules for reusability

5 *Use library routines whenever it is possible* [9] This rule is a consequence of rule 4 Certainly, the existing routines are ready to be used, no time needed for writing and testing

these routines Thus the productivity, and the probability of correctness will increase

6 *Design the algorithms by Structured Programming paradigm* [2, 4, 9, 13] This rule asks to design first the algorithms in a Pseudocode language, and only then to translate them in a programming language Also, it requires to think to the structure of the product, at each level

7 *Define a new data type as an Abstract Data Type (ADT)* [6] The above rule asks the designer to think generally, not in the context of the solved problem An ADT may be viewed as a module that defines a data structure and the operations on this structure This independence of the context has beneficial effect for the reusability of modules Also, an ADT is an open system, i e one can add new operations, not affecting the old ones, and not affecting the programs that already use this ADT

8 *Design input-output routines for each abstract data type* [5, 6] These Input-Output operations are very useful in general Often, when a standardized interface is recommended, for these operations one uses videoformat, such as Turbo Vision from Borland This rule is one way of achieving rule 21

9 *Use object-oriented design* [1, 10] This technique permits to obtain flexible, and easy modifiable programs The programs obtained by this technique are easy to maintain since, by using the hierarchy of classes in libraries of components, a massive reusability of these components becomes possible Also, adding new components does not affect the programs that already use the old components On the other side, the other feature of object-oriented programming, the polymorfism, simplifies communication protocol between objects A program in OOP sense is considered as a structured collection of objects which

communicate by message passing

10 *Strive for continuing invention, and elaboration of new paradigms to the set of your own ones* [4] This idea, due to Floyd, is very well presented in his Turing Lecture He recommended to "identify the paradigms you use, as fully as you can, then teach them explicitly"

11 *Prove the correctness of algorithms during their design* [7] The errors must be eliminated as soon as possible. Trying to prove correctness, some wrong parts may be discovered And this can be done much earlier than running it on a computer Also, if we succeeded to prove it, the confidence in its correctness grows up significantly Gries [7] insists on developing correct programs from the beginning His words are "A program and its proof should be developed hand-in-hand, with the proof usually leading the way"

12 *Concentrate to the important things of the moment, postpone the details later* [9, 13] This rule is connected to the stepwise refinement method But it has some other aspects At all levels give attention to the main things, for example do not lose time to print the results nicely if you are not sure these results are correct

13 *Nevertheless the details are important* [6, 13] First, the software products must respect rigorously the specifications Second, the form of the printed results are more important for users than the entire work done for developing the product These must please the users !

14 *Choose suitable and meaningful names for variables* [7, 13] The readability of a program may be one of its very important quality It is very useful during maintenance phase, when many other programmers have to work on the program More, Gries [7] recommends

to define rigorously the meaning of a variable by an assertion that remains true during the execution of the algorithm

15 *For every variable of a program make sure that it is declared, initialised, and used* [12] A variable may appeared in a program accidentally, other variable may not be initialised since a line of a program was not typed

16 *Use symbolic constants* [6, 13] This rule is a consequence of a Murphy like rule

The constants must be considered variables !

One recommends to define symbolic constants at the beginning of a program (module) procedure and to use the names inside Any modifications means small changes in the definition of the constants, and eliminates further errors

17 *Use names for all data types of the program* [6] We consider that all properties of a type are concentrated in its name Using names, the modifiability of the program is easier Also, the clarity is higher

18 *Use intermediate variables only if it is necessary* [9]

The uncontrolled utilization of auxiliary variables, by breaking expressions, just complicated, in subexpressions assigned to new variables, diminishes the clarity of the program, and makes more difficult the program verification

19 *Declare all auxiliary variables of a procedure as local variables* [6] This rule is connected with the autonomy of the corresponding procedure It offers the following advantages easier testing of the procedure, procedure independence of the context in which it is used, no secondary effects due to unexpected changing of the values of global variables

20 *Be careful at the parameters of the called procedure* [6, 13] Each module must

be used only through its interface, that is, the actual parameters passed to the module, which must correspond to the formal parameters (dummy variables) Respect their meanings, and be careful to the correct usage of the procedure calling mechanism

21. *Verify the value of a variable immediately it was obtained* [6] A variable receives a value by an assignment or by an input operation. In both cases the value must be correct, it is worthwhile to check it Especially for input operation, a variable must be protected from wrong values

22 *Think to pretty writing the text of the program* [9,13].

Most of the programming languages allow free format, i e. the blank spaces may be used freely Use them when writing the text of the program, to improve the clarity of this text It must leap to the eyes the beginning and the end of each statement Use indentation for this purpose Make the structure of your program visible.

23 *Use the FOR statements properly; do not change the value of the counting variable, or the limits inside the cycle* [9] This rule ask to respect the semantics of the For statement Do not use For when Repeat or While control structures are most appropiate Changing the limits, or the value of the counting variable may cause invisible errors, very difficult to discover

24 *Do not leave a FOR cycle through a Goto statement* [9] This rule is specific to Fortran programmers, but may be met in those languages that possess GOTO statements The reasons for respecting this rule are the same as for the rule 23

25 *Avoid GOTO statements* [3] The Goto controversy [3, 8] is well known Using unrestricted Goto statements destroys the good structure of that module These statements

must be used only if the programming language does not possess the standard computing structures

26 *Avoid tricky programming* [9] A program must be maintained, oftenly, by other persons different from the people who wrote it And tricks are not compatible with good structure, clarity, and flexibility. Also, for the portability of the program, one must avoid the implementation dependent features.

27. *Use comments* [9, 13]. The text of a program (module) must be understood easily and unambiguously by all the other programmers who have to read it For this purpose the comments can be very useful. We think that each module must contain comments saying at least what it is doing, i e the specifications of the module, and the meaning of the used variables

28 *Verify (test) the correctness of a module soon after it was obtained* [7]. The rule 10 ask us to prove formally the correctness of a program (module) But, just if we have done it, we still have to test this module After all, the proof may be wrong, or the implementation of a correct algorithm may be incorrect. Ledgard [9] recommended "to hand-check the program before running it" We find this very useful for the beginners, some students better understand their errors running themselves their wrong programs

29 *At each phase verify the programm correctness* [6, 13] The verification of program correctness means the verification of specifications, the formal proof of algorithm correctness, the inspection of the text of the program, and the testing of it Remove any error as soon as possible !

30 *Use assertions to document programs and verify their correctness during*

debugging process [7] If one has proved the partial correctness of the algorithms he has used assertions in some points of the algorithms These assertions must be invariantly true during execution They reflect the meaning of the corresponding variables. In the debugging process verify their correctness. If they are not true some errors have occured, and they must be eliminated

31. *Write good documentation simultaneously with program building* [13] The users need a documentation manual, and the maintenance activities need information about all levels of program development Often, there is no documentation at all. The above mentioned rule asks to write the documentation simultaneously with the development of the program. The program itself must be selfdocumented by comments But it is not enough There must be written documents that show all the decisions at each level of the development process. There must exist documents for specification, design, implementation, and testing. Also, a user manual is needed

32. *Use the existing debugging techniques* [9] We hope to obtain error-free programs But errors may arise, and finding and correcting these errors is an important, and very often, an unpleasant job Every operating system has built in it some debugging aids Use them to assist you in finding the errors

33 *Ask for computer assisted software development* [7, 13] Computers can help people to carry out their unpleasant works Particularly, they can help in program development in different ways Many of them are mentioned in the excellent book of Schach [13] planning the activities, and many activities done by Software Development Environments, known as CASE (Computer-Aided Software Development) There are many

activities that have to be performed during the development of the program, such as computations [4], or various decisions

34 *Think to the program portability* [9] A program must be portable, i e to be able to be run directly on a different machine, other than the original one Portability is not usually an issue to worry about But it may be an important quality of a program Isolate into modules those parts of the program that usually change from computer to computer (such as input/output operations). All other modules can be built portable, using statements corresponding to the "standard specification" of the implementation language, and avoiding the particular extensions which are dependent on the compiler implementation

REFERENCES

1 Coad,P , Yourdon,E *Object-oriented Design*, Prentice-Hall, 1991
2 Dahl,O J , E W Dijkstra, C A R Hoare. *Structured programming*, Academic Press, London, New-York, 1972
3 Dijkstra,E W *GOTO Statement Considered Harmful*, Comm A C M , 11(1968),no 3, p 148
4 Floyd,R W *The Paradigms of Programming*, Comm A C M , 22(1979), no 8, pp 455-460
5 Frenţiu,M , B Pârv, V Prejmereanu *Abstract data types for increasing the productivity in programming*, Seminar on Computer Science, "Babeş-Bolyai" University, Preprint no 5, 1992, pp 8-13
6 Frenţiu,M , and B Pârv, *Metode şi tehnici în elaborarea programelor*, Promedia, Cluj-Napoca, 1994
7 Gries,D *The Science of Programming*, Springer Verlag, Berlin, 1985.
8 Knuth,D *Structured Programming with GO TO Statements*, A C M Computing Surveys, 6(1974), no 12, pp 261-301
9 Ledgard,H F , *Programming proverbs for Fortran programmers*, Hayden Book Company, Inc , New-Jersey, 1975
10 Meyer,B *Object Oriented Software Construction*, Prentice-Hall, Englewood Cliffs, 1988
11 Myers, A *A Controlled Experiment in Program Testing and Code Walkthroughs Inspection*, Comm A C M , 21(1978), no 9, pp 760-768 .
12 Naur,P *Proof of algorithms by general snapshots*, BIT, 6(1966), pp 310-316
13 Schach,S R *Software Engineering*, IRWIN, 1990, U S A