SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 2, 103–105 (1972)

# Guest Editorial

## The Quality of Software

The main problem in the design of any engineering product is the reconciliation of a large number of strongly competing objectives. In the case of general purpose computer software, I have made a list of no less than seventeen:

(1) *Clear definition of purpose*

The aim of every item of software must be most clearly defined; and recommendations on the circumstances of its successful use must be fully explained. Vagueness in this matter is intolerable; and of course, every feature of the program delivered must be oriented towards the declared purpose.

(2) *Simplicity of use*

The software must be very simple to understand and use in the majority of cases, and the extra complexity of its use in less normal circumstances must be kept to a minimum.

(3) *Ruggedness*

As well as being very simple to use, a software program must be very difficult to misuse; it must be kind to programming errors, giving clear indication of their occurrence, and never becoming unpredictable in its effects.

(4) *Early availability*

Software must be made available together with the first delivery of the hardware to which it is relevant. If software is not available at this time, the customer is forced to develop his own *ad hoc* techniques, and may never be weaned to use the more elegant methods of software which has arrived too late.

(5) *Reliability*

The original program delivered must be as free from errors as possible, and if errors are discovered, they must be capable of correction with the greatest rapidity. This is dependent on simplicity of programming techniques and first-class program documentation.

(6) *Extensibility and improvability in light of experience*

Since lack of hardware and lack of experience make it impossible to deliver perfect software in a Mark 1 version, the software programs should be capable of further development and improvement. This again demands simplicity of approach and good documentation.

(7) *Adaptability and easy extension to different configurations*

The purpose of software is to satisfy as many customers as possible, including those with a wide range of configurations. Furthermore, a customer who expands his hardware after purchase will wish to avoid changing his programs or his programming techniques and concepts.

(8) *Suitability to each individual configuration of the range*

In addition to adaptability over a range of configurations, the software actually available for each member of the range should be well suited to the capabilities and needs of that particular configuration.

(9) *Brevity*

Software programs should be as short as possible, particularly those which have to co-exist in the store with the programs which use them. Furthermore, the use of the software should not involve extra length in the program which makes use of it.

(10) *Efficiency (speed)*

The speed of software programs should be sufficient to justify their use in most circumstances.

(11) *Operating ease*

The most critical factor in the efficiency of an installation is often the smoothness of the operating system. Software should be designed to make the job of the operator as simple as possible.

(12) *Adaptability to wide range of applications*

Since the purpose of software is to find wide applicability, it is necessary to consider the widest possible areas of application in its design and implementation.

(13) *Coherence and consistency with other programs*

As far as possible, software programs should be compatible with each other, and capable of being used either separately or in conjunction with each other. Furthermore, any overlap between the programs available should only be accepted if the need justifies it.

(14) *Minimum cost to develop*

The cost of software development in manpower and machine time is a vital factor in the planning of a suite of software programs.

(15) *Conformity to national and international standards*

When national or international standards for character codes, tape formats or languages have been set up, or seem likely to be set up, these should be observed to the maximum extent.

(16) *Early and valid sales documentation*

Sales documentation describing the most important features of the software under development should be available early in the life of the project; and they should remain valid when the product is finally delivered.

(17) *Clear accurate and precise user's documents*

In addition to sales documents, the user's documents should contain clear instructions on a program's proper method of use, and an accurate description of its properties.

The document should be available early in the life of the project, and should be kept scrupulously up to date.

There can be little doubt that modern software fails to reconcile these objectives, particularly with respect to one or more of (1)–(6), (8)–(10), (14) and (17). What is the solution ? The first necessity is for the software designer and his customers to recognize that there is a conflict of objectives, and that there is the need for compromise. Secondly, it is essential that the compromise be selected in the light of a sound knowledge of good algorithms and program structures, and not merely as a result of wishful thinking of a sales or product specification department.

As an example of the way in which a good choice of technique should influence a software specification, consider the Elliott ALGOL translator for the 503. The compiling technique chosen in 1960 was what is now known as 'single-pass topdown syntax-oriented compilation', which simultaneously achieves high-speed compact compilers as well as high-speed compact object code. In order to reconcile these objectives, it was necessary to make non-essential (but apparently arbitrary) restrictions on the ALGOL subset to be implemented. I believe that the slight sacrifice in objective (15) was more than compensated in the customer's own interest by the over-all quality of the product measured in terms of all the objectives. Attempts to transfer programs from the 503 to other machines were hindered far more by the low quality of other manufacturers' compilers than by incompatibility of subsets.

Unfortunately, in the past decade, software writers' main pride has been to implement an arbitrary specification handed down to them from 'above' (e.g. ALGOL 68, PL/1); and not to pay much concern to the quality of either the specification or the product. They tend to argue from the fact that you can do *anything* by software to the conclusion that you can do *everything* by software. And you can, too; but is it worth it ? Only very few software designs (a notable example being PASCAL) have actually been made to optimize the quality of the product in the light of known software techniques.

So my advice to the designers and implementors of software of the future is in a nutshell: do not decide exactly what you are going to do until you know how to do it; and do not decide how to do it until you have evaluated your plan against *all* the desired criteria of quality. And if you cannot do that, simplify your design until you can.

C. A. R. HOARE
*The Queen's University of Belfast,*
*Belfast,*
*N. Ireland.*