to Alvey Directorate

Additional recommendations suggested by C.A.R. Hoare - 4th July

IEE, BCS, BSI and NPL should be funded to produce a series of high quality software standards, independent of commercial pressure. Funds should be sufficient to hire consultants from Universities and Industry; and to place contracts for trial implementations.

IEE and BCS should be funded to design a professional certificate of qualification in fault-free programming. This should be a direct and objective test of the candidate's ability to apply known methods for avoidance of error on safety-critical projects.

In schools, a shift towards more structured languages and programming systems should be accelerated, if necessary by purchase of more modern hardware and software.

The simpler mathematics of programming should be introduced into the computing syllabus at the secondary school level.

Government procurement contracts should require a guarantee that all software faults detected in the first two years after delivery should be corrected by the supplier under guarantee, without prejudice to further claims to consequential damages for these or for subsequent errors.

On software products and contracts, disclaimer of responsibility for errors and inadequacy of the product should be declared illegal. Suppliers should be recommended to take out insurance against direct and consequential damages arising arising from programmer negligence.

A series of 'summer schools' should be funded to assist propagation of improved methods among academics, teachers and industrial programmers.

Quality

Draft

C.A.R. Hoare, 4th July, 1985

A. The end

Our objective is to penetrate the world software market by establishing a pre-eminent reputation for the quality of British software. The criteria for quality of software, like those of other engineering products are numerous, but fairly standard. The following list of seventeen criteria was compiled twenty years ago. Most points apply equally to marketed software and in-house program development.

1. Clear definition of purpose.

   Software must be designed from the start to meet a clearly defined purpose, and every decision must be justified by appeal to that purpose.

2. Simplicity of use.

   The major barrier to wider and more successful use of computers is often the complexity of the user interface: this results in long periods of training/acclimatisation, loss of confidence and reliability.

3. Ruggedness.

   As well as being simple to use, software must be difficult to misuse. User error must be treated gently, and never lead to unpredictable effects.

4. Early availability.

   The first tolerable product on the market place often gains an unassailable lead, which later products of higher quality cannot shake. Delays in delivery of process control and commercial software are often more expensive than all other software costs put together.

5. Reliability

   Computer hardware is incredibly reliable; the software is a notorious let-down. In some applications the results of software unreliability could be disastrous.

6. Extensibility in the light of experience.

   The way to capture a market is to deliver a good product this year, a better one next, and an unbeatable one the year after. Well-designed and well-structured programs offer this opportunity - though this fact should never dilute the engineer's prime responsibility to get it right first time.

7. Adaptability and easy extension to different configurations.

   The range of variety in hardware configurations grows wider with the advance of technology. To open up wider markets, and reduce the need for user retraining, software must incorporate abstractions capable of adapting to many concrete configurations, both those existing at time of first delivery and those developed later.

8. Suitability to each configuration in the range.

   The user of a particular configuration does not wish to sacrifice the particular powers of the hardware to the dictates of uniformity or compatibility.

Cont/....2

9. Brevity

Programs and data should not be extravagant in the use of store. The cost of store is going up (relative to processing power), and demands on it are increasing. However cheap store is, twice as much store is twice as expensive. It is the duty of an engineer to be economical in the management of all resources. Compact programs tend to be faster, more reliable and cheaper to produce and maintain.

10. Efficiency (speed)

Fast response time (even below one second) in complex application areas, (especially with interactive graphics) is a vital quality. Computer processing power is increasing, and the cost is reducing; but this does not nearly match the rate of increase in user expectations. Increasing efficiency on a fixed hardware configuration can give a significant competitive advantage since it permits the customer to lengthen the interval between successive hardware upgrades.

11. Operating ease.

This is a crucial factor in modern interactive applications. The design of the man-machine interface needs ever increasing attention.

12. Adaptability to a wide range of applications

For general purpose software, languages, operating systems and utilities, a wide range of application is especially important. Technical means are required to avoid loss of quality by excessive generality.

13. Coherence and consistency with other programs.

A program which is part of a suite of programs with a coherent design philosophy is much more attractive; and leads to a wider and more stable market base.

14. Minimum cost to develop

The supplier who has reduced production costs and lead times will have a significant competitive advantage, as well as better profits.

15. Conformity to national and international standards.

Unfortunately, in the last twenty years, the quality of standards in software has seriously declined. They are arbitrary, obscure, complicated, expensive, unreliable, late,.... and often they fail most signally to meet the primary objective of standardisation, namely trouble-free interconnection of products from different suppliers.

16. Early and valid sales documentation

Under this heading should come all other criteria of successful marketing.

17. Clear, accurate, and precise user documents

This is so important that a user's manual should be written before a project is approved for implementation. Unnecessary design complexity is often revealed while it can still be remedied. In writing a user manual, the designer clarifies his view of the needs and background of his user population, and thereby improves the quality of all his subsequent design decisions.

Contd/....3

B. The means

Twenty years ago achievement of software quality was more a matter of luck than good judgement. Fortunately, during the last twenty years much research and development has taken place in programming methodology, and much of the best of it in Britain. The results of this research are now available to be exploited in Britain or elsewhere.

1. Mathematical methods of programming

The major breakthrough in the last ten years has been the introduction of mathematical modelling and proof in all stages of program specification, design and implementation. At the analysis stage, it is used to explore specification and design options for software before any commitment is made in implementation. A wide range of design options can be formulated and discussed with colleagues and clients without the fog of misunderstanding and ambiguity which in the past has often characterised the early design process.

Formulation of a mathematical model or specification of a software design helps us in achieving many of the other criteria of quality. Relevance to purpose can be assessed at the earliest stages. By abstracting from details of implementation, the user interface is simplified; adaptability to different configurations can be planned; and a clear logical structure is laid out for user manuals, with clear definitions of the minimal set of necessary technical terms. By sound specifications, constructed in advance of implementation, it is possible to plan for extensibility and coherence of a suite of programs, which cannot all be written at one time. By examination and elimination of unnecessary special cases, the software can be made more rugged, shorter, more reliable and cheaper to develop. In particular, mathematics can check the ruggedness of a design by ensuring that every user action can be immediately reversed. By careful specification of the interfaces between modules of a large program, there is the prospect of proving the correctness of the design before implementation of the modules starts. This could significantly reduce the high cost of program changes during development, and eliminate expensive and unpredictable delays due to correction of errors discovered during system integration tests. Finally, the mathematical method of development of designs can be carried right down to the level of code, thereby achieving a reliability comparable with that of mathematical proof. Mathematical proof methods must be used first on programs critical to human safety; but as experience and confidence grows, they will be profitably applied on all commercial projects, where reduction of errors will reduce costs and increase customer satisfaction.

In the longer term if mathematical methods are applied to the formulation of standards, it may help to tackle some of the appalling problems arising in this area too. It would not be technically difficult to establish reputation that British Standards are technically superior to international ones.

The introduction of mathematical methods generates considerable enthusiasm among programmers who have suffered from the problems which can be solved thereby; there is a danger that the methods may be tried even in cases where they are not yet sufficiently well developed, or where their use is not cost-effective. Furthermore, there are many excellent products that have been designed and implemented without any explicit appeal to mathematics; such products allow little scope for improvement. Nevertheless, these risks must be faced in the interests of raising the general standards of quality throughout the British software industry.

Contd/....4

2. Improved Tools

In the last twenty years, improved tools have significantly reduced the cost of software production. The most important programming tool is a programming language: modern high-level programming languages offer considerable security and reliability by compile-time checks; they provide an opportunity to construct programs which are readable by a programmer's colleagues and successors; and this contributes to reliability, and adaptability; modern compilers give an opportunity for fast turnround of prototype or experimental programs; and they considerably reduce development costs. The introduction of individual program development workstations is a considerable aid to small projects, provided that it is not used as a substitute for careful analysis and design prior to implementation. For larger projects, the concept of a program support environment offers assistance to management in the control of costs and timescales; and inclusion of configuration control is indispensible where adaptations are needed for a wide range of configurations. The integration of all these developments together with the relevant mathematical methods is being pursued in the Alvey Software Engineering Programme.