SOFTWARE ENGINEERING

C.A.R. Hoare

On April 8-9, 1976, the British Computer Society is holding its first Symposium on Software Engineering. In this article, the chairman of the Symposium presents his views on the subject, which are not necessarily shared by the speakers he has invited to address the Symposium.

If words could cure the ills of our profession of programming, what a healthy and highly respected profession it would now be! We have had "modular programming"; the craze for "structured programming" has hardly yet reached the height of its commercial profitability; and already we have a newcomer to the charts - the theme we have all so long been waiting for - yes, it's "SOFTWARE ENGINEERING". The experienced programmer will greet the gladsome tidings with a stifled yawn, and turn to more urgent and important tasks.

But perhaps there is something he could learn from these catch phrases on their passage from popularity to oblivion. Certainly, the latest combination of the new but already tarnished word "software" with the old and respected profession of engineering is such a startling contradiction that it should give us pause. Let us compare the ideals of the professional engineer with those adopted by some programmers of the present day.

One outstanding characteristic of the professional man, be he a doctor, architect, or engineer, is that he understands the real needs of his client or employer, often very much better than the client himself; and he has the ability and status to persuade the client to recognise his own interests and to abandon his less useful flights of fancy. Then he has the professional skill to recommend from a range of known and trusted techniques those methods that in the given circumstance will achieve the required effect at minimum cost and inconvenience to the client. And finally, he has the professional integrity to resign his post or commission if his recommendations are not accepted.

I fear that there is a sad contrast with some programmers, whose only wish is that their client should "make up his mind what he wants", and who will welcome his most elaborate fancies as a challenge to their programming ingenuity. How many of them are ignorant of, or prefer to ignore, the known techniques used successfully by others, and embark on some spatchcocked implementation of their own defective invention? And I know only one programmer who resigned on the spot when his advice was not taken by his less technically competent manager.

A second characteristic of the good engineer is his vigilance in seeking every opportunity to reduce the costs and increase the reliability of his product. He realises that the conflict between these two objectives can be resolved only by preserving the utmost simplicity of concept, specification, design, and implementation. Above all, he insists that he shall have a complete understanding and control over every aspect of his project - and the more difficult his project, the more firmly will he insist on the simplicity without which he cannot understand what he is doing.

Here again, we find exactly the opposite characteristic in some of our best programmers, who deliberately avoid simplified solutions; they obtain satisfaction from the sophistication of their designs and methods, and derive excitement from engaging in projects of a complexity slightly beyond their ability to understand and control. They may well succeed once; but on the next occasion they may discover that there is no way of distinguishing (in advance) between what is slightly and what is totally beyond their comprehension.

A great advantage of the present day engineer is that his designs are based on sound mathematical theories and computational techniques, discovered over the years by his brilliant predecessors, and now enshrined in textbooks and undergraduate teaching, in mathematical tables, and in standard codes of practice. But in spite of the soundness of his theory, he still has many causes for worry that his abstractions (and his product) may break down - a faulty casting, a defective batch of components, a lazy workman, or an unpredictable natural hazard.

The computer programmer has little worry of this kind: his working material is the hardware of the computer itself, and its reliability can usually be taken for granted. Certainly, by far the most significant cause of failure in software are the errors and oversights of the programmer himself. But here perhaps he is not wholly to blame, since he has no widely accepted mathematical or theoretical foundation for his work. This is a most urgent topic of research at our Universities and elsewhere, and it is to be hoped that the results will be most widely and most rapidly propagated.

A final point of contrast lies in the working tools of the profession. An engineer naturally demands of his tools the highest quality and precision,

reliability, convenience, and cheapness in use. In many professions, the tools are quite simple; in others they are more complex. But in either case the engineer has developed an intuitive understanding and ingrained mastery of their proper uses; and this frees him to devote his whole intellectual effort to the understanding and solution of his clients' problems.

The basic tools of the programmer are the programming languages and compilers, job control languages and operating systems, utilities and other software supplied in profusion by the manufacturer of his computer. But what a sorry comparison with the tools of other professions! That they are unreliable, that they are profligate of computer time and storage, that they are inconvenient in operation - these are facts that have been long recognised and widely suffered. Perhaps the worst symptom (and also a large part of the cause of the trouble) is their extraordinary and still increasing complexity, which totally beggars the comprehension of both user and designer. Among manufacturers' software one can find what must be the worst engineered products of the computer age. No wonder it was given away free - and a very expensive gift it was, to the recipient!

But still we have some experienced programmers and managers who actually welcome the stuff, praise it, want more of it, and even pay for it. Here perhaps the fatal attraction is the very complexity, which would revolt the instincts of any engineer, but which, to the clever programmer, masquerades as power and sophistication. He may have even less creditable motives: the use of unreliable tools both increases and excuses the unreliability of his programs; the use of inefficient tools both increases and excuses the inefficiency of his programs; and the complexity of his tools can protect him from close scrutiny or control of his client or employer. And finally, after a few years experience of some particular product, the programmer finds that even his partial understanding of it can command a high salary; and he has the strongest motive for refusing to learn something new, and for rejecting the idea that it might possibly be an improvement. And his manager who committed himself to that product many years ago has an even stronger personal and financial interest in its perpetuation.

The attempt to build a discipline of software engineering on such shoddy foundations must surely be doomed, like trying to base chemical engineering on

the phlogiston theory, or astronomy on the assumption of a flat earth. But the study of manufacturers' software is an excellent way to sharpen our understanding of the principles of software engineering, both because of its consistent violation of those principles, and because it makes a serious and creditable attempt to define the working tools of the software engineer. That is why the forthcoming BCS Symposium on software engineering will concentrate on this aspect of the subject.

The Symposium will take an optimistic, practical and forwardlooking approach. There are many ways in which existing tools can be used more effectively - by adoption of supplementary software packages, by instrumentation and tuning, by program editors and preprocessors, by structured programming aids, training manuals and courses, and by standards of programming practice. On the first day of the Symposium, a series of expert speakers will survey the range of methods which are immediately available for practical use.

On the second day, we shall look slightly further ahead, and describe some of the possibilities of further improvement that are now being opened up by fundamental and applied research. Even if the practical difficulties of change delay the widespread application of results of new research, it is important that programmers and managers should understand now what they are; so that they are never again led astray by the specious promise of sophistication and complexity.

In this short sermon on the theme of software engineering, I have made many allegations against the quality of software, and against the competence, intelligence, and integrity of programmers. But I have not given a single example to support my case, nor have I named a single name. Let me do so now: I name the guilty man: I name myself. Within myself I have discovered all the faults which I have ascribed to programmers in general. If my remarks carry any conviction, it can only be because my reader has made a similar discovery.