“The Role of Formal Techniques: Past, Current and Future or How Did Software Get so Reliable without Proof?”

C.A.R. Hoare

University of Oxford

December 1995

Extended Abstract

Twenty years ago it was reasonable to predict that the size and ambition of software products would be severely limited by the unreliability of their component programs. Crude estimates suggest that professionally written programs delivered to the customer can contain between one and ten independently correctable errors per thousand lines of code; and any software error in principle can have spectacular effect (or worse, a subtly misleading effect) on the behaviour of the entire system. Dire warnings have been issued of the dangers of safety-critical software controlling health equipment, aircraft, weapons and industrial processes, including nuclear power stations. The arguments were sufficiently persuasive to trigger a significant research effort devoted to the problem of program correctness. A proportion of this research was based on the ideal of certainty achieved by mathematical proof.

Fortunately, the problem of program correctness has turned out to be far less serious than predicted. A recent analysis by Mackenzie has shown that of several thousand deaths so far attributed to reliance on computers, only ten or so can be explained by errors in the software: most of these were due to a couple of instances of incorrect dosage calculations in the treatment of cancer by radiation. Similarly predictions of collapse of software due to size have been falsified by continuous operation of real time software systems now measured in tens of millions of lines of code, and subjected to thousands of updates per year. This is the software which controls local and trunk telephone exchanges; they have dramatically improved the reliability and performance of telecommunications throughout the world. And aircraft, both civil and military, are now flying with the aid of software measured in millions of lines — though not all of it is safety-critical.

So the questions arise: why have twenty years of pessimistic predictions been falsified? Was it due to successful application of the results of the research which was motivated by the predictions? How could that be, when clearly little software has ever been subjected to the rigours of formal proof? The objective of these enquiries is not to cast blame for the non-fulfilment of prophecies of doom.
The history of science and engineering is littered with untrue predictions and broken promises; indeed they seem to serve as an essential spur to the advancement of human knowledge. Nixon's campaign to cure cancer within ten years was a total failure; but it contributed in its time to the understanding on which the whole of molecular medicine is now based. The proper role for an historical enquiry is to draw lessons that may improve present practices, enhance the accuracy of future predictions, and guide policies and directions for continued research in the subject.

The main body of the paper is a survey of current software engineering practice. It reveals that the techniques employed to achieve reliability are little different from those which have proved effective in all other branches of modern engineering: rigorous planning of procedures for design inspection and review; raised levels of machine support for design automation; quality assurance based on a wide range of targeted tests; careful structuring of the product into modules and layers; continuous evolution by adaptation of products already in widespread use; relaxation of resource constraints and deliberate over-engineering (otherwise known as defensive programming or belt-and-braces). Formal methods and proof play no greater role in large-scale programming than they do in any other branch of modern engineering.

The conclusion of the enquiry will be that in spite of appearances, modern software engineering practice owes a great deal to the theoretical concepts and ideals of early research in the subject; and that techniques of formalisation and proof have played an essential role in validating and progressing the research. However, technology transfer is extremely slow in software, as it should be in any branch of safety critical engineering. Because of the backlog of research results not yet used, there is an immediate and continuing role for education, both of newcomers to the profession and of experienced practitioners. The final recommendation is that we must aim our future research on goals which are as far ahead of the current state of the art as the current state of industrial practice lags behind the research we did in the past. Twenty years perhaps?