

SUPPLEMENTO N. 1 - VOL XII (1975)

CALCOLO

Rivista trimestrale pubblicata dal
CONSIGLIO NAZIONALE DELLE RICERCHE (CNR)
con la collaborazione della
ASSOCIAZIONE ITALIANA PER IL CALCOLO AUTOMATICO (AICA)

PROCEEDINGS OF A
MEETING ON 20 YEARS OF COMPUTER SCIENCE
PISA, JUNE 16 - 19, 1975

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE DEL CNR
VIA S. MARIA 46 - 56100 PISA - ITALIA

Meeting on 20 years of Computer Science, Pisa, June 16-19, 1975

Computing System Reliability: Prospect and Retrospect

B. Randell

THE UNIVERSITY OF NEWCASTLE UPON TYNE COMPUTING LABORATORY

Well over a hundred years before the development of the modern electronic computer, most of the fundamental concepts of computers were invented by Charles Babbage, who then spent much of the rest of his life in an unsuccessful effort to turn his ideas into reality. Babbage was motivated by a concern for the accuracy of published mathematical tables, and gave much thought to the problems of ensuring the reliability of the results that his planned Analytical Engine would produce [1].

Similarly, the problem of achieving adequate operational reliability was one that all the modern pioneers of the computer had to face directly. For example, the relay computers developed by George Stibitz and his colleagues at the Bell Telephone Laboratories were notable for the extent to which they checked their own activity, and could safely be left to function unattended over long time periods. The problem was particularly difficult for the developers of ENIAC, the first general purpose electronic computer, due to the hitherto low reliability of electronic valves - some critics predicted that in view of the number of valves called for by the design of ENIAC, it would suffer valve failure at a rate far exceeding that at which the failed valves could be located and replaced.

In the event the problem was solved brilliantly by John Eckert and the rest of the ENIAC project team. Thus ENIAC provided an immense step forward in the provision of devices which could carry out extremely lengthy and complex pre-designed activities, without human intervention. However at first the 'pre-designed activities', to which these early computers were applied were the performing of scientific calculations, and the reliability of the computers was not quite good enough to encourage the unquestioning acceptance of their output. Thus comparatively few people were involved at all directly with computers, and the extent to which they relied on the continuity and the correctness of the functioning of their computers was quite limited.

As the size, performance and reliability of computers has increased, so has the complexity of the tasks that they are used for, and the extent of the reliance placed upon them. This reliance may take many forms. For example there may be no satisfactory manual alternatives to the computer-based procedures, large numbers of non-specialist users may be in direct contact with the activities of the computer, or the external activities that are being influenced or even controlled by the computer may be extremely time-critical.

The hardware reliability thus needed has been sought from increased component reliability, resulting from improvements in technology, and where component reliability has not been considered sufficient, from so-called fault-tolerant system designs. These designs have incorporated such techniques as majority voting, parity checking,

multi-processors, etc. It is indicative of the level of reliability that has been achieved that, although the general public may still be being misled, it is clear that the majority of stories in the mass media about computer errors are in fact due to human errors, generally in the specification and implementation of computer programs, or in the input of data to the computer.

There is indeed a tendency to regard hardware design as a solved problem, and to refer to current difficulties in the implementation of complex computing systems within budget and schedule constraints, and to performance and reliability specifications, by the term "the software crisis" [2]. However this terminology is quite misleading. The problem is that we are trying to produce designs for processes which will be able to cope with extremely complicated situations, processes which are specified with such detail and accuracy that the process can be carried out by an automaton. It is this requirement for absolute accuracy which can make the problem much worse than that of, say, the design of an aircraft. Such a design results in an immense set of blueprints, specifications, etc., to guide a manufacturing process which is far from fully automatic, and provides ample time and opportunity for human supervision and response to unexpected situations.

This problem of design complexity can occur whether one is producing a design which is to be expressed as a computer program, or instead by the layout of minute amounts of silicon in LSI chips. Hardware designers have in the past been subject to technological constraints such that the logical complexity of the computer designs they have attempted has usually been far less than that of the programs which will be executed by these computers. Indeed an important criterion for choosing a hardware rather than software implementation of an algorithm was (and is) its comparative simplicity and hence the low probability of its requiring modification. However the advent of LSI is removing much of the technological constraint, and hardware designers are approaching the trap that software designers are still trying to escape, the trap of producing designs of a complexity which cannot be fully mastered. The unreliability that is caused by unmastered design complexity, of either hardware or software, seems much more dangerous than unreliability due to faulty components in hardware systems whose logical design is basically sound. The type, seriousness and frequency of system failures caused by faulty components can be the subject of plausible probability estimates, and techniques for employing additional hardware in order to mask the effects of such failures are comparatively well established. But this is not the case with failures caused by residual design errors in an apparently fully operational system, errors possibly due to an inability to foresee all the different sets of circumstances the system would have to deal with, or to realise that the design was inadequate to cope with foreseen but previously unencountered circumstances.

Much progress has been, and continues to be, made in the development of techniques for specifying the functions to be carried out by a system, and for producing designs whose correctness with respect to given specifications can be the subject of believable a priori arguments, exhaustive testing being totally impractical for systems of any real complexity. However few would predict if and when such techniques could be developed sufficiently to enable the designers of systems such as the present-day large real-time systems to be

justifiably confident that their systems are completely free of design errors. It is therefore interesting to examine those (comparatively few) systems, intended for extremely complex tasks with very high reliability requirement, which are widely regarded as having been successful. Such systems reveal themselves to have been designed so as to minimise the level of reliability needed from the computing system alone, both by limiting the tasks which are automated, particularly those with stringent real-time constraints, and by retaining the possibility of human judgement being used to supplement or even override the activity of the computing system [3].

Most present-day research which is aimed towards the problems of achieving very high levels of reliability from completely automated complex software systems concentrates on the problem of ensuring that the system is free from design errors [4]. The alternative (or rather complementary) approach, that of incorporating provisions into the system for detecting and recovering from failures which might even be due to residual design errors is being undertaken, for example, at Newcastle, under the sponsorship of the Science Research Council of the United Kingdom [5].

A problem with such a goal is that it involves increasing the functions which are to be performed by the system. This will increase the size of the software system, but must not increase its overall complexity if it is to provide a nett gain in reliability - complex provisions for detecting and recovering from failures caused by residual design errors would be all too likely to harbour design errors themselves!

This work has led us to the development of an overall approach to the structuring of complex computing systems, and to the design of a novel set of techniques for providing system error detection and recovery. The general approach, though designed for use against residual design errors also has, we believe, potential relevance to the design of facilities for the more conventional problem of recovery from failures caused by, for example, incorrect data, or hardware malfunctions. The current stage of the work is that a first prototype system is largely implemented, and plans are well under way for a more advanced system. However these systems are meant merely as test-beds for our ideas, and much work remains to be done to assess the practicality and range of applicability of our approach, and the extent to which it can provide cost-effective improvements in the overall reliability of complex computing systems.

To sum up, the size and power of present-day computers tempt people to undertake the design of immensely complex systems. The speed and reliability of these computers have led to their being used in extremely demanding time-critical environments. However the potential seriousness and unpredictability of system failures arising from residual design errors should be used as an argument against accepting overambitious system specifications. This applies particularly to complex real time systems in environments where serious consequences, perhaps even the loss of life, could result from failure to achieve correct and continuous system functioning. Recent research into programming methodology and program proving are increasing our ability to master rather modest levels of complexity, and are thus of considerable value. But in a "fault-intolerant" system [6] which depends on correctness rather than on internal error detection and

recovery, one cannot assume that the seriousness of the failure that could be caused by a residual design error is in any way related to the difficulty of finding and removing that design error. Even if, as design errors are corrected, such a system is experiencing increasing periods of failure-free operation, this provides little evidence that future failures will be minor. Much more research is needed therefore into methods for organising complex systems so as to minimise the consequences of unexpected failures, and into determining the best balance of manual and automated techniques for achieving adequate levels of fault tolerance. Thus, just as the early pioneers of computing struggled to extend the size of their computers to the limits imposed by the reliability of their hardware technology, so in future we will struggle to extend the size of our systems to the limits set by our comprehension of such complex systems and the reliability constraints which such complexity implies.

References

1. C. Babbage. On the Mathematical Powers of the Calculating Engine. Manuscript dated 26th December 1837. (Printed in The Origins of Digital Computers: Selected Papers (B. Randell ed.) Springer Verlag, Berlin (1973) pp.17-32.)
2. P. Naur and B. Randell (Eds.) Software Engineering. NATO (January 1969).
3. J. Aron. Apollo Programming Support. Software Engineering Techniques (J.N. Buxton and B. Randell, eds.) NATO (April 1970) pp.43-48.
4. Proc. International Conference on Reliable Software, Los Angeles, 21st-23rd April 1975.
5. B. Randell. System Structure for Software Fault Tolerance. In ref. 4 above, pp.437-449.
6. A. Avizienis. Fault Tolerance and Fault-Intolerance: Complementary Approaches to Reliable Computing. In ref. 4 above, pp.458-464.