Formal Aspects of Computing (1989) 1: 5-18

Formal Aspects of Computing

Development Methods for Trusted Computer Systems

Martyn Thomas
Praxis Systems plc, 20 Manvers Street, Bath BA1 1PX, United Kingdom

Key words: Safety critical computer systems; Trustworthiness; Good practice; Formal methods

Introduction

Should we trust computers, and if so, to what extent? This is one of the most important questions facing our industry. It will be asked more and more often, and it is highly desirable that we, the professionals, agree on our answer and are able to justify it.

Whether or not we *should* trust computers, the reality is that we - society - behave as if we trust computers more every year. We trust computers with our privacy, with our money, with the defence of our country, and with our lives. Few of us, perhaps none of us, realise how widespread is the trust we place in computer systems and in the men and women who design and implement them.

In this paper, I shall show that critical computer systems are already widespread and that their use is growing rapidly. I shall describe the main reasons why computers are used for these applications, and consider some spectacular failures and whether or not we should be concerned. I shall then review a few technical approaches to developing critical systems, and attempt to present an overall picture of the state of the industry. Finally I shall suggest that some urgent actions are needed.

Much of this paper involves safety, and this raises a great difficulty. It is not my wish to create public alarm, or widespread fear of computer systems in safety-related applications. Indeed, I want to say at the outset that I do not believe that such alarm or fear is justified. Nevertheless, sensational press reporting of previous public discussions of safety issues has caused problems, which

Correspondence and offprint requests to: M. Thomas, Chairman, Praxis Systems plc, 20 Manvers Street, Bath BA1 1PX, UK.

have greatly reduced the extent to which computing professionals are willing to reveal details of safety-related systems - and especially of failures in such systems. Yet, if we are to decide the degree of trust which can be placed in computer systems, and if we are to understand the contribution which different development techniques bring to the safety of a system, we must have information and the freedom to discuss it.

The examples of failures which I give are not to be taken as criticism of any specific person, company or product.

Some Trusted Systems

I describe below a few of the applications where computer systems are implicitly trusted by society. In each area, the incorrect behaviour of the computer system would significantly increase the risk of a failure, possibly with serious consequences. In most of the areas below, the computer system is only part (and often a small part) of the total system. Often, the designers of the system have provided back-up mechanisms to minimise the effects of a failure in the computer system. I consider later a few of the important ways that system designers seek to ensure the integrity of their systems; the purpose of this section is to illustrate by example the degree to which we already trust systems which contain computers, and the possible consequences if that trust is unjustified. I also attempt to show the rate at which new, trusted computer systems are being developed.

Civil Aviation

Computers are used extensively in civil aviation. On the ground, they are used for a wide range of purposes, from controlling airfield lighting to the air traffic control systems which provide the controllers with radar plots and aircraft details. A new computer system will soon be introduced which will automatically detect and report on potential collisions between aircraft in the controlled airspace over London. It is believed that this will permit the London airspace to become even more congested without compromising its excellent safety record.

In the air, computer systems are used (inter alia) for navigation, engine control, and automatic takeoff and landing. In the Boeing 757 the instrument panel has been replaced by VDUs, with computer-generated instrument displays. In the Airbus A320 (which has recently come into service) the computer systems have full authority over the aircraft's controls, in three dimensions: the pilot is not moving the ailerons directly, for example, but through a computer which will override the pilot if the control movements are beyond those it has been programmed to accept. The first Airship with a full-authority computer control system was demonstrated in October 1988.

On the A320 Airbus and the next version of the Boeing 747, the three-man crew has been reduced to two by eliminating the role of the flight engineer. To a first approximation, computers are being given the flight engineer's job of monitoring the vital systems.

Civil airliner systems are among the most safety-critical, as a serious accident to a wide-bodied jet could kill 300–400 people.

Medicine

Computers are increasingly used in medicine in the following areas:

Patient records. Data such as blood-types, allergies and test results are stored on computers. This raises issues of privacy as well as safety, if records are retrieved incorrectly. The privacy issue in particularly topical, as the number of tests for HIV (the virus which can lead to AIDS) increases.

Prescription. General practitioners have started to rely on computer systems to identify potential drug interactions when prescribing.

Patient monitoring. Computers are used to monitor vital functions of patients in operating theatres and intensive care units. If the values go outside limits, an alarm is raised to attract the attention of staff.

Therapy. Computers are becoming widespread in equipment as diverse as heart pacemakers and radiotherapy machines. Computer systems sometimes control the rate of drug administration to patients receiving intravenous drips.

Diagnosis. Computers are used for x-ray imaging, body scanners and similar equipment.

Typically, an accident would only kill one person at a time, although a recurrent fault might kill several before it was corrected.

Road Vehicles

Computer systems are found in antiskid brakes, where a system failure could cause brake failure. Computers are also being introduced into self-adjusting suspension systems, where a failure whilst cornering could cause loss of directional control. Engine management, electronic ignition and fuel injection are other common applications. Future developments already proposed include drive-by-wire; head-up displays; and collision detection and avoidance, where the computer could be given authority over the car's speed and direction.

A typical accident could kill or injure the occupants of one or several cars, and possibly other road users.

Water Industry

Computer systems are used for such purposes as monitoring the levels of lakes and reservoirs, controlling dams, and controlling water treatment, including the addition of fluoride.

Defence

Computer systems are used widely for defence purposes, including use of flight control of aircraft, for missile guidance, for arming weapons, for weapons management and control on aircraft, for scheduling the safe movement and storage of explosives and toxic materials, for cryptography and cryptanalysis, for surveillance and image enhancement and for battle management.

Nuclear Reactors

British civil nuclear reactors do not use computers for reactor protection (although military reactors are believed to do so). This will change with Sizewell B, which will be the first British civil reactor to have a computer-controlled protection system. This system has a hardwired backup for the failures which are presumed to be most likely.

A worst-case failure would cause a few hundred short-term deaths, many more long-term, and severe environmental damage.

Emergency Services

Police and other emergency services increasingly use computers for command and control, for example in dealing with civil emergencies. System failure during such an emergency could cost many lives.

Finance

Computers are used widely in trusted financial applications, ranging from automatic bank teller machines and electronic funds transfer, to electronic trading in securities and other markets. A worst-case failure could possibly cause the collapse of one or several financial institutions. It has been suggested that the world stock-market crash in October 1987 was exacerbated by automatic trading by "portfolio insurance" programs.

Chemical Plant

Computer systems are widely used for controlling highly exothermic chemical reactions. In some cases even the emergency trip systems are computer-based. The situation in the chemical industry differs greatly from that in civil aviation as many of the plants are owned by small companies with limited resources, and the control systems and safety systems employed differ widely from plant to plant.

The degree of injury which would be caused by a worst-case accident depends substantially on the location of the plant. A severe accident in the UK might kill 30 people.

Computer systems may also be used to control the chemical processes which produce drugs, pesticides and fertilisers. If these processes were to go wrong and not be detected, the results could affect very many people. An incident of this nature has apparently recently occurred [Com88]: an input error in preparing the master batch documents for a pharmaceutical computer apparently led to 6400 bottles of cough medicine being produced with ten times too much of one constituent. The bottles were recalled; but our concern should perhaps be directed at the data-validation routines in the computer system.

Industrial Control and Robotics

Computers control a wide range of industrial equipment. Some examples are hydraulic presses, machine tools, unmanned vehicles, guillotines, and robots for

welding, assembly, and painting. A typical accident could lead to the loss of a limb, or injury (perhaps fatal) to one person.

Railway Signalling

Computer systems are used to replace the complex systems of relays which interlock signals and points to ensure the separation of trains. As with road traffic-lights, the systems are designed to fail safe, with all signals set to stop. A worst-case accident could kill tens of people.

Other Areas

This brief survey is far from exhaustive. Among the areas not covered are the secondary, design applications, where a computer is used to design safety-critical structures such as bridges, the Severn Barrage, or the structure of a nuclear reactor. The discovery of an error in a design program for nuclear reactors led to the US Nuclear Regulatory Commission shutting down five reactors [ACM79].

Conclusion

Computer systems are currently used in a very wide range of applications where the consequences of failure (of the system of which they are part) could be severe. The use of such systems is increasing rapidly, as is the complexity of the applications (and, consequently, of the computer systems). Increasingly often, the computer system has no hard-wired, low-complexity backup system. I believe it is important that we are able to monitor the growth in the use of computer systems for critical applications, and to be confident that we are not inadvertently creating areas of unacceptable risk. In a moment I shall consider whether there is evidence which could justify a degree of concern about the way critical computer systems are developed and used. Before that, however, I want to consider briefly the reasons why computer systems are being introduced into critical areas, as this provides a valuable context for later discussions.

The Benefits of Computers

As we have seen, computers are in use in a wide range of critical application areas, and this use is increasing in range, volume and complexity. There are three interrelated reasons for this: economics, functionality, and safety.

Computer systems are very powerful: they are able to monitor a large range of inputs, perform complicated calculations, and drive many outputs, and all this at very high speeds. For this reason, there are many applications which would simply be impossible without using computers. This power also allows for finer control: if you can monitor more variables and take action quickly on what you find, you can control your process so that it is nearer its optimum efficiency. With cruder control mechanisms you may have to allow much greater safety margins, or to tolerate many more occasions when the controlled process is shut down by the safety system, either of which could cost a lot of money.

Computer systems are relatively cheap, so the cost of building the control and protection systems may dictate the use of computers. They are light and small and consume little power.

Integrated circuits can be very reliable components, under the right conditions. They can even be made radiation hardened.

A Health and Safety Executive workshop [UKHnd] has concluded that the use of programmable systems can improve safety. Their main reasons were that the programmable system could monitor large numbers of process variables to detect fault conditions, that they could enable operation staff to work further away from hazardous areas, and that they could reduce risks from human operating errors.

It can be seen that there is a balance between cost and safety. Safety can always be improved at increased cost, but the point arrives where the risk is acceptable and the cost of improvement is too high. In considering the social implications of computers we should look critically at such areas, where safety margins may be eroded for economic reasons, and replaced by arguments that computer technology is providing equivalent, or greater, levels of safety.

Society has a right to ask how we can be certain of these levels of safety, and what steps we take to ensure that every such system is developed to the necessary high standards. These are issues to which I return later.

Is There Cause for Concern?

Developing critical systems is a difficult, time-consuming, and expensive process which requires great skill and experience. The control system cannot be considered in isolation from the controlled process (I shall give an example shortly of the sort of problems which can occur if it is). The potential hazards have to be identified and, as far as possible, eliminated by careful design of the whole process. Only when this has all been done can we draw up the specification for the computer system.

This specification may be wrong. It may inadequately reflect the task which, *with hindsight*, we would have wished the computer system to perform. If the specification is contradictory, we should be able to detect it with analysis techniques. If it is incomplete, we may have enough information to discover it (for example, it may not specify the required behaviour for all possible input values). If, however, the specification is functionally wrong, we can only hope to realise it by animation or testing (or by some other form of review by experienced humans). Otherwise our implementation will seek to satisfy the given specification as accurately as possible, with the result that the computer system will be "correct" (with regard to the specification) but the overall behaviour will be wrong.

When society asks how safe a system is, what can we say? Unfortunately, very little with any certainty. Testing will tell us almost nothing (since testing will exercise a very small percentage of possible system behaviours). Indeed, if testing has found any errors at all, there may be justification for claiming that the system is likely to contain too many errors for it to be put into use. This is not an argument for omitting testing of critical systems, merely for avoiding using the results to attempt to quantify the safety of the system.

It is rarely practical to test a critical system under real conditions of use for the necessary length of time to achieve statistically-significant results. As soon as we move away from real-world testing, we start to introduce unquantifiable

probabilities that we are doing something wrong. We cannot usually get a formal specification of the real world requirement, because guaranteed models of the real world are not available. It seems to me that anything we do thereafter to derive a numerical probability of failure for a system will be built on sand. At best, we may be able to *estimate* a probability, but the likely error in the estimate will be orders of magnitude too large for critical systems.

Traditional methods of estimating failure probabilities for critical systems derive from low-complexity, hard-wired systems. In these, the probability of hardware component failure is relatively easy to quantify, and it is probably justifiable to assume that the hardware failure rate greatly exceeds the probability of failures from design errors. In practice, unquantifiable failure probabilities (such as the design error rate) are ignored in safety and reliability calculations.

This approach breaks down with programmable systems. The hardware is generally very reliable, and the software is totally reliable (in the sense that it will continue to display the same functionality it had when it was first designed). The most likely causes of failure are specification and design errors - but how can these be quantified? It is my belief that they cannot - at least, not to the required precision - and that pragmatic, non-numerical techniques must be employed to provide assurance of adequate safety.

Some Failures

For obvious reasons, the developers and owners of critical systems are usually unwilling to discuss failures which have occurred. Nevertheless, some failures have been described and they provide useful insight into the ways in which critical systems go wrong. (The richest source of descriptions of critical system failures or "near misses" is an ACM forum [Neund].)

I have previously described a number of reported failures, as illustrations of the sort of problems which have occurred with safety-critical systems [Tho88]. I conclude that despite the skill and experience of the teams developing critical systems, potentially serious problems still occur, and that we cannot assume that the unquantifiable errors in complex systems development will be detected by current techniques, or that they are so infrequent that they can be ignored.

Many of the failures described [Tho88] result from incorrect specifications, where the designers of the computer systems did not recognise the real-world implications of their specification and design decisions. There is also evidence of implementation and component errors. Some description of these problems comes from the Computing Division of RSRE Malvern, and has been reported by Cullyer [Culnd]. In this lecture, Cullyer reported experience with static code analysis techniques.

Static code analysis is a powerful technique for revealing the potential behaviour of a computer program. It involves processing the source text (in some well-defined language) to convert it to a labelled, directed graph which represents the possible control flow and data usage. This graph is then reduced where possible, identifying anomalies such as unreachable code, or data which is used before it is initialised. The simplified graph is automatically annotated with algebraic expressions which reveal the functional dependency of each output on the inputs. This can then be compared with the intended relationships between outputs and inputs to reveal errors. There are at least two commercial suites of programs which provide versions of this technology.

Cullyer reports [Culnd]:

> It has been demonstrated using [commercially available tools] that programs written for high-integrity equipment may contain a significant number of errors when first delivered for operational use.
> Taking a broad average over the analysis done by RSRE and other establishments during 1982–1985, up to 10% of program modules or individual functions were shown to deviate from the original specification in one or more modes of operation. Such discrepancies were found even in software which had undergone extensive checking on multi-million pound test rigs. Many of the anomalies were too minor to have any perceptible effects, e.g. a discrepancy of one part in 32,000 in some computation using 16-bit arithmetic. However, about 1 in 20 of the functions which static code analysis had shown to be faulty, i.e. about 1 in 200 of all new modules, proved to have errors which had direct and observable effects on the performance of the system being controlled. For example, potential overflows in integer arithmetic were detected, involving a change in sign of the direction of deflection of an actuator (e.g. "turn left" when the correct action is "turn right").

RSRE also investigated the suitability of commercial microprocessor chips for use in safety-critical applications. They discovered, Cullyer reports [Culnd]

> ... the following issues
>
> a. Modern microprocessors are so complicated that even experienced designers may misunderstand some aspects of their behaviour;
>
> b. Many of the devices on sale have design errors at silicon level;
>
> c. Microprocessor handbooks ... describe the operation codes for chips in a manner which sometimes is ambiguous and may at worst convey totally wrong information;
>
> d. The designers ... change the internal details of the devices several times per year, resulting in chips of differing functionality from one production batch to another; ...
>
> ... these uncertainties could not be tolerated. In particular, the control of nuclear reactors ... requires a scientific and engineering certainty which is hard to achieve with commercial microprocessors.

In consequence, RSRE has invented and developed a new 32-bit microprocessor, VIPER, with a mathematically formal specification and a rigorous development path [Cul87].

Critical Systems Development

It is beyond the scope of this paper to survey all of the technical approaches currently employed in developing critical computer systems (see Leveson [Lev86] for a valuable survey), but it is useful to look at some general approaches. Note that this section deals with the development of the computer system, although, as we have seen above, this must be seen in the context of the overall controlled system, since that is where the hazards lie.

Diverse Development

Diverse development is a powerful aid to the critical systems developer. At a basic level, the verification and validation of the system is carried out by a different team from the developers. Beyond this, multiple versions of the system may be implemented and run in parallel, with logic which requires that the multiple channels agree on the output for a given input. The multiple channels may be developed by different teams using different models of microprocessors, perhaps implemented in different fabrication technologies, programmed in different languages and cross-compiled from different host computers. In extreme cases the channels may be diverse from beginning to end - for example the danger threshold in a chemical reaction may be able to be detected by either a temperature sensor or a pressure sensor. By using both, one for each of a dual channel system, diversity can actually start at the physics of the sensor system.

The rationale behind diversity is that separate teams are extremely unlikely to make identical or related mistakes - so it is unlikely that both channels will fail at the same time. There are technical problems with the approach (for example, the design of the voting logic; detecting which channel is at fault if two channels disagree; and ensuring that a fault in one channel is not masked so that when the second channel fails the whole system fails) but these can be overcome, and diversity is required and used in many critical applications.

Unfortunately, some assessors have assumed complete independence of the channels when calculating safety or reliability figures, so that two channels, each with 10^{-4} probability of failure, are assumed to give 10^{-8} probability of failure in parallel. This suffers from two difficulties: how we know that the channels are each 10^{-4}, and how we can be sure that there are no "common mode" failures, which will cause simultaneous failures of each channel.

Redundancy

Redundancy is generally used to reduce the impact of random hardware failures, and as I have described earlier, software does not exhibit random failures. Redundancy is often combined with diversity, so that a triplicated diverse system will be implemented with the requirement that two out of three of the channels agree.

Fault Tolerance

Computer systems can be built to be fault tolerant, to a degree. For example, a calculation can be performed and its result checked to be within bounds - if it is outside the bounds the calculation can be repeated using a different algorithm. Such approaches again suffer from technical difficulties. There may be common-mode failures between the alternative algorithms, or between the algorithms and the test of acceptability. There will usually be errors which the acceptance test cannot detect.

It is very good (and widespread) practice to program defensively, checking for situations which "cannot occur" and taking appropriate action. This level of redundancy provides a level of tolerance to design errors and hardware failures.

Fault tolerant designs add significantly to the safety and reliability of critical systems.

Testing

Critical systems are usually tested extensively, with various analyses to ensure that all the program text is exercised at least once, that all branches have been taken in each possible direction, and so on. Such testing may use an artificial test rig - which can have advantages of speed, reproducibility and the ability to exercise particular values of input parameters - or the real world. Test rigs have disadvantages: costs, time potentially wasted on impossible inputs, and common-mode failures with the development specification, for instance. The real world has disadvantages too: the controlled process may not be available for testing against, or the result of a test failure may be potentially too great.

There is much evidence that testing does not detect all significant faults [Culnd] (and it is evident that, as testing can seldom if ever be exhaustive, testing will only show the presence of errors, not their absence). There is also the problem of what you should do when an error is detected in testing. Logically, the corrected system is a new system which should be completely revalidated - but this is impractical if a large number of faults is detected during development, testing and early use.

Formal Approaches

Developments in applied mathematics [[Hoa69, Dij76, Jon86] have made it possible to apply mathematical analysis to computer system specifications and implementations, both hardware and software. Of course, there are restrictions, some of them severe. The system specification and development must be recorded in a language or notation which is mathematically well-defined. The mathematical techniques are only well-developed at present for sequential systems and for limited types of computer hardware. The systems must not be too large or complex, or they exceed the capability of current mathematicians to handle them. Staff need mathematical training. There can be specification errors which do not show up as incompleteness or inconsistency.

Nevertheless, these techniques show great promise, and they are already in growing use. The UK Ministry of Defence has said that such techniques will become mandatory for military safety-critical software [Defnd].

The mathematics which underlies these methods is mathematical logic (discrete mathematics). It is the mathematics which models precisely the nature of digital systems, and it offers a scientific basis for the development of computer systems. The successes with static code analysis and with the VIPER development, both of which rely on mathematical methods, has been described earlier. These methods are the way of the future, offering far greater certainty throughout the development processs.

However, formal methods do not address the whole problem, because there is clearly still the problem of understanding the controlled process, developing an adequate specification, analysing the system for potential hazards and the consequences of failures and so on. And, of course, even a mathematical analysis or proof has some probability of being incorrect [Fet88]. Formal methods should

be viewed as a way of strengthening the development process, by formalising many of the informal processes which are currently employed. It must be stressed, however, that current informal methods can be extremely important in controlling risk, and that changes in methods should be made cautiously.

Quality Assurance

The whole development, operation and maintenance of a critical system will properly be carried out within the procedures of a rigorous quality control and quality management system. This will require that agreed procedures are followed, that all important decisions and actions are independently reviewed, and that adequate audit trails and documentation are maintained.

The quality assurance should extend to the development tools, to ensure that no faults are introduced by building systems out of the wrong versions of component parts, and that errors are not introduced in the process of translating source text into binary images in the store of a computer. It is very difficult to ascribe meaningful failure probabilities to these steps.

Review

In my opinion, the current state of critical systems development is that of a sophisticated craft industry. Standards are variable - some are very high, other lamentable. There is no agreed, common approach to developing programmable, critical systems or to assessing their adequacy for the intended task. There is no scientific basis on which we can assign to complex systems of hardware or software the often-quoted, very low, probabilities of failure. There is no authoritative source of data on the number or range of critical systems in use or under development, no public record of their complexity or of their failure-rate in use. In short, there is an absence of control and a lack of information.

Yet there is evidence of every-increasing reliance being placed on computers. There is also good reason to believe that computer systems, properly deployed, can provide increased levels of safety compared with earlier approaches, and bring substantial economic benefits.

It is evident that system reliability and safety are a function of system complexity. This means that there must be limits to the complexity of control systems which can reasonably be adjudged suitable for use in critical applications. We need to agree those limits, and devise some mechanism for ensuring that important systems do not attempt to go beyond what we believe to be reasonable.

We need to agree the safety and reliability levels which it is reasonable to claim for systems which have used specified development practices, and to agree the level of risk which it is acceptable to entrust to systems at each level.

Space has prevented me from reviewing the state of national and international standardisation; had I done so, I would have presented a picture of unnecessary diversity between industries and nations. This is a legitimate cause of concern. Someone living in Southern England is as much at risk from a French nuclear accident as from a British one, yet the safety standards seem very different. We should work through the industry to achieve harmonisation in these important areas.

There is good evidence that formal methods, including static code analysis, reveal software errors which escape detection using conventional methods, yet

most safety-critical systems are still being developed without using these techniques. Indeed, some safety-critical systems are being installed without any formal hazard analysis and, if frequent anecdotal evidence is to be believed, some developers even ignore rudimentary principles of software engineering. Computer science and computer technology are far from mature, yet society is behaving as though it had limitless faith in computer systems.

I believe that we have a duty to try to ensure that all safety-critical systems are developed to agreed high standards. Safety is a legitimate area for society's concern and action, and so this is where we should start. (Perhaps we shall be able to turn our attention to other classes of critical system later.) If we are to achieve high minimum standards, there is much work to be done. There is certainly no cause for panic, but the work is urgent and should not be delayed.

What Needs to be Done?

Exploitation of Current Best Practice

Different companies and different industries follow widely differing practices when developing safety-critical computer systems. As a result, the safety of two systems may be very different although the consequences of failure may be the same.

There is much in common in the requirements for safety-critical computer systems in different application areas, which means that very similar practices are appropriate in all industries. The measure should be the consequences of a failure - this should set the level of risk which is considered acceptable, and this in turn should determine the development practices to be adopted as a minimum.

It is important to be able to demonstrate compliance with these minimum standards - a standard which does not support testing for compliance is of very limited usefulness. This would seem to require that the standards are prescriptive - that they actually lay down what has to be done and the way it should be done. The standards must also be kept up-to-date; in this fast-developing technology, this will involve a process of continuous review and improvement.

Recognition of the Fundamental Role of Mathematics

We need to introduce mathematical rigour wherever we can throughout the development process, to supplement and strengthen the techniques which are currently employed. Engineering judgement supported by calculation and logical argument is preferable to engineering judgement without such support. We need to cut through the debate about the merits of formal methods as alternatives to currently accepted methods and concentrate on setting the best of current development methods on a sound, formal base.

Setting the Limits of Safety

There will continue to be economic pressures to control ever more complex processes, using every more complex computer systems. To protect society, and

to remove intolerable pressures from individual engineers, we should define the safety levels which are required for systems of a given criticality, and define (perhaps arbitrarily) the safety levels which can be claimed for a given implementation technology. This will enforce minimum standards, as described a moment ago, and it will also set limits on the complexity of applications which can be attempted at any time.

Licence for Release to Service

To complete the control over safety-critical systems, we then need a mandatory licence for release into service of any system above a defined level of criticality. There are industry-specific precedents for this - telephones have to be licenced before they can be connected to the public network, because of the damage that faulty equipment could do; cars need specific or type approval to go on public roads; the Nuclear Installations Inspectorate licences nuclear reactors, the Civil Aviation Authority (CAA) licences aeroplanes, the Government Communication HQ licences computer systems which handle classified information....

Given the common requirements for safety-critical computer systems, it should be a great help to the existing regulatory authorities to have agreed common criteria for acceptability and licencing. In addition, mandatory licensing would catch all the areas where systems are not currently explicitly regulated, which would be wholly beneficial.

Collection of Information

If the mechanisms I have just described were set in place, it would create a framework which could enable us to collect statistics about the numbers of safety-critical computer systems in use, their criticality and the methods used in developing them, and the sorts of failures which occur and their frequency. This information would be extremely useful in guiding the development of future standards.

Again there is at least one precedent. The Air Navigation Order requires that any accident to an aeroplane be reported to the CAA, so that it can be analysed and lessons drawn which could improve safety for all.

Harmonisation

Whenever new controls are introduced there is a danger that they create non-tariff barriers to trade, with consequential economic damage. For this reason, it is highly desirable that the licencing procedures are agreed internationally, if this can be achieved, and work is currently being undertaken to create international standards [IECnd].

Conclusion

Well-designed computer systems can be safer than hardwired alternatives, and computer systems can control processes which are too complex for hardwired solutions, or where the hardwired solution is uneconomic.

So should we trust computers? Within limits, yes - but we do not know where those limits are, or whether many systems have already crossed the line. Although most current systems are adequately safe, there are economic and marketing pressures which will lead to even more complex systems in the future. The rate of growth in both the number and the complexity of safety-critical systems seems to be high, and urgent action is therefore needed to create a framework within which we can exploit the power of computers safely and economically.

Mathematical methods have a fundamental role to play in improving the certainty of computer systems development and the safety of trusted computer systems. This journal, by providing a forum for technical debate and the publication of refereed results, will make a valuable contribution in this most important process.

Acknowledgement

This paper is a shortened, updated and amended version of the BCS/Unisys Annual Lecture [Tho88].

References

[ACM79] *ACM Software Engineering Notes*, 4, 2 (1979).

[Com88] *Computing Australia*, 13 June 1988.

[Culnd] Cullyer, W. J.: *Should We Trust Computers*? Lecture to the Society for the Application of Research, Cambridge, UK. RSRE, St Andrews Rd, Gt Malvern, Worcestershire WR14 3PS.

[Cul87] Cullyer, W. J.: Implementing Safety-Critical Systems: the VIPER Microprocessor, In: *Proc. Workshop on Hardware Verification*, Calgary, Canada, January 1987.

[Defnd] *Defence Standard 00-55.*

[Dij76] Dijkstra, E. W.: *A Discipline of Programming.* Prentice-Hall, 1976.

[Fet88] Fetzer, J. H.: Program Verification: the Very Idea. *Communications of the ACM*, 31 (1988).

[Hoa69] Hoare, C. A. R.: An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12, 576–580, 583 (1969).

[IECnd] International Electrotechnical Committee [IEC] WG65A, TC9 and TC10.

[Jon86] Jones, C. B.: *Systematic Software Development Using VDM.* Prentice-Hall, 1986.

[Lev86] Leveson, N.: *ACM Computing Surveys*, 18, 1986.

[Neund] Neumann, P. (ed.): *ACM Forum on Risks to the Public in Computers and Related Systems.* Extracts reprinted in *ACM Software Engineering Notes.* The Forum is published electronically on USENET as newsgroup comp. risks.

[Tho88] Thomas, M.: *Should We Trust Computers*? BCS/Unisys Annual Lecture 1988, available from Maureen Murphy, BCS, 13, Mansfield Street, London W1 (£10).

[UKHnd] UK Health and Safety Executive (unpublished).

Received July 1988
Accepted in a revised form November 1988 by D. J. Cooke