The Role of Data in Safety-Related Systems

Neil Storey, B.Sc., Ph.D., FBCS, MIEE, CEng; School of Engineering,
University of Warwick; Coventry, CV4 7AL, UK

Alastair Faulkner, M.Sc., MBCS, CEng; CSE International Ltd.; Glanford House,
Bellwin Drive, Flixborough DN15 8SN, UK

Keywords: data, safety-related systems, software, safety

Abstract

When considering the production of a computer-based system, it is common to partition the arrangement into *hardware* and *software* elements. The software part of such an arrangement is taken to include both the *instructions* that are executed by the processor, and the *data* that is used and produced by these instructions. In some cases, a large amount of data forms an essential element within the system and plays a vital role in ensuring its correct operation. In such situations it is perhaps more appropriate to partition a system into *hardware*, *software* and *data*, to allow appropriate importance to be given to each element. This is particularly appropriate in safety-related applications where the safe operation of the system is dependent upon the correctness of the data. Unfortunately, the various standards and guidelines that relate to the production of critical systems are concerned almost exclusively with methods of ensuring the 'safety' of the hardware and the executable portions of the software of a system, and say almost nothing about the nature, production or testing of the data.

This paper looks at the nature of data in a range of safety-critical applications. It then considers the characteristics of data faults and proposes a systematic approach to tackling them.

Introduction

The safety of a complex, safety-related system is a property of the complete system, rather than of individual components. Therefore when designing such systems, attention must be paid to all its elements and aspects.

Over the years a large number of standards and guidelines have evolved to assist engineers in this task. Most industries that are actively involved in the production of safety-related systems have devised guidelines or standards that describe the hazards associated with the mechanisms or components normally used within that industry. There are also generic standards, such IEC 61508 (ref. 1) that provide guidance across a range of industrial sectors.

Within the various standards and guidelines there is much useful information on methods of identifying and tackling hazards associated with the failure of mechanical, electrical and electronic components. The literature also contains guidance on the treatment of hazards associated with humans within critical systems. However, the largest volume of material in this area is concerned with the identification and treatment of hazards associated with software. Many standards are concerned exclusively with software aspects, for example references 2, 3, 4 and 5, which are concerned respectively with: civil aviation; military systems; nuclear power stations; and railway signalling. The generic standard IEC 61508 covers systems implemented using a wide range of technologies, but again, a large part of its guidance is concerned with software development issues.

Software development receives so much attention within the various standards because this is widely seen as the area that represents the greatest challenge when attempting to produce a dependable and safe system. In computer-based systems much of the complexity resides within the software, and it is therefore not surprising that this is where the problems also lie. However, systems vary tremendously in the form of the software they contain.

One way in which systems vary is in the volume and nature of the data used by the software. Data has quite different characteristics from the executable elements of software, and it is the view of the authors of this paper that the various standards pay insufficient attention to this aspect of the development of safety-related systems.

The Role of Data

All computer programmes make use of some form of data. Most programmes will contain a set of constants that is used within its operation, and will calculate other values as the program is executed. Temporary, or intermediate values, may be stored in specific memory locations or on a stack. All these quantities can be thought of as *data*, which can be either constant or variable.

While all programmes will make use of some form of data, some make much more extensive use of data, which can take various forms. For example:

- Calibration constants
- Device characteristics
- Plant configuration data
- Terrain or topological data

These forms of data are each used to adapt a perhaps standard piece of software to a specific instance, situation or plant. With the widespread use of COTS software we are seeing an increasing number of such applications.

A characteristic shared by each of these classes of data is that they are normally generated quite separately from the development of the 'executable' section of the software. For this reason they may be outside of the normal process of verification applied to the software. However, the task of generating such data may be a complex and demanding one. If one considers, for example, the data used to describe the layout of a railway network, or the terrain surrounding an airport, is clear that the work involved, and the scope for errors, are considerable.

The examples given above represent just a few of the many situations where data plays a major role in determining the safety of a system. In many cases the data forms a significant, distinct, component, that is often generated quite separately from the executable parts of the software. In such circumstances there would seem to be great advantages in considering the data as a separate entity, with its own development requirements and lifecycle.

To see how this can be done we need to consider how the creation of data fits within the overall development process, and to identify the particular problems associated with it.

Risk-based Development Methods

Safety-related industries have devised a range of development methods to help ensure the safety of computer-based systems. These development methods invariably adopt a risk-based approach, which begins by identifying the hazards associated with a proposed system, and then moves on to ensure that the risks associated with these hazards are kept to an acceptable level.

The early stages of hazard analysis are concerned with identifying the major hazards associated with the application itself, irrespective of the way in which it is implemented. This analysis gives an indication of the inherent hazards involved, and will give an early indication of the required safety integrity level (SIL) of the system. The SIL plays a large part in determining the development methods that will be adopted and therefore the overall cost of the project.

The hazards identified, together with the SIL, will greatly influence the overall architecture of the system, and will help to select the methods that will be used to implement the functions of the system. Since safety is determined by the characteristics of the *complete* system, rather than by individual elements, hazard and risk analyses must be applied to *all* elements within the system, be they mechanical components, electronic hardware, computer software, or even the human elements of the system. Unfortunately, these system elements have very different characteristics and tend to introduce hazards in different ways.

One aspect of the analysis of hazards is concerned with the effects of failures of the various components in the system. This is of particular importance in the case of hardware elements, which are subject to random failure. Perhaps a more demanding aspect of hazard analysis attempts to investigate the effect of systematic errors in the system - possibly as a result of design errors. Such considerations will clearly affect the detailed design of individual modules, but may also play a part in determining the overall implementation methods used.

In a particular system it may be possible to implement a given function in a range of different ways, perhaps using very different techniques or even different technologies. This is illustrated in Figure 1, which looks at a range of ways of implementing the same function.

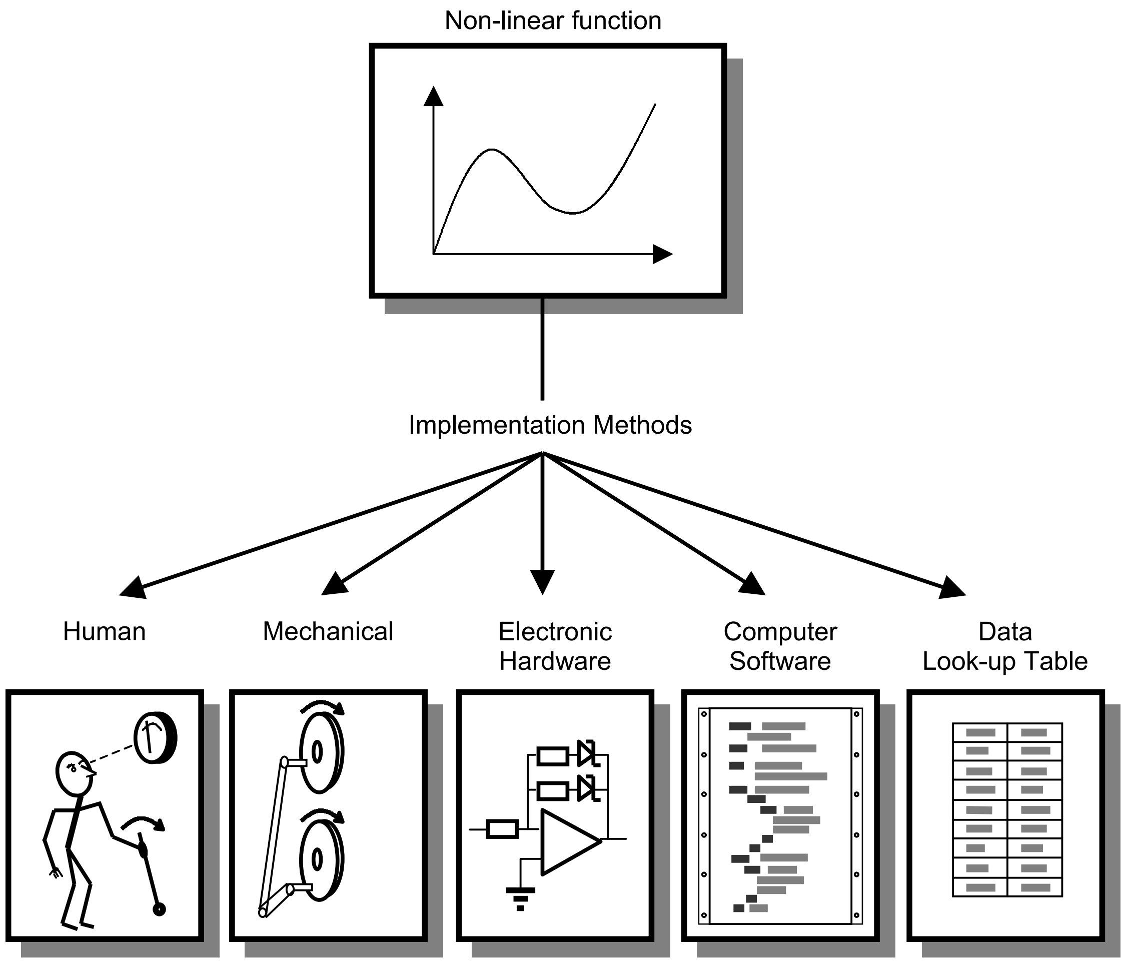

Figure 1 - Some Alternative Implementation Methods

Implementation Methods

Figure 1 illustrates several alternative methods of implementing a simple non-linear function.

The first method shown uses a human operator to produce the non-linear behaviour, using learned skills or a set of defined rules. The second method uses some form of non-linear mechanical arrangement. The diagram suggests a solution based on levers, but alternative methods might use cams, or some form of pneumatic or hydraulic arrangement. In many situations an electronic system is preferred, which could involve simple non-programmable hardware (as in the third example), or perhaps a computer-based system. Where a computer is used to implement the non-linear function, this could be achieved in a number of ways. Perhaps the most obvious approaches are that the function could be implemented using a mathematical model executed within the software (as in example 4), or that the function is stored in a data table representing the relationship between particular input and output values (as in the last example).

An important point to note is that several of these implementation methods combine a range of components or elements. For example, the human operator may well require electronic or mechanical components in order to perform the required task. Similarly, any implementation using a computer will require hardware, software and data in order to function. However, when describing the implementation we would normally use a term that indicates the basic approach taken to produce the required functionality. In the examples in Figure 1 we would therefore describe the various implementation methods as being: human-based, mechanically based, hardware-based, software-based and data-based.

In order to make an appropriate choice between a range of implementation methods, we need to understand the characteristics of each approach. In safety-related applications, one of the key considerations will be the susceptibility of the approach to different kinds of fault.

System Faults

IEC 61508 defines a fault as an "abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function". Faults may be divided into *systematic* and *random* faults, and the nature of the faults will depend on the form of the elements concerned. Table 1 gives examples of possible causes of faults that can effect elements implemented in a range of ways.

From the table it is clear that all forms of implementation are susceptible to specification errors. This is because the specification forms the starting point of any design, and so errors in this document will affect any form of system in a similar way. Other sources of systematic faults will depend on the techniques being used. In a human-based system these could relate to tendencies within operators to make particular kinds of mistake, or an inability of users to respond to a particular set of circumstances in the time available. Such faults are systematic since they will occur each time that the appropriate set of circumstances exists. Systematic faults in hardware-based or computer-based implementations might relate to design errors, or to systematic manufacturing problems. Data within computer-based systems will clearly suffer from specification errors as with other forms of system. Logic would suggest that data would also suffer from other forms of systematic error depending on the methods used to create it. In some cases data is produced manually and would therefore be subject to human error. In other cases it might be produced automatically, leaving it susceptible to the forms of systematic error associated with the techniques used to produce it.

The nature of random faults also varies from one form of system to another. Some forms of human errors are effectively random, in that they cannot be predicted and will not necessarily re-appear in similar circumstances. Mechanical and electronic hardware also exhibit random faults, in the form of random component failures. Software and data however, are more complex issues.

Accepted doctrine on this topic, as reflected in IEC 61508, suggests that all software faults are systematic. This is based on the fact that a fault within a program will be experienced *predictably* each time the program is executed. This line of argument would suggest that measures such as reliability and 'mean time to failure', cannot be applied to software.

Despite the so-called 'predictability' of software failures, often our experience of such things is quite different. When using personal computers for example, we will often experience a software failure when performing a task we have successfully performed before. Such failures often give the *appearance* of being random. This has led some engineers to suggest that some forms of software fault *can* be subjected to statistical analysis (for example, see reference 6).

Table 1 - Examples of the Causes of Random and Systematic Faults

Implementation Method	**Systematic Faults**	**Random Faults**
Human	Specification errors; systematic human errors due to misunderstanding or lack of ability or skill.	Random human errors
Mechanical	Specification errors; design errors or systematic manufacturing errors.	Random component failures
Electronic Hardware	Specification errors; design errors or systematic manufacturing errors.	Random component failures
Computer Software	Specification errors; software design errors or coding errors.	?
Data	Specification errors, data design errors or systematic collection/generation errors.	?

Engineers who subscribe to this view argue that the complex process involved in generating software means that faults can be randomly distributed throughout the code. The location or effects of these faults can not be predicted, and they therefore give the appearance of being random. This suggests that *unknown* software faults *can* be subjected to statistical analysis. However, once a fault has been identified it can no longer be treated in this way. This view, while not universally held, does have considerable support from many engineers working in this area, and has led to work on techniques to predict the reliability of software (for examples, see reference 7). Since, for most purposes such faults behave as if they *were* random, for the remainder of this paper we will refer to them as *random faults*, remembering however, that this *randomness* is illusory.

If one accepts that the executable portion of software is susceptible to both systematic faults and faults that give the appearance of being random, then the same reasoning would seem to suggest that *data* is also susceptible to such faults. This would suggest that unknown data faults may also give the appearance of being random, and may also be subjected to statistical analysis. Again, in this paper we will refer to such faults as *random data faults*, always remembering the limitations of this notation.

Since data is susceptible to faults, it follows that it should be subjected to hazard and risk analysis. *However, experience shows that this is often not the case.* Certainly, the various standards say almost nothing about the application of such techniques to data, and anecdotal evidence from industry suggests that this area is often completely ignored.

Tackling Faults in High Integrity Systems

Much of the process of developing dependable systems is related to dealing with faults - a process of 'fault management'. In non-critical systems good design and the use of quality components can often produce acceptable performance. However, in systems requiring a high integrity additional measures must be taken to overcome the effects of faults. Broadly these may be divided into four groups of techniques:

- Fault avoidance
- Fault removal
- Fault detection
- Fault tolerance.

Fault avoidance aims to prevent faults from entering the system during the design stage, and is a goal of the entire development process. Fault removal attempts to find and remove faults before a system enters service. Fault detection techniques are used to detect faults during operation so that their effects may be minimised. Fault tolerance aims to allow a system to operate correctly in the presence of faults.

From the discussion above it is clear that data is subject to three primary forms of faults, namely:

- Errors in the specification of the system
- Systematic faults in data production
- Random faults in data production.

Before considering ways of tackling these various forms of fault, it is perhaps useful to see how faults are dealt with in other system elements. Tables 2 to 5 give examples of fault management techniques used in other areas of system development.

Table 2 - Faults associated with Humans and Examples of Fault Management Techniques

	Systematic	**Random**
Fault examples	Faults due to specification errors; misunderstandings or lack of ability	Random human mistakes.
Fault avoidance methods	Operator training: e.g. driving lessons. Ergonomic design techniques.	
Fault removal methods	Operator training and testing, e.g. use of flight simulators.	
Fault detection methods	Operator monitoring systems, e.g. dead-man's handle. Input or command screening techniques, e.g. syntax or spell checking.	
Fault tolerant techniques	Operator redundancy, e.g. use of two pilots in a civil aircraft. Limitations of operator's scope of action, e.g. aircraft envelope protection, automotive ABS and traction control.	

Table 3 - Faults associated with Mechanical Elements and Examples of Fault Management Techniques

	Systematic	**Random**
Fault examples	Faults due to specification errors; design errors, systematic manufacturing errors, etc.	Random component failure.
Fault avoidance methods	Conservative design methods, use of design rules, modelling techniques. Use of reliability engineering to predict susceptibility to random failures.	
Fault removal methods	Use of prototypes for destructive testing, fatigue testing, performance testing and estimation of failure rate. Environmental simulation testing.	
Fault detection methods	System monitoring systems, e.g. automotive temperature sensors, vibration sensors, oil pressure gauges.	
Fault tolerant techniques	Static or dynamic redundancy e.g. duplicated mechanical components, standby generators and automotive spare tyres.	

Table 4 - Faults associated with Electronic Hardware and Examples of Fault Management Techniques

	Systematic	**Random**
Fault examples	Faults due to specification errors; design errors, systematic manufacturing errors, etc.	Random component failure.
Fault avoidance methods	Formal design methods, use of design rules, modelling techniques. Use of reliability engineering to predict susceptibility to random failures.	
Fault removal methods	Dynamic testing using prototypes, such as destructive testing, fatigue testing, performance testing and estimation of failure rate. Static testing such as formal verification and design reviews. Design and environmental simulation.	
Fault detection methods	Hardware monitoring systems, functionality checking, consistency checking, signal comparison, checking pairs, instruction monitoring, loopback testing or watchdog timers.	
Fault tolerant techniques	Static, dynamic or hybrid redundancy, e.g. triple modular redundancy (TMR), *N*-modular redundancy (NMR) or standby-spares.	

Table 5 - Faults associated with Software and Examples of Fault Management Techniques

	Systematic	**Random**
Fault examples	Faults due to specification errors; software design or coding errors; compiler or other tool errors.	Unknown, random programming mistakes.
Fault avoidance methods	Animation of the specification. Rigorous or formal design methods. Use of appropriate software design languages such as Ada.	
Fault removal methods	Dynamic testing using prototypes, such as functional and performance testing. Static testing using static code analysis tools, formal verification or design reviews. Design or environmental simulation.	
Fault detection methods	Software monitoring systems, range checking and reasonability checking.	
Fault tolerant techniques	Diverse redundancy using as *N*-version programming or recovery blocks.	

Tackling Data Faults

Before progressing in our discussion of methods of tackling *data faults* we need to be clear about our meaning of this term.

The corruption of a byte within a ROM, or the corruption of information being sent between two computers, would not normally be considered as a software fault, but as a failure of the hardware concerned. Similarly, in this paper the term *data fault* is taken to mean a fault within the data developed for use by a system, and does not include any corruption of that data by the system itself.

In setting out to tackle the problems associated with data faults we get very little assistance from the literature, or from the various standards in this area. For example IEC 61508 says almost nothing of the problems associated with data, or how to address them. This omission would seem to reflect the general situation within industries working on safety-related systems, since a number of confidential interviews, performed as part of the work described in this paper, suggest that data faults are often largely ignored within the development of critical systems. The results of these interviews suggest that:

- Data is often not subjected to any systematic hazard or risk analysis.
- Data is often poorly structured, making errors more likely to be produced and harder to detect.
- Data is often not subjected to any form of verification.

Given the absence of any accepted practice in this area, it seems sensible to look at the techniques used to tackle faults of other types (as listed in Tables 2 - 5) to see if these suggest methods of tackling data faults. In fact, looking at the various tables it is clear that very similar methods are used to tackle a wide range of faults. For example, static or dynamic redundancy can be used to tolerate faults in many implementations. In computer hardware this might take the form of triplicated modules with some form of voting (static) or a standby-spare arrangement (dynamic). In software this might be implemented using *N*-version programming (static) or by recovery blocks (dynamic). In a mechanical system this could be represented by operating several motors together (static) or having a standby motor (dynamic). In a human-based arrangement this could take the form of having several operators working together and checking each other (static) or some form of fault detection and a standby operator (dynamic). Given these examples it seems appropriate to suggest that static and/or dynamic redundancy might also be an appropriate method of tackling data faults. Applying similar logic, it seems likely that appropriate methods of fault avoidance, fault removal and fault detection could also be used to increase the dependability of data. Table 6 suggests an approach to data fault management based on techniques used in other areas.

Note that when data is stored or transmitted it may be appropriate to make use of *information redundancy* such as checksums, parity bits or error correcting codes, to increase confidence in the correctness of the data. These measures are not included in Table 6 since they are effectively coping with hardware (or software) errors in the storage or transmission of the data, not with errors in the generation of the data itself.

Table 6 - Faults associated with Data and Examples of Fault Management Techniques

	Systematic	**Random**
Fault examples	Faults due to specification errors, data design errors or systematic collection/generation errors.	Unknown, random data collection/generation errors.
Fault avoidance methods	Animation of the specification. Rigorous or formal design methods. Use of automated and verifiable methods of data collection/generation.	
Fault removal methods	Appropriate static and dynamic testing techniques, modelling and simulation.	
Fault detection methods	Data monitoring systems, range checking and reasonability checking.	
Fault tolerant techniques	Diverse redundancy using static or dynamic techniques.	

Data Fault Management

It would not be appropriate in this paper to describe in detail the methods that should be used for each stage of the development of a safety-related system. This is partly because the appropriate methods depend on the nature of the system being developed, and partly because such a treatment would suggest a consensus on such matters that does not currently exist. However, it is possible to look at the methods of fault management used for other system elements and to make a few observations.

Perhaps the most important observation is that the *structuring* of the data is very important. When creating safety-related software we would normally avoid using thousands of lines of unstructured assembly code, because such coding is error prone and because such programmes are very difficult to verify. However, this approach is directly analogous to the completely unstructured techniques used to manage data in many safety-related systems. Data should be structured to facilitate verification, and to permit fault removal and fault detection techniques to be used.

In systems requiring a high safety integrity level, it might be appropriate to include fault tolerance in the form of data redundancy. As with software redundancy this would require diversity, since simply copying a data structure gives no protection against any form of systematic fault. Such diversity could take the form of *N-version data* (equivalent to *N*-version programming) using equivalent, but independently developed data structures. As with its software equivalent, the disadvantage of this approach is its very high development cost, and for this reason it is likely that such techniques would be reserved for only highly critical applications.

One of the problems facing the developers of safety-related systems relates to development tools. IEC 61508 gives a considerable amount of guidance on the selection of tools and techniques for the development of *software* for safety-related systems, but no similar guidance exists for *data* development tools. It is possible that this omission reflects an absence of dedicated tools in the area. Of great importance in the development of high integrity software is the availability of a range of *static code analysis* tools to help in the process of fault removal. Perhaps what is needed is a corresponding set of *static data analysis* tools to perform a similar function when developing data structures.

We noted earlier that data-driven systems often take the form of a 'standardised' system that is configured for a particular application by the use of configuration data. This can lead to a situation where the common elements of the system are developed and verified using systematic, well-structured techniques, while the configuration data is produced in an unstructured and uncontrolled manner at some later stage. Logic would suggest that any data that is relevant to the correct operation of a safety-critical system should be considered as part of the overall system and developed accordingly.

Recommendations

To tackle the problems outlined above, this paper makes a series of recommendations:

- The architectural design of a safety-related system should clearly identify the data elements within the system, distinct from the hardware and software components.
- The data within the system should have its own development lifecycle, with appropriate stages of hazard analysis, design and verification.
- The data should be described by an appropriate data model that is self-sufficient, clear, analysable and unambiguous.
- The data model, and its populated data set, should be developed, verified, documented and maintained, using techniques appropriate to the Safety Integrity Level (SIL) of the system concerned.
- The tools and techniques used in the creation of data for safety-related systems should be appropriate to the SIL of the system concerned. This might require tool manufacturers to development new tools, specifically for this purpose.
- Data that is derived from external sources, or that is developed separately from the safety-related system itself, should be subject to the same requirements of verification and documentation, as data produced as an integral part of the project.

Discussion

Standards such as IEC 61508 have been developed over many years with input from hundreds of engineers from around the world. It is therefore *unrealistic* to imagine that the suggestions given in this paper can in any way represent a definitive treatment of this matter. Rather, this paper sets out to publicise the need for further work in this area, and hopefully, to promote future revisions of standards like IEC 61508 to include this very important area.

While current standards say little specifically about the development of data, their coverage of design and development methodologies does provide useful guidance on techniques that might be appropriate to the production of data. However, effective use of this guidance relies on the identification of data as a distinct entity to which such methodologies should be applied.

In an industrial environment, commercial pressures often encourage engineers to perform the minimum amount of work necessary to demonstrate adherence to a given standard. In such situations, the identification of a major new system element, requiring its own development lifecycle, may not be attractive. It is therefore possible that the development of data will not be treated appropriately until this forms part of the various standards.

Conclusion

The widespread use of COTS is leading to an increasing number of systems where standard hardware and software are configured for a particular application (or instance of an application) by the use of custom data.

This paper makes the case for considering data as a separate and distinct entity within a computer-based, safety-related system. It also discusses the effects of faults within this data, and argues that these may be systematic or apparently random in nature.

Although the various standards give little guidance on the creation or management of data, the paper argues that many of the techniques used in the development of other system elements may be directly applicable. However, engineers working on the production of safety-related systems need clear and authoritative guidance on these issues.

References

1. IEC 61508 Functional Safety of electrical / electronic / programmable electronic safety-related systems Geneva: International Electrotechnical Commission, 1998.

2. RTCA DO 178B / EUROCAE ED-12B Software Considerations in Airborne Systems and Equipment Certification Washington: Radio Technical Commission for Aeronautics, Paris: European Organisation for Civil Aviation Electronics, 1992.

3. Interim Defence Standard 00-55 The Procurement of Safety Critical Software in Defence Equipment. Glasgow: Directorate of Standardisation, 1991.

4. International Standard 880 Software for Computers in the Safety Systems of Nuclear Power Stations. Geneva: International Electrotechnical Commission, 1986.

5. RIA Safety Related Software for Railway Signalling (Consultative Document) London: Railway Industry Association, 1991.

6. Littlewood B, Forecasting software reliability, in Software Reliability, Modelling and Identification, Lecture Notes in Computer Science 341 (Ed. Bittanti S), 141-209, London: Springer-Verlag, 1988.

7. Lyu M. R., (ed), The Handbook of Software Reliability Engineering New York: McGraw Hill, 1996.

Biography

N. Storey, B.Sc., Ph.D., FBCS, MIEE, C.Eng. School of Engineering, University of Warwick, Coventry, CV4 7AL, UK. Telephone - +44 24 7652 3247, facsimile - +44 24 7641 8922, e-mail - N.Storey@warwick.ac.uk.

Neil Storey is a Director within the School of Engineering of the University of Warwick. His primary research interests are in the area of safety-critical computer systems. He is a member of the BCS Taskforce on Safety-Critical Systems and has a large number of publications including both journal and conference papers. Neil is also the author of several textbooks on electronics and safety, including “Safety Critical Computer Systems” published by Addison-Wesley.

Alastair Faulkner, MSc., MBCS, C.Eng; CSE International Ltd., Glanford House, Bellwin Drive, Flixborough DN15 8SN, UK. Telephone +44 1724 862169, facsimile +44 1724 846256 email - agf@cse-euro.com

Alastair Faulkner holds an MSc degree in Computer Science from Salford University and is a Chartered Engineer. His background is in software development mainly concerned with computer based command and control systems. He now works on a large UK Rail infrastructure project. Alastair's research interests are in the data management of data-driven safety-related systems. He is also a Research Engineer with the University of Warwick and is studying for an Engineering Doctorate.

Paper Release Form
19th International System Safety Conference

Title of Paper: The Role of Data in Safety-Related Systems

I hereby authorize the System Safety Society to publish the paper listed above in the Proceedings of the 18th International System Safety Conference. Further, I agree to the following policy and notice regarding copyrights.

It is the policy of the System Safety Society, the sponsor of the International System Safety Conference, not to copyright the proceedings in order to provide the widest access for academic and educational use. Authors are free to copyright their papers as long as they agree with this policy. The policy to be contained in the proceedings is as follows:

Permission to print or copy: The copyright of all materials and commentaries published in these proceedings rests with the authors. Reprinting or copying for academic or educational use is encouraged and no fees are required; however, such permission is contingent upon giving full and appropriate credit to the author and the source of publication.

Author: Neil Storey

Address: School of Engineering
University of Warwick
Coventry
CV4 7AL
UK

Work Phone: +44 24 7652 3247
Home Phone: +44 24 7641 5517
FAX: +44 24 7641 8922
E-Mail: N.Storey@warwick.ac.uk

Signature Date

Author: Alastair Faulkner

Address: CSE International Ltd
Glandford House
Bellwin Drive
Flixborough
DN15 8SN
UK

Work Phone: +44 1724 862169
Home Phone: +44 161 338 2682
FAX: +44 1724 846256
E-Mail: agf@cse-euro.com

Signature Date

Mail to: John Livingston
Boeing Reusable Space Systems
555 Discovery Drive
Mail Code ZA-12
Huntsville, AL 35806-2809
(256) 971-3005, fax (256) 971-2699
john.m.livingston@boeing.com