



Engineering Safety Management

Yellow Book 3

Application Note 2

Software and EN 50128

Issue 1.0

Disclaimer

Railway Safety has taken trouble to make sure that this document is accurate and useful, but it is only a guide. The company does not give any form of guarantee that following the recommendations in this document will be enough to ensure safety. Railway Safety will not be liable to pay compensation to anyone who uses this guide.

Published by Railway Safety on behalf of the UK rail industry

Contents

1	Introduction	1
2	Planning a safety argument for software	2
2.1	Summary	2
2.2	What is and is not software?	3
2.3	Safety integrity	3
3	Specifying software	6
4	Delivering software	7
4.1	Software not developed to EN 50128	7
4.2	Other standards	7
4.3	Process assessment	7
4.4	Design analysis	8
4.5	Service record	8
4.6	EN 50128 and IEC 61508	9
5	Configuring software	10
5.1	Introduction	10
5.2	Guidance	10
6	Documents referenced	14
7	Acknowledgements	15
A	Appendix: Guidance on Serial Interlocking Specifications	16

I INTRODUCTION

The Yellow Book Steering Group has decided to supplement Yellow Book 3 with a series of application notes. Each note provides more detailed guidance on a particular aspect of the Yellow Book. This document is an application note on software and EN 50128. It provides guidance on some aspects of managing the development and use of software and avoiding some common problems.

The Yellow Book [1] provides little guidance on software. Section 9.8 of volume 2 refers the reader to EN 50128 [2], a standard for railway application software. It also follows EN50128 in recommending that you produce a software requirements specification and specify a safety integrity level for the software.

This note is consistent with that guidance but expands on it in four main areas:

- Planning a Safety Argument for Software
- Specifying Software
- Delivering Software
- Configuring Software

This note is written primarily for people working in these areas but may be useful to other people as well, such as independent safety assessors and safety authorities.

We have taken trouble to make this note accurate and useful, but it is only a guide. We do not give any form of guarantee that following the recommendations in this note will be enough to ensure safety.

This note is intended to be read with the Yellow Book. Although it provides guidance on good practice there may be other ways of tackling the problems described in this note which are not described here but are still good practice.

EN 50128 is part of a family of standards and should be read in conjunction with EN 50126 [3] and EN 50129 [4]. This family of standards is designed to be broadly consistent with IEC 61508 [5], a cross-industry, international standard for developing safety-related systems. As EN 50128 and the rest of its family are European standards which specialise cross-industry good practice to the rail industry, we regard them as authoritative sources of good practice in the rail industry.

At the time of writing, EN 50128 was being re-written and a new version had passed its final vote and was awaiting publication. If the new version is available then you should consult this.

We are continually working to improve the Yellow Book and we welcome comments. Please contact us at the address below, if you have a suggestion for improvement.

The ESM Administrator
Praxis Critical Systems,
20 Manvers Street,
Bath BA1 1PX
UK

Tel: +44(0)1225 466991
Fax: +44(0)1225 469006
Email: info@yellowbook-rail.org.uk

2 PLANNING A SAFETY ARGUMENT FOR SOFTWARE

2.1 Summary

If the system that you are building contains software then you will have to consider the software in the system safety case. You will have to include a software safety argument in the system safety case unless you can show that the system design is such that:

- the behaviour of the software cannot conceivably contribute to a hazard, and
- the system does not rely on the behaviour of the software to mitigate hazardous events.

You should consider the form of this argument from the outset when designing the system and the software. Section 3.2.6 of Yellow Book 3 Volume 1 advises that, among several other things, a safety case should show that “*you have set adequate requirements and met them*”. Generally, to support such a claim, you will need to show in the software safety argument that:

- a) the software safety requirements are sufficient;
- b) the software meets its software safety requirements; and
- c) if the software is configurable, that configuring it has not introduced risk (or, if it has, that this risk has been controlled).

The software safety requirements will specify the **behaviour** of the software and its **safety integrity**, which is a measure of the confidence that the software will behave safely.

We consider that EN 50128 represents good practice for development of railway software, including software outside the strict scope of the standard.

Following EN 50128 will deliver evidence that the software meets its safety requirements. However, following EN 50128 is not sufficient to demonstrate that these requirements are adequate and the safe behaviour of the software will depend upon hardware provisions. Therefore, software safety activities must be undertaken as part of a programme of activities to ensure the safety of the system as a whole as described in the Yellow Book and EN 50129.

Note also that EN 50128 does not provide guidance on all forms of software – it assumes a procedural programming language, for example – and that its treatment of software integrity is not underpinned by an agreed theoretical basis.

EN 50128 promotes a qualitative approach to software integrity but the issue of whether software failure can be modelled probabilistically is an issue of contention.

This application note has been written to supplement the standard by pointing out some problems that people building railway systems face and providing guidance on overcoming them.

The rest of this section looks at two particular issues that should be dealt with at the planning stage:

- what is and is not software; and
- the safety integrity of software.

2.2 What is and is not software?

Most of the time this is not a difficult question: it is quite clear what is software and what is not. In general software is a sequence of instructions that are carried out by some item of hardware, normally a general-purpose computer processor.

However, there are grey areas. For example, Field Programmable Gate Array (FPGA) devices are reconfigurable logic gate networks. They are programmable, may have internal states and have complex software-like functions, and are configured using something that looks very much like a programming language.

Other systems have behaviour that is defined by configuration data, which may have many of the features of software.

The question arises whether items such as the FPGA “programme” or the configuration data should be considered as software. To provide a useful answer to this question it is worth reviewing the differences between hardware and software.

Simple hardware systems have few internal states. It is generally possible to demonstrate that such systems perform as expected, through the use of logical analysis of the design, and exhaustive testing of the implementation.

Software, due to its sequential mode of operation, can change its behaviour radically based on input data. As the size of software grows, the number of states it can be in, and number of possible paths through it can grow at an exponential rate. Even for relatively simple software, the number of paths through will be very large. To fully test all the paths through the software, with all possible inputs and stored states becomes intractable for all but the smallest programs.

This is especially the case with real-time software that uses interrupts, in which the flow of control for the software is harder to model. Systems such as interlockings are even more complex to test because there may be several distinct ‘states’, for example where there are two independent trains in the interlocked area.

We recommend a practical approach to this question. It is the complexity of software that makes it necessary to use standards like EN 50128. Because it is generally impossible to test and or analyse every possible path through the software it is necessary to rely on the process used as well as the design itself to make a safety argument.

As a rule of thumb, we suggest that if a device has few enough internal stored states that it is practical to cover them all in testing, it may be better to regard it as hardware and to show that it meets its safety requirements by analysis of the design and testing of the completed device, including exhaustive testing of all input and state combinations.

If the programmable device has the complexity of software, then some at least of the guidance in EN 50128 is likely to be useful. However this guidance may not be applicable without modification. Several requirements (see for instance table entries A20.4, A19.3 and A12.3) assume a procedural language and so would not be directly applicable to programs written in other languages. In these cases, EN 50128 may be useful as a guide but you will have to replace inapplicable requirements with other tools, techniques and measures that meet the same underlying need.

2.3 Safety integrity

The Yellow Book glossary defines safety integrity to be:

“The likelihood of a system, product or other change satisfactorily performing the required safety functions under all the stated conditions within a stated period of time.”

Generally speaking, if a system includes software, then the safety integrity of the system will depend upon the safety integrity of the software.

Most hardware failures are random and their likelihoods can be estimated as probabilities using reliability-engineering techniques.

All software failures, however, are systematic. Software does not wear out or break. Most software failures are the result of errors in the software which themselves result from failures in the development process, such as incorrect specification (for instance specifying the wrong behaviour in the event of an error), or a mistake when implementing this specification.

The Yellow Book follows standards such as IEC 61508 [5], EN 50126 and EN 50128 in suggesting that you derive **safety integrity levels (SILs)** for system functions from target probabilities for their failure and then use these to define the rigour of the development process (see Volume 2, section 9.6); the SILs are derived from tables in EN 50129. It also follows another standard, DEF-STAN 00-56 [6] in offering guidance on apportioning SILs between functions where more than one function must fail for a hazard to occur (see volume 2, section 9.7). This guidance note reconfirms this advice. However, we are aware that the following issues arise in following it:

- **Choice of SIL Tables**

The tables used to define SILs are different in the Yellow Book, IEC 61508 and EN 50129. In fact the Yellow Book table was consistent with a late draft of IEC 61508 but a late change in the standard rendered them different. The differences between the failure probabilities given in IEC 61508 and the Yellow Book are less than 15%, which is not normally enough to give a different result. The Yellow Book and IEC 61508 are more conservative than EN 50129, that is with the same input they will give rise to the same SIL or a higher one.

We confirm our recommendation to use the tables in the Yellow Book (or IEC 61508), even when following the rest of EN 50129. Currently these are more conservative than EN 50128 so do not consider that this should undermine any claim to have followed EN 50129 and we expect the next issue of EN 50129 to make its tables more consistent with those in IEC 61508.

You should decide which table to use at the beginning of a project.

- **Probability of Software Failure**

Apart from the differences in SIL tables described above, IEC 61508, EN 50129 and the Yellow Book are in broad agreement on process for taking a *target* failure probability and deriving a SIL. However estimating *actual* failure probabilities is more controversial. There is no consensus within the software engineering community on methods of predicting the probability of software failures or even whether it is valid to assign a probability to these failures at all.

EN 50128 provides no method for estimating the probability of software failure. The practice of using the worst-case probability associated with the SIL of the software is not supported by the standard. We are aware that this practice has been followed on some railway systems, nonetheless. We do not endorse it although we do not consider it to be a completely unreasonable approach as the requirements of the standard would be open to challenge if they routinely resulted in software that failed more often than this limit.

Without estimating the probability of software failure it is not possible to estimate the probability of failure of a system containing software. It is possible, however, to estimate the probability of system failure from non-software causes and to present this figure, carefully explained, together with the SILs of the system function in the Safety Case. If you are using fault trees, the probability of system failure from non-software causes can be calculated by setting the probabilities of software failure to zero although it must be understood that this is a device for *excluding* software failure from the calculation, not an assumption that software does not fail.¹

Section 4.5 below discusses the use of service record to justify claims for safety integrity.

- **Functions with different SILs**

The process described in EN 50129 derives the SIL of a system from the SILs of its functions. We recommend retaining the SILs for the individual functions and referring to them when reviewing the design. If the SIL of a system or some software is dominated by one function it may be possible to design the system to reduce the overall SIL.

Practitioners have successfully justified designs with software functions of different SIL on the same processor, although EN 50128 does not provide any support for this practice. To be able to use software functions of varying SIL on the same processor, you must be able to produce a safety argument that demonstrates that the lower SIL functions cannot influence the behaviour of those with higher SILs. This may be through mechanisms that prevent interference such as memory protection or shown by analysis of the code, for example by demonstrating that no part of the code will write to memory outside of its designated area.

However, this can be difficult to do, and the effort required may be excessive compared with other solutions to the same problem.

In the text above we have noted some problems without providing complete solutions to them. We hope, over the coming years, to see developments in this area that will improve our understanding of software reliability and its prediction followed by improvements in standards such as EN 50128 as a result.

We have also noted some areas in which practice has deviated from EN 50128. We do not encourage gratuitous non-compliance but, while the area develops, we do think that it is reasonable for projects and safety authorities to consider alternative approaches to solve specific problems, provided that they can be justified.

¹ Be careful however if the software includes functions that protect against other hazard causes. Setting the probability of failure of such functions to zero can result in a zero estimate for the probability of the hazard. In these circumstances you may need to provide probabilities for nodes in the fault tree below the top event, if you are to provide the reader with useful information.

3 SPECIFYING SOFTWARE

EN 50128 requires a Software Safety Requirements Specification and a Software Requirements Specification for safety related software. It is possible to combine these into one document, but the safety-related requirements should be clearly identified.

The Software Safety Requirements Specification will play a pivotal role in the Safety Case for the system. As we noted in section 2.1 above, you will need to show that:

- the software safety requirements are sufficient; and
- the software meets its software safety requirements;

To support this, the Software Safety Requirements Specification must be complete, precise, and intelligible to both those developing the software and those applying it. Of course it is also desirable for the Software Requirements Specification to have all these attributes, or indeed any other Requirements Specification.

The following checklist is written for a system specification but may also help in writing and reviewing the Software Safety Requirements Specification:

- Every requirement should be unambiguous, that is admitting only one possible interpretation.
- The specification should be complete. It should include all the customer's and other stakeholders' requirements and those required by the context (standards, legislation and so on). Each requirement should be stated in full and any constraints or process requirements that affect the design should be completely specified. The specification should include both what the system must do, and what it must not do.
- The specification should be correct. As a minimum every requirement should have been verified by both the stakeholder it comes from and someone capable of judging that the system specified is safe.
- The specification should be consistent. There should be no conflict between any requirements in it, or between its requirements and those of applicable standards.
- Every requirement should be verifiable. There should be some process by which the developed software can be checked to ensure that the requirement has been met.
- The specification should be modifiable. Its structure and style should be such that any necessary changes to the requirements can be made easily, completely and consistently in a controlled and traceable manner.
- Every requirement should be traceable. Its origin should be clear and it should have a unique identifier so that it can be referred to.

Some specific issues arise when specifying serial interlockings and Appendix A provides guidance on these issues.

4 DELIVERING SOFTWARE

4.1 Software not developed to EN 50128

Following the process described in EN 50128, including the provisions for record keeping, will deliver evidence that a program meets its Software Safety Requirements Specification (including the specified SIL).

Sometimes it may not be practicable to follow this process. One reason may be that designer wishes to use software that has already been developed. COTS (Commercial Off The Shelf [7]) and SOUP (Software Of Unknown Pedigree [8], which may include software developed within your organisation for which there is no surviving design process documentation) are classes of such software but there are others. For brevity we will talk about COTS in the remainder of this section but the advice given is applicable to other classes of previously developed software.

Paragraph 9.4.5 of EN 50128 includes requirements relating to the use of COTS but following these may not be most practicable approach in every case.

You will need to show that the COTS meets its safety requirements, including its safety integrity requirements. It is possible to make a convincing argument for this in many cases. However, it may be difficult for higher SILs and it is not guaranteed to be possible in every case. You should work out, at least in outline, how you will make the argument before committing yourself to using COTS.

The safety argument for COTS may be complicated by the fact that COTS often includes functions that are not required and not used. You will need to show that the presence of these functions has no hazardous side-effects. It may also be impossible for the user to find out exactly how COTS software was developed.

Activities that may deliver evidence to support your argument, include reliance on other standards, process assessment, design analysis and analysis of service record. It is usual to use a mixture of several of these.

4.2 Other standards

If you have followed another well-recognised standard for safety-related software, then you may be able to base your argument for its safety integrity on that. Some possible standards include:

- Mü 8004 [9]
- Def Stan 00-55 [10]
- IEC 61508 [5]
- RIA 23 [11]

4.3 Process assessment

Where you have not done something that EN 50128 recommends, you may still be able to claim that you have achieved the desired safety integrity if you have used alternative measures or techniques and you can justify a claim that they are suitable and at least as effective. Alternatively, if the process used has largely followed EN 50128 but has fallen short of its requirements in isolated areas then it may be possible to carry out the omitted activities or generate the omitted outputs after the event. Carrying out these activities later than the standard prescribes may in some cases reduce the SIL that can be claimed, and may also lead to extra work, time and cost with little material benefit.

When developing software you should be aware that much of the data produced during software development lifecycle is easily lost but expensive to replace. Even if you have no specific plans to base a safety argument on the development process used, it may still be a good investment to keep records of the process in case you need them later.

4.4 Design analysis

Software is generally too complex, and has too many states, to prove by analysis that it behaves exactly as it should.

It may, however, be possible to show that some simple properties hold of the software and this may be enough to show that a software safety requirement is met or to form part of such a demonstration.

For example, it may be possible by careful analysis of the input/output statements in a programme and its control flow to show that two output instructions will always occur in a particular order.

It may also be possible, by careful inspection of the data path for an item of data, to show that it cannot be corrupted on the way. It is generally much harder to show that it will always be delivered.

Tools exist that allow you to perform static analysis of program code, in order to prove certain properties of a system, such as the absence of run-time exceptions, or the adherence to certain coding standards.

You should bear in mind the SIL you are trying to achieve when considering whether this approach is workable and if so what tools and techniques to use.

Conclusions from analysis typically depend upon assumptions such as “code cannot be overwritten” and “return addresses on the stack cannot be corrupted” which you should identify and confirm. If you analyse the source code rather than the object code there will always be an assumption about the integrity of the compiler which you will have to confirm (see the Yellow Book application note for Railway-Level Issues [12] for more information about managing assumptions).

If the possible safety arguments are considered during the architectural design of the system it may be possible to design the system to make the safety arguments easier.

4.5 Service record

If your software is already in service, you may be able to collect some evidence for its safety integrity from its service record.

It may be possible to make a direct claim for the frequency of hazardous software failures without recourse to SILs from records of its operation in service, provided that you can show all of the following:

- The records of failures are thorough and accurate.
- The software is under change control and the version for which the claim is made is substantially the same as the versions for which records were kept.
- The software will be subject to a similar pattern of use to that for which records were kept.
- The total time in operation of the software is known.

The data used needs to be either complete or a statistically valid subset. Any bias in the collection of data will invalidate conclusions drawn from it. The data needs to include information about the environment that the system was operating in, and the manner in which it was being used. If you are basing part of a safety argument upon such data, you should be able to demonstrate that the data used is of a high enough quality. This may require that the party providing the data also provides details of the collection method, sampling techniques and storage regime used.

See BS 5760 part 8 [13] and EN 50128 for further specific advice on the use of previous experience.

It may also be possible to make a direct claim for the frequency of hazardous software failures, without recourse to SILs, from records of testing, provided that:

- the test inputs were random and
- the software will be subject to a similar pattern of use to that for which it was tested.

However it is not generally statistically valid to claim that the mean time between a hazardous failure is more than one third of the total duration of use or testing for which records were kept, and then only if no hazardous failures occurred. In practice it is difficult to make claims for a safety integrity better than SIL 2 using a service record or testing data.

4.6 EN 50128 and IEC 61508

EN 50128 and IEC 61508 both place requirements on the production of safety-related software although the scope of IEC 61508 is wider than EN50128 and covers topics treated by EN50129 as well. EN 50128 is customised for railway applications and embodies good practice for both signalling and train-borne systems. In the context of the railways, EN 50128 is the primary standard that should be followed. On the whole, EN 50128 is derived from IEC 61508-3. However there are two significant differences:

- As described in section 2.3 above, EN 50129 and IEC 61508 associate SILs with different ranges of probability of failure. In this case the requirements of IEC 61508 are more onerous.
- IEC 61508 does not define the term “SIL 0”. If it is used to mean non-safety-related then designating software as SIL 0 excludes it from the scope of IEC 61508. EN 50128 does define SIL 0 and sets requirements for it, essentially setting requirements on all software in railway control and protection.

Yellow Book guidance follows IEC 61508 when associating SILs with different ranges of probability of failure and acknowledges the use of the term “SIL 0” to refer to functions which are not relied upon at all to control risk.

Some aspects of the guidance in IEC 61508 and EN 50128 are difficult to interpret for software where the instructions are not executed in sequence (see Appendix A).

5 CONFIGURING SOFTWARE

5.1 Introduction

Modern software is highly configurable. In our experience, a significant number of failures result from errors in the configuration of a particular installation of software rather than from the development of the software in the first place. Moreover, an error in configuration data may lead to complex and subtle hazards of the system that are hard to identify and correct.

Therefore it is important that as much attention is paid to the configuration of software as to its design and development.

There are two main classes of configuration data:

- that which describes how the software is to operate, the configuration of the actual software components, and
- that which describes the environment in which the software is to operate, for example the track layout, or the description of the timetable.

Configuration data may be largely static (for instance track layout), or it may be dynamic, entered by people during the operation of the system (for instance train delays).

This section provides guidance on configuring software.

5.2 Guidance

You should treat the integrity of configuration data, with the same degree of importance as you treat that of the software itself. The approach taken to creating the data should be as rigorous as that taken during software development.

You should analyse the software to establish, for each item of data, any hazards which incorrect values might cause.

When doing this you should consider at least the following ways in which data may be incorrect (this list may not be complete):

- Omission of data
- Corruption of data
 - Duplicate or spurious entries
 - Erroneous/corrupt data that is structurally correct
 - Structural faults
 - Type or range faults
 - Value errors where the value is plausible but wrong
 - Referential integrity failure between data
 - Volume, too much/little data
 - Incorrect ordering of data

Note: errors in some data items can cause unpredictable results. It may be simplest to regard these as potential causes of all hazards.

There is no precise agreement on how to treat data of different integrity but it may be useful to assign SILs to data items in order to focus attention on the most critical. This may be done by identifying the highest SIL of any function which might deliver a hazardous output as the result of an incorrect value of this data item.

5.2.1 Specifying configuration data

When developing software that uses configuration data, you should specify both the grammar (that is the structure) and the lexicon (the permitted values) of the data. This specification should be complete and consistent. The specification of the data should form part of the overall specification of the system, and should be produced with the same degree of rigour as the rest of the specification.

This specification should also include a description of the manner in which the data is to be stored, including the data formats to be used (e.g. the format for real numbers, the character set of text), and the manner in which the data are to be used (e.g. which values represent the end of a record).

You should describe, as accurately as possible, the meaning of the data and the manner in which it is to be used. There are likely to be connections between different data items. One data item may refer to another data item or there may be a relationship between the values of the two items. You should document these connections.

You should consider how to detect errors in the data. You should consider the use of error detecting codes, sanity checks, and consistency checks. Checks should be considered both during the preparation of the data, and when the system is being used. Be careful however with automatic error correction in case it should create incorrect data. Corruption in storage and transmission may be more safely handled by requesting that data be sent again.

Your specification should describe error detection mechanisms and define what the system should do if it detects an error. Where practicable, software that is presented with erroneous configuration data should fail in a manner such that it maximises safety, while indicating the failure and, when it is known, its cause. Failures should be recorded in order that the causes may be investigated. Changes in error rate may indicate a failure in a communication medium (e.g. a loose connection), or a change in the environment (e.g. increased interference from new equipment).

5.2.2 Managing and preparing configuration data

You should define and write down the process and tools to be used for preparing, checking and inputting the data. You should ensure that any tools used to prepare or test data have sufficient integrity that they will not compromise the integrity of the data.

You should take every practicable opportunity to introduce automated checks of data values or of relationships that should hold between data items.

You should ensure that anyone entering data at a screen is given feedback of the values that they have entered.

When issued, the Yellow Book application note for Human Error: Causes, Consequences and Mitigations [14] will provide guidance on controlling the risk arising from human error, which you may find helpful when considering possible errors made by the people entering the data.

You should maintain data under configuration management. You should use the same methods of configuration management as you would for software of the same safety integrity level.

5.2.3 Storing and transmitting configuration data

Data may be stored on magnetic (floppy/hard disk, magnetic tape), optical (CDs, DVDs), or solid-state (Flash RAM, Static or Dynamic RAM, [E]EPROMs) media. Data may be transmitted over wires (serial, Ethernet), optical fibre, optical wireless (infra red), and wireless radio.

Stored data may be susceptible to corruption from a range of environmental factors:

- Electric or magnetic fields
- Excessive heat or cold
- Chemical reactions
- Ionising radiation
- Unwanted modification (either human or automatic).

All storage media will deteriorate over time. You should assess the possible aspects of the environment that may affect the media on which you store configuration data. You should assess the time that data is to be stored and the possible factors that may influence the persistence of data on the media. Some media (especially magnetic and optical) will deteriorate from use, and will therefore have a lifespan determined in part by the frequency of use (both reading and writing). When selecting media you should take into account the likely frequency that data will be read and written, and choose the media appropriately. You should have procedures in place for the assessment of media being used in order to prevent the loss of data through media deterioration.

Corruption during the read or write process may occur due to electrical or mechanical failure. In order to minimise this possibility several strategies may be used:

- Read back data that is written. Be aware that many storage devices (especially hard-drives) use temporary storage to improve performance; ensure that the version stored is read back in order to ensure that it has been written properly.
- Write data in multiple locations on a single medium, or use redundant media. Read all copies of the data in order to discover individual recording errors.

Where data will not be changed often, you may wish to use some method to prevent it being accidentally overwritten. Such methods may include:

- Physically disabling the data writing component of the hardware, for example providing a switch to disable writes to memory after data is loaded.
- Using media that cannot be overwritten, such as CDs or PROMs.
- Using protection provided by operating systems.

Transmission of data is also subject to environmental influences and system failures. The environmental factors will depend on the medium:

- Both wired electrical, and wireless radio will be subject to electromagnetic interference.

- Wireless radio and optical will be susceptible to problems with propagation. Infrared and certain frequencies of radio will require line of sight, or will have a range that is affected by obstacles.

There are four main classes of failure for a transmission system:

- Loss of data.
- Corruption of data.
- Delay to data.
- Incorrect ordering of data.
- Insertion of spurious data.

You may also need to consider the possibility that someone may deliberately introduce correct-looking data into the transmission channel.

There are many well-understood protocols that can manage these failures. You should use one that is appropriate for the medium, and the information that you are sending. You may also wish to consider other techniques for improving the reliability both of the connection and the data sent across it:

- Using multiple wired connections that follow diverse paths to eliminate common causes.
- Using mechanisms to minimise interference such as balanced lines, or spread spectrum wireless transmission.

When sending or storing data, you should consider the use of error detecting codes.

EN 50159 [15, 16] provides further guidance in this area.

6 DOCUMENTS REFERENCED

1. Engineering Safety Management, issue 3, Yellow Book 3, Railtrack, 2000.
2. EN 50128:2001, Railway Applications – Communications, signalling and processing systems – software for railway control and protection systems.
3. EN 50126:1999, Railway applications. The specification and demonstration of reliability, availability, maintainability and safety (RAMS)
4. ENV 50129:1998, Safety related electronic systems for signalling
5. IEC 61508: 1998, Functional Safety of electrical/electronic/programmable electronic safety-related systems
6. Defence Standard 00-56 (Issue 2), Safety Management Requirements for Defence Systems, Ministry of Defence, 13th December 1999
7. The application of COTS technology in future modular avionic systems, G. Wilcock, T. Totten, A. Gleave and R. Wilson, Electronic & Communication Engineering Journal, August 2001, available through IEE Professional Network-Aerospace.
8. Methods for assessing the safety integrity of safety-related software of uncertain pedigree (SOUP), Health and Safety Executive, 2001
9. German Federal Railways Standard Mü 8004
10. Defence Standard 00-55 Requirements for Safety Related Software in Defence Systems, 1997
11. Safety related software for railway signalling (RIA 23), Railway Industry Association, 1991
12. Yellow Book Application Note 1: Railway-Level Issues, Issue 1.0, December 2002
13. BS 5760, Reliability of Systems, Equipment and Components, Part 8: Guide to assessment of reliability of systems containing software, 1998
14. Yellow Book Application Note 3: Human Error: Causes, Consequences and Mitigations, *in preparation at the time of issue*
15. BS EN 50159-1: 2001, Railway Applications – Communications, signalling and processing systems – safety related communication in closed transmission systems.
16. BS EN 50159-2: 2001, Railway Applications – Communications, signalling and processing systems – safety related communication in open transmission systems.

7 ACKNOWLEDGEMENTS

This guidance was prepared with the help of the following people who provided their time and expertise as professionals committed to improving railway safety. Their views do not necessarily reflect those of their employers. Their contribution is gratefully acknowledged.

Jeff Allan, Railway Safety
Paul Cheeseman, Lloyds Register-MHA
Trevor Cockram, Praxis Critical Systems
John Corrie, Mott MacDonald
Peter Duggan, Invensys
Bruce Elliott, Atkins Rail
Simon Errington, Lloyd's Register MHA
Richard Imhoff, Alstom
Iain Johnston, CSE International
Paul Leader, Praxis Critical Systems
Ian Shannon, Atkins Rail
Chris Shepperd, Siemens
Jeremy Whitley, Bombardier Transportation

A APPENDIX: GUIDANCE ON SERIAL INTERLOCKING SPECIFICATIONS

Signal interlockings used on railways have utilised three principal technologies: mechanical, electrical relay and solid-state processor-based systems. Mechanical and processor interlockings are both serial in operation, but much existing design is based on assuming relay logic, which is parallel. Thus signal engineers routinely use notations such as relay circuit diagrams to specify the logic they require. However, consistency with these diagrams is difficult to assure when processed serially because relay circuit diagrams describe parallel processing.

There is a tendency to build systems such as interlockings using serial computing devices. However signalling engineers, used to parallel logic, who specify interlockings in the traditional manner, need to understand the problems of producing an implementation with serial processors.

Relay-based devices operate in parallel in a way that is very hard to verifiably duplicate using serial processors. Devices, especially interlockings, enter intermediate states as the result of a sequence of events. States are stored explicitly within a serial computing device, but in relay-based systems, some storage is achieved through cross coupling. Relay-based devices do not explicitly separate logic and state.

Using a specification for a relay-based device to construct a serial computing device to perform the same function will be problematic because it is difficult to validate that a serial computing device will always behave in an identical manner to a parallel relay-based system. A small change in a relay diagram based specification could change not only the immediate behaviour of the interlocking, but also the manner in which information is stored for future use.

It is important that you are able to prove the serial specification fully reflects the intentions of the signalling engineer for all conditions. Producing a serial specification directly from typical relay circuits makes this difficult.

Relay diagrams and other fundamentally parallel specification notations are not well suited to the specification of serial processor based systems. It is better to base the specification on the desired function of the interlocking as follows:

- Identify all the states of the system and the transitions between those states.
- Isolate the elements that require storage, notably the different locking elements.
- Then specify the rest of the logic as operations upon these stores.

It is then possible to construct a specification that can be proven achieved by a serial processor (and, actually, this specification can be proven for either serial or parallel implementations). This approach is considered good practice in normal software engineering; understanding and identification of the information to be manipulated and the actions to be performed is a key stage in developing most software.

All notations and methods involve some compromise but, in general you should use notations and methods that are appropriate to the task, not simply those that people are accustomed to using. Interlockings are driven by data and the specification of this data has to be understood by both signalling engineers and software engineers. Be careful not to use symbols that can be interpreted differently by different people.

Published in February 2003 by:
Railway Safety
Evergreen House
160 Euston Road
London NW1 2DX
Phone: +44 (0)20 7904 7518
www.railwaysafety.org.uk

Distributed by:
Praxis Critical Systems Limited
20 Manvers Street
Bath BA1 1PX.
Phone: +44 (0)1225 466991
www.praxis-cs.co.uk

Copyright © Railway Safety 2003

You can download further copies from:
www.yellowbook-rail.org.uk