Rail Industry Guidance Note
GEGN8650
Issue: One
Date: March 2017

Guidance on High-Integrity Software-Based Systems for Railway Applications

Synopsis

This document provides guidance on the procurement and specification of high-integrity software.

Published by RSSB

Issue Record

Issue	Date	Comments
One	04/03/2017	Original document. To provide industry with the good practice guidance for procuring high-integrity software that resulted from T1047.

This document will be updated when necessary by distribution of a complete replacement.

Superseded Documents

This Rail Industry Guidance Note does not supersede any other Railway Group documents.

Supply

The authoritative version of this document is available at www.rssb.co.uk/railway-group-standards. Enquiries on this document can be forwarded to enquirydesk@rssb.co.uk.

Contents

List of Figures

Part 1 Introduction

1.1 Purpose

1.1.1 This document gives guidance on high-integrity software-based systems, as set out in *2.1* for railway applications. This document does not set out requirements.

1.1.2 This document has been developed to assist with:

a) The process for the procurement of high-integrity software and software-based systems.
b) The preparation of specifications for high-integrity software and software-based systems.
c) The contractual arrangements for software suppliers for software-based systems.

1.1.3 This document gives guidance to those in procurement and project teams who procure, receive and review high-integrity software or software-based systems, but do not have a detailed knowledge of software development processes.

1.1.4 Commercial Off-the-Shelf (COTS) software is not considered specifically in this document as it is not considered high-integrity. If the COTS software is customised to perform safety functions, then the guidance set out in this document may be followed.

1.1.5 This document does not constitute a recommended method to achieve fault-free high-integrity software. Software development processes are set out in European standards as set out in *1.3*.

1.2 Background

1.2.1 There are key safety benefits and technological drivers to introduce programmable technologies in the industry. High-integrity programmable controllers (hardware) rely on high-integrity software to perform safety functions.

1.2.2 High-integrity software and software-based systems are being used more frequently on the railways, so there is increased potential for safety-critical failures. The High Integrity Systems Group (HISG) was set up by RSSB to support the industry's understanding of these issues. The group commissioned the research report T1047 'Industry Guidance on High-Integrity Software', which has been used to inform this guidance note (GN).

1.2.3 In recent years there have been safety incidents involving software-based systems in operational use on the Great Britain (GB) railway where software faults have been a contributory factor. Two of these, Milton Keynes and Desborough, were investigated by the Rail Accident Investigation Branch (RAIB) and its reports for these incidents are in the references.

1.2.4 More details of principles concerning software engineering practices can be found in section G of research report T1047. These are not included in this GN as they are covered by European standards and are outside the scope of this document, as set out in *1.1*.

1.3 European standards relevant to this guidance note

1.3.1 BS EN 50128:2011 is currently the only European standard that includes detailed requirements for software for the rail industry and it addresses communication, signalling and control and protection systems. Software for use on rolling stock will be covered by prEN 50657, which is currently being developed.

1.3.2 BS EN 50155:2007 includes some generic requirements for software which are particularly applicable to COTS products.

1.3.3 BS EN ISO 9001:2015 provides requirements on quality management of software.

1.3.4 BS EN 50159:2010 includes some generic requirements for software relating to transmission systems.

1.3.5 BS EN 50126:1999 sets out the requirements for system Safety Integrity Levels (SILs) and safety management requirements.

1.3.6 BS EN ISO 13849-2:2012 includes some requirements for software for machinery. On Track Machines and On Track Plant have to comply with the Machinery Directive and may use BS EN ISO 13849-2:2012 as a presumption of conformance.

1.4 Approval and Authorisation

1.4.1 The content of this document was approved by the Control Command and Signalling Standards Committee on 27 October 2016.

1.4.2 This document was authorised by RSSB on 27 January 2017.

Part 2 What are High-Integrity Software and Software-Based Systems?

2.1 High-integrity software

2.1.1 High-integrity software is software that has gone through a rigorous development process with the aim of giving a high level of confidence that the software will perform as intended. The need to always perform as intended is paramount, as otherwise the result is likely to lead to large financial costs, disruption or physical harm. High-integrity software can therefore be considered to be the same as safety-critical software.

2.1.2 Rigour is defined in terms of how faithful the transformations from requirements to code are, and the assurance activities in the different phases of the development.

2.1.3 The following methods are used to support rigour:

- Validation and verification (V&V) process.
- Top down design methods.
- Modularity.
- Clear documentation and traceability.
- Robust configuration management and change control.
- Cross-discipline co-operation.
- Appropriate consideration of organisation and personnel competency issues.

2.1.4 The V&V process is applied during the development life cycle, according to good practices, to reduce the potential for functional failures as a result of errors or faults in the software. The V&V process is set out in BS EN 50126-1:1999.

2.1.5 Software verification provides objective evidence that the design outputs of a particular phase of the software development life cycle meet all of the specified requirements for that phase and maintain compliance of the product against overall product specification.

2.1.6 Software validation provides confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled.

2.1.7 Systematic verification and validation techniques need to be applied, ideally during the software development, or during integration of already existing pieces of software. For COTS products, validation techniques can be applied for assurance purposes.

2.1.8 The potential for software faults, which result in failures, increases with the complexity of software, and predicting all of the different ways of failure is generally not feasible. While it is not possible to identify all failures, the developer attempts to identify the failure modes as part of the development process.

2.1.9 It is not possible to test all behaviours generated by any complex software using dynamic testing alone. Exhaustively testing all binary states to add two integer inputs of 32-bits (yielding 2^{64} distinct test cases) would take hundreds of years, even if tests were performed at a rate of thousands per second. Static analysis techniques can analyse the code more quickly but also does not identify all faults. Software testing, in all its forms, is therefore only one part of the validation of the software.

2.1.10 In modern safety-critical systems, high-integrity software performs safety functions to mitigate risks that have been identified by safety analysis. Other functionality provided by high-integrity software could pose a risk if it fails.

2.1.11 High-integrity software or software-based systems can only be produced if the software is built correctly by the development team and the software is verifiable efficiently. For example, selecting an appropriate programming language and development tools that minimise the introduction of errors will have a much bigger impact on the defect density of software than adding more dynamic tests.

2.1.12 Software SILs are an indication of the rigour required of the activities of validation and verification of the software. The higher the software SIL, the higher the need for a reduction in the likelihood of software errors; hence, the more rigorous the software development.

2.1.13 In the context of the railway, BS EN 50128:2011 distinguishes five different software SILs:

- SIL 0 (lowest integrity).
- SIL 1.
- SIL 2.
- SIL 3.
- SIL 4 (highest integrity).

Note: SIL 3 and SIL 4 are generally regarded as high-integrity.

2.1.14 As the acronyms of software Safety Integrity Level and system Safety Integrity Level are the same, they are referred to as software SIL and system SIL in this GN.

2.1.15 High-integrity software can be procured separately or as an integrated part of a system.

2.2 High-integrity software-based systems

2.2.1 High-integrity software-based systems are those integrated systems of hardware and software where the software is high-integrity.

2.2.2 The procurement of these systems may include the processes set out in this GN for high-integrity software, in addition to the standard procurement processes for the system. It is expected that the same documentation will be delivered for the system as it would be if the software was being procured separately.

2.3 Measuring integrity

2.3.1 Random failures versus systematic failures

2.3.1.1 In the context of safety-critical systems, two categories of failures are identified:

a) 'Random failures' which are caused by random hardware faults and can be measured quantitatively.
b) 'Systematic failures' which are caused by systematic faults / design faults and can only be assessed qualitatively.

2.3.1.2 Systematic failures occur under deterministic conditions as a result of a fault or a sequence of faults that have been introduced during software development. Systematic faults can be created at any stage of the system's life cycle, including specification, design, development, operation and maintenance.

2.3.1.3 Unlike traditional engineering, not all behaviours of software-based systems can be predicted, and testing trials are not sufficient to predict all systematic failures. For this reason, random and systematic faults introduced during the development of the software, and software-based systems, need different kinds of mitigations, and controls, against safety risks.

2.3.1.4 The risk arising from systematic failures can be mitigated by following a rigorous verification and validation process that is associated with a design and production process with the appropriate methodology and techniques according to the software SIL. This aims to eliminate systematic or design faults. In this context, the software SIL has a qualitative meaning.

2.4 Software development life cycle

2.4.1 Figure *1* is a simplified version of the V cycle model for the software development life cycle set out in BS EN 50128:2011. This model illustrates the intermediate phases between the beginning and the end of the development life cycle. In each phase of the development, there is the possibility for a design error to be introduced.

Note: This is one of many software development life cycles that may be used. This guidance does not recommend any particular life cycle.

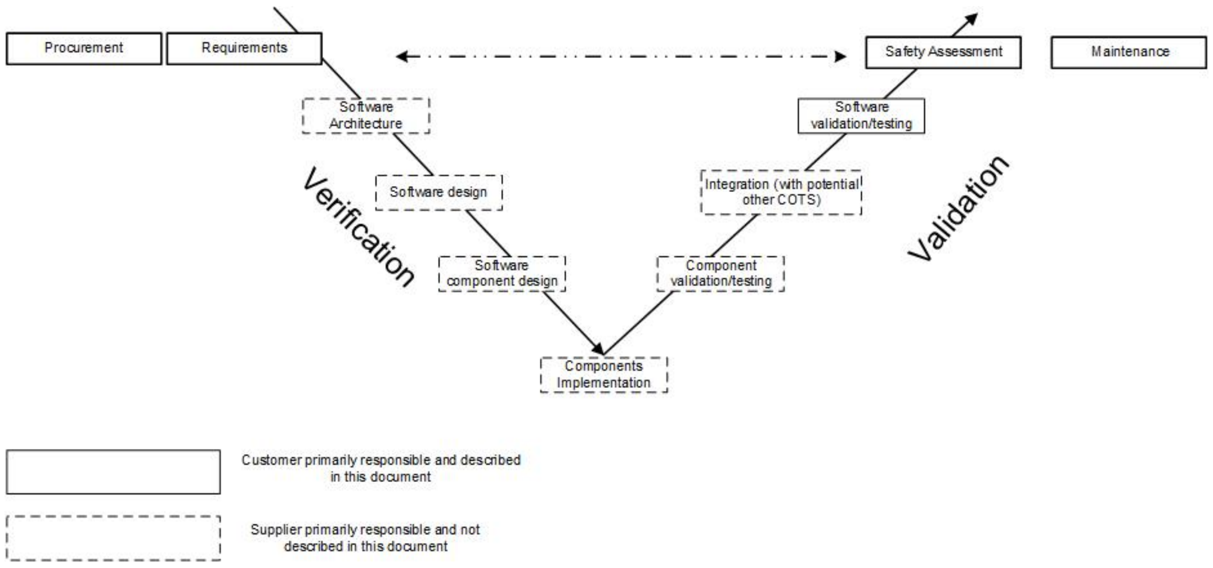
Procurement
Requirements
Safety Assessment
Maintenance
Software Architecture
Software validation/testing
Software design
Integration (with potential other COTS)
Software component design
Component validation/testing
Components Implementation
Verification
Validation
Customer primarily responsible and described in this document
Supplier primarily responsible and not described in this document

Figure 1: Simplified V model for software development life cycle

2.4.2 Some common design faults that can be introduced in each phase of the development are:

a) Omissions in the (safety) requirements.
b) Incorrect specification of the software architecture.
c) Incorrect design of the software or its components.
d) Undetected coding error during implementation.
e) Lack of design in the code to deal with erroneous or unexpected parameters.
f) Misinterpretation of the requirements by the software designer.

2.4.3 Unlike physical systems where the majority of faults are introduced during manufacture and maintenance, most of the faults in software are design errors introduced during the development phase. Software does not suffer from wear-and-tear – it will not change until it is upgraded or otherwise modified. Therefore, once the software is shipped, the design faults – or ‘bugs’ – will be hidden and remain latent until activation.

Note: Software may also change as the result of modifications by malicious software or changes in the software operating environment.

2.4.4 As with any complex system, discovering the design faults through testing is very difficult and may not be feasible, as set out in *2.1.9*.

2.4.5 To reduce the cost of the modifications it is always better to perform rigorous verification of the software design and the test specifications, but this is not always enough to avoid design changes. Regression testing is therefore required every time a safety function is affected. Modern simulation techniques allow validation of a model of the design quite early in the process, thus reducing the amount of rework and facilitating the impact analysis on the design of any required change.

2.4.6 Testing alone is not sufficient to eliminate software faults, for two main reasons:

a) Faults can be introduced in any phase of the software life cycle, yet only code can be extensively tested. Identifying faults introduced from previous phases of the development life cycle from testing code alone can be challenging or nearly impossible in any complex software.
b) The number of all possible software faults is too big to be tested.

2.4.7 Rigorous development techniques assist in reducing the number of faults introduced during the design phase and provide information to support V&V activities. Annex A in BS EN 50128:2011 sets out development techniques for different software SIL levels.

2.4.8 Employing appropriate V&V techniques provides assurance about the level of integrity of the software. Table A5 of BS EN 50128:2011 sets out guidance on V&V techniques for each software SIL level.

2.4.9 Validation techniques can be applied post-development, for example for COTS software, and could provide assurance about the integrity of the software. Obtaining sufficient evidence for full validation can be problematic, so this is often only suitable for lower integrity COTS software.

2.4.10 The verification process and associated verification activities applied to each phase of the life cycle of the software substantially help in eliminating design errors and faults as the development progresses.

2.4.11 BS EN 50128:2011 sets out a number of ways to perform the V&V activities. V&V activities are not easily interchangeable and some methods / activities are more formal than others and are appropriate in specific development circumstances. Verification methods to reduce faults in the code and the data vary, from code inspections by independent developers to automatic formal proofs using methods set out in BS EN 50128:2011 Table A 11.

2.4.12 The verification methods applied to the software process reduce the number of faults in the final software product.

2.4.13 Formal methods are the most rigorous ways for proving the conformance of software (in the final phase of the development) with the specification / requirements (initial phase).

2.4.14 The V&V process does not help in assessing if the intended requirements, which address the original problem, have been adequately captured. Methods for assessing this are set out in Part *4*.

2.4.15 Formal methods, as set out in D.28 of BS EN 50128:2011, use mathematically-based notations and tools to:

a) Formally verify the functional requirements of a system (or of the individual components of a system).
b) Refine (via a sequence of refinement phases, each of which is formally proven correct) the functional requirements into a 'low-level' implementation of those requirements.
c) Validation of the requirements and the fulfilment of the safety requirements if simulation techniques are available.

2.4.16 The application of formal methods requires a well-understood and defined set of requirements. Historically, the application of formal methods required a highly skilled workforce trained in these methods. There are now tools that do not require these skills as they hide all the mathematical analysis from the user. These tools can also be used to entirely remove low-level programming errors and security holes.

2.4.17 Mathematically-based notations are best used in small, well-understood and specific projects such as interlocking design, due to the skills and resources required to apply them, unless the tools described in *2.4.16* are used.

2.4.18 Formal methods for modelling requirements can be used to help uncover hidden and unintended assumptions. Once the requirements are fixed, the V&V process can only assess whether the software conforms to the specifications.

2.4.19 Formal methods and modelling techniques are proven tools to deliver a better understanding of the system boundaries, functional states and behaviours. The use of these techniques will enable a more accurate and clear requirements definition.

Note: Formal methods are only highly recommended in BS EN 50128:2011 for high-integrity software, as other methods are available.

2.4.20 As software development is an iterative process, a good change management process is used to ensure traceability of the requirements throughout the life cycle of the software.

Part 3 Guidance on the Procurement of High-Integrity Software and Software-Based Systems

3.1 Introduction

3.1.1 Procuring software against incomplete or incorrect requirements significantly increases the costs and time for software development. It is therefore important that requirements are adequately captured, researched and documented before placing a contract.

3.1.2 Changes in requirements at late stages of the software development life cycle will lead to additional costs, due to the effort involved in revising the work and V&V activities from earlier phases.

3.1.3 Since change is often inevitable, it is recommended that the change management process is included in the contract.

3.1.4 The following sections provide guidance on the procurement of high-integrity software.

3.2 Requirement development

3.2.1 Requirements are the starting point for the software development life cycle, as shown in Figure *1*. Even if the software is fault-free, the code will behave exactly as specified in the requirements with inadequate requirements delivering inadequate software.

3.2.2 As with hardware systems, the technical specification of high-integrity software and software-based systems relies on technical experts who have a deep understanding of the safety and integrity principles, and of how the system needs to behave in the domain of application, including the overall business needs (service performance, reliability and operability / maintenance).

3.3 Documenting the design choices

3.3.1 At each phase of the development life cycle shown in Figure *1*, design choices are made to ensure the description of the original requirements is gradually transformed into code. It is important that the original requirements can be easily traced into the code, to facilitate V&V activities. Documenting the traceability of the requirements at each phase during the development of the software facilitates the tracing of mistakes / errors / ‘bugs’ and changes made in each phase of the development life cycle. Tracing requirements retrospectively from the code is both very time consuming and costly. Annex D.58 of BS EN 50128:2011 provides guidance on traceability of requirements.

3.3.2 Safety analysis determines the safety requirements for the software. As part of any change to the requirements the impact of the change is considered on the software SIL level and therefore if the V&V techniques need to be changed or the process repeated.

3.4 Documenting the selection of V&V activities

3.4.1 Documenting the V&V activities at each phase of the software development is an integral part of being able to independently assess if the process to achieve the required software SIL has been followed.

3.4.2 BS EN 50128:2011 sets out a choice of V&V activities to meet the appropriate software SIL.

3.4.3 Good practice is for the supplier to justify the choice of V&V activities. Typically, the justification takes into account:

a) Complexity of the software.
b) Software development life cycle model chosen.
c) Coding language chosen for the software (section 6.7 in BS EN 50128:2011).
d) Availability of the tools needed to perform the V&V activities.

3.4.4 Methods of adequately documenting the verification activities are set out in section 6.2.4 of BS EN 50128:2011.

3.4.5 Methods of adequately documenting the validation activities are set out in section 6.3.4 of BS EN 50128:2011.

3.4.6 It is important when procuring software, to obtain possession of the V&V documentation to facilitate future development of the software and, if required, independent assessment of the software SIL level.

3.4.7 The higher the software SIL that is required, the more rigorous the V&V process is and the more stringent the maintenance plan, as set out in section 9.3 of BS EN 50128:2011.

3.5 Maintenance plan

3.5.1 To preserve the software SIL throughout the lifetime of the software used in a safety-critical system, maintenance and obsolescence plans are used to record amendments and changes, providing traceability to be assessed. The maintenance plan also shows the current status of the software.

3.5.2 The higher the SIL that is required, the more rigorous the V&V process is and the more stringent the maintenance plan, as set out in section 9.3 of BS EN 50128:2011.

3.5.3 The maintenance plan may also include information about:

- Configuration management.
- Disposal of the software.
- Retirement of the software.
- Bug fixes.

3.5.4 Obsolescence plans and strategies need to take place at development stages (hardware and software). COTS hardware obsolescence will potentially drive software retest and / or re-design with significant cost implications.

3.6 Procurement of high-integrity software or software-based systems

3.6.1 Before beginning the procuring of high-integrity software, the specification is prepared in draft. This starts with a hazard analysis of the system, by the system and / or software designer and ideally with user input, to determine the tolerable or acceptable failures for the safety function belonging to the system. In turn, failure criteria are assigned to each component or sub-system, including software-based components.

3.6.2 The principles for writing the software requirements and specification are set out in Part *4*.

3.6.3 The principles for supplier management, including software maintenance and V&V activities, are set out in Part *5*.

3.6.4 Details of suggested requirements in the contract are set out in *5.2.1*.

3.6.5 This process facilitates the correct software specification and therefore supports the contract management.

3.6.6 Where COTS software is being considered, the participation of all stakeholders in the decision provides confidence that it is sufficient to meet the system requirements; that is, meet the minimum requirements of clause 7.3.4.7 of BS EN 50128:2011. Stakeholders could include:

- System supplier.
- System operator.
- Safety department.
- COTS supplier.
- Others, as appropriate.

3.6.7 Where software-based systems are being procured, the same process is followed if the software is being specified as part of the system. In the case where the software is already written and being integrated into a system, it is expected that the documentation described in this GN is provided with the system.

3.6.8 Where software or software-based systems are being customised for a particular use, the same process as set out in this GN is required, but some of the tasks assigned to the supplier in this GN may actually transfer to the customer if they are the party responsible for the customisation.

Part 4 Guidance on the Preparation of High-Integrity Software Specifications

4.1 Determining the software SIL

4.1.1 The hazard analysis, undertaken prior to the start of the procurement process (see *3.6.1*) combined with the information in clause 4 of BS EN 50128:2011, helps with the allocation of the software SIL, which in turn determines the selection of V&V activities, as set out in *3.4*.

4.2 Adequate requirement specification

4.2.1 Adequate requirements are complete, correct, and consistent and include the identification of safety functions. Any requirement that is incorrect, unclear, ambiguous or omitted is a systematic fault.

4.2.2 Complete requirements are typically supported by documentation that describes the source and reason for the requirement and its development history.

4.3 Principles for complete requirements

4.3.1 Safety analysis during requirement development

4.3.1.1 High-integrity software is often developed for safety-critical systems; therefore, safety requirements need to be adequately elicited, documented and validated. Undertaking this analysis at the very early stage in the development of software gives confidence that the safety function will be achieved.

4.3.1.2 Safety analysis can be undertaken according to the process defined in the Common Safety Method for Risk Evaluation and Assessment (CSM RA). Guidance on CSM RA is set out in GEGN8640, GEGN8641, GEGN8642, GEGN8643, GEGN8644 and GEGN8645.

Note: Alternative safety management systems may be used where CSM RA is not applicable.

4.3.2 Requirement categorisation

4.3.2.1 It is unlikely that all requirements can be derived without an iterative process. The criteria for requirements and the related documentation are set out in clause 7.2 of BS EN 50128:2011.

4.3.2.2 In order to check the completeness of the requirements, it is useful to consider different categories of requirements. Examples include:

a) Functional.
b) Safety.
c) Security, for example protection from cyber attacks and the access rights.
d) Operational, for example interaction with other systems and user interface.
e) Software performance, for example speed of processing and compatibility with other system processes.
f) Input and output data.
g) Reliability.
h) Maintainability.
i) Compatibility with other software applications and communication protocols.
j) Data access connectivity (database connection compatibility).

4.3.2.3 Identifying all stakeholders in a project and understanding how they are related to different categories of requirements helps to prevent important details being omitted.

4.3.2.4 In the case of a new software development, or in the case of integration of different pieces of software, the involvement of software engineers in the process of specifying the requirements can assist with the supplier's understanding of the requirements.

4.4 Principles for correct requirements

4.4.1 Documenting the system's operating environment

4.4.1.1 It is the customer's responsibility to collect and appropriately document information (requirements and assumptions) about interfaces and interactions with the operating environment. This is because it is the customer who has the knowledge about the operating environment of the system that the software is designed to work in.

4.4.1.2 Typically, the operating environment of a software-based system contains:

a) Operators / users.
b) External interfaces (interfaces with equipment or interfaces with people).
c) Externally connected equipment or systems.

4.4.1.3 Inconsistent or incomplete specifications of any of the interfaces could lead to failures.

4.4.1.4 A 'data dictionary' may be created and maintained as part of the system specification to facilitate understanding of the data items used in the software. The data dictionary describes the meanings and the formats (dimensions, units and representations) of static and dynamic data items that the software-based system either receives from, or transmits to, its intended operating environment.

4.4.1.5 A data dictionary reduces the potential for different parties making different assumptions about the meanings and formats of data items, and provides an objective and agreed point of reference for checking the dimensional correctness of the calculations specified to be performed within the software-based system.

4.4.1.6 The data dictionary supports the implementation of defensive programming, as set out in Annex D. 14 of BS EN 50128:2011, which is a useful technique to prevent failure due to unexpected inputs.

4.4.2 Documenting what the software must and must not do

4.4.2.1 It is possible for software to create hazards in the system's operating environment if it does not perform as intended. It is therefore necessary for the customer to identify the limits of operation of the software and specify this to the supplier.

4.4.2.2 Identifying the limits of the inputs, for example, minimum and maximum values, allows invalid inputs to be identified and fault messages created.

4.4.2.3 A systems engineering approach can be used to aid this process, particularly for safety-functions. Example steps are:

a) Define what is the objective or safety-target for the function.
b) Identify and list input variables and expected outputs.
c) Define and document test-cases or scenarios that will demonstrate expected results:
   i) Expected inputs result in expected outputs.
   ii) Unexpected inputs do not result in undesired outputs.

4.4.2.4 An example of a good requirement that defines the limits of operation is 'When button x is pressed, the speed of the train is displayed in miles per hour'.

4.5 Principles for consistent requirements

4.5.1 Managing changes in software requirements

4.5.1.1 Keeping track of changes to software requirements during the specification development can be challenging.

4.5.1.2 Principles, to keep requirements consistent include:

a) Documenting the requirements and the rationale for choosing each specific requirement.
b) Keeping requirements under strict version control.
c) Using an automatic tool to document changes in requirements.
d) Modelling the requirements at the appropriate level of abstraction, if possible.

4.5.2 Verification of requirements

4.5.2.1 Verifying software requirements provides confidence that the correct requirements have been elicited from the software requirements specification. Useful methods of verification include:

a) Prototyping (useful for systems with user interfaces).
b) Simulating the requirements and assessing them for consistency against the core requirements.
c) Scenario based walk-throughs.
d) State machines.
e) Animation and model simulation.
f) Model-based design tools.

Part 5 Management of Software Suppliers for Software-Based Systems

5.1 Introduction

5.1.1 When issuing a contract for the development, or integration, of a software-based system, it is the customer's responsibility to ensure that the contract is written so that the customer's intentions and requirements are made explicit. This includes the specification of activities set out in this GN.

5.1.2 The customer may consider performing a project risk analysis of what might go wrong during the software development, and monitoring the true state of the project so that early corrective action can be undertaken.

5.1.3 For high-integrity software contracts, the creation and delivery of adequate documentation and plans during the development phase is critical. These plans may then be updated during the integration phases, as appropriate. Clause 5.3.2.4 of BS EN 50128:2011 identifies the following documentation:

a) Software Assurance Quality Plan, which documents the choices for the software development model and for the rationale of the decisions in each phase of the software development.
b) V&V Plans, which document the rationale for the choices made for the V&V tasks in the V&V process.
c) Configuration and Management Plan, which documents the data needed to correctly run the software and any activities during the operational use of the software, including how to deal with error reporting, error patching and the supplier's support to ensure the adequate running of the software.
d) Other documents appropriate to the software SIL level, as set out in Table A1 of BS EN 50128:2011.
e) Software architecture.

5.1.4 Documentation of the development of software is useful in reducing costs in the event of a change of supplier. It is very costly to reverse engineer undocumented complex code. Documentation that provides traceability of activities, requirements, and architectures in each phase of the software development, facilitates both changes in the project and in the supplier.

5.1.5 Formal methods, as set out in *2.4*, for the V&V process may require staff with specialised skills in software engineering. To assess the validity of the choices of the V&V process, consideration may be given to hiring an independent specialist to provide these skills.

5.1.6 Selecting the right supplier is also an important part of ensuring that high-integrity software is delivered to specification. The most important aspect of selecting the right supplier is ensuring that they have experience of delivering software to the required, or a higher, SIL level and preferably in the rail domain. It is often difficult to make the change from one SIL level to a higher one. Supplier selection for software follows the same structured process as for hardware, with the details above being one of the criteria.

5.2 Principles for managing suppliers

5.2.1 Technical requirements for contracts

5.2.1.1 It is useful to include contractual clauses that provide:

a) Support during the identification of the different categories of requirements.
b) Support during the requirements definition.
c) Support during the testing and / or simulation phase.
d) Adequate management of the agreed changes in the requirements.

e) Documentation that defines all the requirements in a structured manner.
f) Documentation that lists error codes (and solutions).
g) Documentation that lists the log messages to be generated, displayed and logged by the system.
h) Training during introduction of the software.
i) Continued technical support for the software (helpdesk).
j) Provision of updates to the software.
k) Definition of the delivery method for upgrades to the software.
l) Definition of the Intellectual Property Rights (IPR) or ownership of the software.
m) Lifetime support for the software (obsolescence management).
n) The software functional safety report or the evidence for the software independent safety assessment.
o) Management of cyber security.

Note: IPR may be managed through the use of an escrow agreement. Guidance on this is given in GEGN8607.

Note: Guidance on cyber security has been written by the Department for Transport (DfT) in 'Rail Cyber Security - Guidance to Industry'.

5.2.2 Need for independent certification of software

5.2.2.1 The requirements for assessment of software are set out in BS EN 50128:2011. Higher-integrity software always requires an assessment by an assessor who has the qualifications set out in Table B.8 of BS EN 50128:2011 and is independent, as set out in clause 5.1.2 of the same standard.

5.2.2.2 If the technical changes related to introducing the software or system are significant, as defined in article 4 of the CSM RA, then the risk assessment for CSM RA requires an independent assessment. The independent assessment applies to the whole change, not a specific certification for the software. A duty holder could ask an independent assessor to evaluate the documentation produced by the software suppliers as part of the compliance with CSM RA.

5.2.2.3 Independent assessment can also be requested by the customer; therefore, good practice is to consider whether independent assessment is required and provision for it included in the contract.

5.2.2.4 Where software already has an independent certification, the customer may review the independent certification to determine the applicability of the certification to their use of the software, and may request additional independent certification.

5.2.3 Generating evidence of V&V activities

5.2.3.1 There are two necessary separate aspects to software correctness:

a) The correctness of the transformation of the requirements into source code (verification).
b) The correctness of the software source code meets the needs of the user, as set out in the software requirements (validation).

5.2.3.2 Both processes are documented to provide evidence that the software meets its requirements.

5.2.3.3 The production of a Factory Acceptance Test (FAT) report as part of the evidence enables the customer to progress with User Acceptance Testing (UAT) with confidence.

5.2.4 Specifying notations to be used

5.2.4.1 Notations are the way in which software is written, for example, the language used. More rigorous notations cost more as they require more time and specialised skills. Therefore the notations to be used should be specified to match the requirements of the software SIL.

5.2.4.2 BS EN 50128:2011 sets out the different notations that could be used for each software SIL.

5.2.4.3 There is benefit in the supplier specifying the notations that are to be used so that they use notations that they are familiar with. The selected notation type is recorded in the contractual documents.

5.2.4.4 Rigorous notation facilitates the fault detection process.

5.2.5 Role of the customer

5.2.5.1 The customer should expect to be involved throughout the software development process to respond to technical queries. The involvement will be through mutual agreement.

5.3 Principle for managing suppliers for the maintenance of the software

5.3.1 It is useful to include contractual obligations that enable the future maintenance of the software. The maintenance could either be done collaboratively with the developer or the contract could enable a third party to do the maintenance. The items to consider in the contract include:

a) IPR for the source code, final software applications, any associated tools to run or test the code (compilers, simulators etc) and its documentation (including training modules). Where this is not possible, a software escrow agreement can be used instead.
b) Documentation for the software development, as set out in Part 4.
c) Documentation for the V&V activities, including the activity selection justification.
d) Documentation of all tests performed with the expected outcome and their results.
e) Arrangements for the deployment and maintenance of versions of the software-based system in its operating environment, including, for example, arrangements to allow the customer to prepare and modify application data for instances of the software system without the involvement of the supplier.
f) A maintenance plan to support the future development of the software.
g) Training in the use of the software.

5.3.2 Requirements for a maintenance plan are set out in section 9.3 of BS EN 50128:2011.

Definitions

Cant Deficiency	The difference between actual cant and the theoretical cant that would have to be applied to maintain the resultant of the weight of the vehicle and the effect of centrifugal force, at a nominated speed, such that it is perpendicular to the plane of the rails.
CSM RA	Common Safety Method for Risk Evaluation and Assessment. COMMISSION REGULATION (EU) No 2015/1136 of 13 July 2015 amending Implementing Regulation (EU) No 402/2013 on the common safety method for risk evaluation and assessment.
Error	Discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition. *IEV192-03-02* **Note:** Note to entry 1. An error within a system may be caused by failure of one or more of its components, or by the activation of a systematic fault. *IEV192-03-02*
Failure	Loss of ability to perform as required. *IEV192-03-01* **Note:** Note to entry 1. A failure of an item is an event that results in a fault of that item. **Note:** Note to entry 2. Qualifiers, such as catastrophic, critical, major, minor, marginal and insignificant, may be used to categorise failures according to the severity of consequences, the choice and definitions of severity criteria depending upon the field of application. **Note:** Note to entry 3. Qualifiers, such as misuse, mishandling and weakness, may be used to categorise failures according to the cause of failure.
Fault [Software]	A fault is an incorrect software system state that prevents it from performing as required. It may result from failures in system components, design errors, environmental interference, or operator errors.
Integrity [Software]	Software functional integrity refers to the fidelity of the code to the intended specifications, that is, functions implemented in the software would behave as intended in the requirements.

Procuring	Procuring is a process of identifying, planning, and benchmarking goods and services that are purchased. The procurement process includes defining the outcomes that are being sought, and determining whether there is agreement between buyers and suppliers on an optimum service design.
Reliability	The ability of an item to perform a required function under given environmental conditions for a given period of time. *BS EN 50128:2011*
Risk analysis	The systematic use of all available information to identify hazards and to estimate the risk. *CSM RA*
Risk assessment	The overall process comprising a risk analysis and a risk evaluation. *CSM RA*
Safety	The freedom from unacceptable risk to the outside from the functional and physical units considered. *IEV351-57-05*
Safety critical	Directly influencing safety (when applied to equipment or systems). *GKGN0802*
Safety integrity	The ability of a system to achieve its required safety function under all the stated conditions within a stated operational environment and within a stated period of time. *BS EN 50129:2003*
Safety Integrity Level (SIL)	A number which indicates the required degree of confidence that a system will meet its specified safety function. *BS EN 50129:2003*
Software safety integrity level (Software SIL)	A classification number which determines the techniques and measures that have to be applied to software. *BS EN 50128:2011*
Sub-System	A portion of a system that fulfils a specialised function. *BS EN 50129:2003*
System	A set of sub-systems that interact according to a plan. *BS EN 50129:2003*
System safety integrity level (System SIL)	A classification number which indicates the required degree of confidence that an integrated system comprising hardware and software will meet its specified safety requirements. *BS EN 50128:2011*
Traceability	Degree to which a relationship can be established between two or more products of a development process, especially those having a predecessor/ successor or master/subordinate relationship to one another. *BS EN 50128:2011*
Validation [Software]	A process of analysis followed by a judgment based on evidence to determine whether an item (for example, process, documentation, software or application) fits the user needs, in particular with respect to safety and quality and with emphasis on

	the suitability of its operation in accordance with its purpose in its intended environment. *BS EN 50128:2011*
Verification [Software]	A process of examination followed by a judgment based on evidence that output items (process, documentation, software or application) of a specific development phase fulfil the requirements of that phase with respect to completeness, correctness and consistency. *BS EN 50128:2011*

Abbreviations

CCS	Control, Command and Signalling.
COTS	Commercial-Off-The-Shelf.
HISG	High Integrity Systems Group.
IEC	International Electrotechnical Commission.
IEV	International Electrotechnical Vocabulary.
RSSB	Rail Safety and Standards Board.
SC	Standards Committee.
SIL	Safety Integrity Level.
Software SIL	Software Safety Integrity Level.
SRS	System Requirements Specification.
System SIL	System Safety Integrity Level.

References

The Catalogue of Railway Group Standards gives the current issue number and status of documents published by RSSB. This information is also available from http://www.rssb.co.uk/railway-group-standards.co.uk.

RGSC 01	Railway Group Standards Code
RGSC 02	Standards Manual

Documents referenced in the text

Railway Group Standards

GEGN8607	Guidance on the Use of Escrow Agreements for Rail Applications
GEGN8640	Guidance on Planning an Application of the Common Safety Method on Risk Evaluation and Assessment
GEGN8641	Guidance on System Definition
GEGN8642	Guidance on Hazard Identification and Classification
GEGN8643	Guidance on Risk Evaluation and Risk Acceptance
GEGN8644	Guidance on Safety Requirements and Hazard Management
GEGN8645	Guidance on Independent Assessment
GKGN0802	Glossary of Signalling Terms

RSSB Documents

Rail Cyber Security - Guidance to Industry	http://www.rssb.co.uk/Library/improving-industry-performance/2016-02-cyber-security-rail-cyber-security-guidance-to-industry.pdf
T1047	Industry Guidance on High Integrity Software http://www.sparkrail.org/Lists/Records/DispForm.aspx?ID=11362

Other References

BS EN 50126:1999	Railway applications. The specification and demonstration of reliability, availability, maintainability and safety (RAMS). Basic requirements and generic process
BS EN 50128:2011	Railway applications – Communication, signalling and processing systems - Software for railway control and protection systems

BS EN 50129:2003	Railway applications – Communication, signalling and processing systems - Safety-related electronic systems for signalling
BS EN 50155:2007	Railway applications. Electronic equipment used on rolling stock
BS EN 50159:2010	Safety-related communication in transmission systems
BS EN ISO 13849-2:2012	Safety of machinery — Safety-related parts of control systems. Validation
BS EN ISO 9001:2015	Quality Management Systems. Requirements
prEN 50657	Committee draft for comment Railway Applications - Rolling Stock Applications - Software on board of rolling stock, excluding railway control and protection applications
RAIB Desborough	Rail Accident Report - Passenger door open on a moving train near Desborough, 10 June 2006 https://www.gov.uk/raib-reports/passenger-door-open-on-a-moving-train-near-desborough
RAIB Milton Keynes	RAIB review of the railway industry's investigation of an irregular signal sequence at Milton Keynes, 29 December 2008. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/528154/SI12010_101223_Milton_Keynes.pdf

Other relevant documents

Railway Group Standards

None.

RSSB Documents

None.

Other References

None.