DGUV
Deutsche Gesetzliche Unfallversicherung
Spitzenverband

IFA Report 2/2016e

Safety-related application software for machinery – The IFA matrix method

Authors: Michael Huelke (†)
Institute for Occupational Safety and Health of the German Social Accident Insurance, Sankt Augustin

Norbert Becker, Manfred Eggeling
Department of Electrical Engineering, Mechanical Engineering and Technical Journalism, Bonn-Rhein-Sieg University of Applied Sciences, Sankt Augustin

Published by: German Social Accident Insurance (DGUV)
Glinkastrasse 40
10117 Berlin
Tel.: +49 30 288763800
Fax: +49 30 288763808
Internet: www.dguv.de
Email: info@dguv.de

– December 2024 –

Database of publications: www.dguv.de/publikationen

Title image: ©Monkey Business – fotolia

ISBN (online): 978-3-948657-70-3
ISSN: 2190-7994

Available: Available from your accident insurance provider or at
https://publikationen.dguv.de/DguvWebcode?query=p022716

Abstract

Safety-related application software for machinery – The IFA matrix method

Manufacturers in the machine construction sector are increasingly using application programming of safety controls in order to implement safety functions. The current EN ISO 13849 and EN 62061 standards are the first to define requirements concerning the development of software employed for safety functions. The requirements are intended to prevent hazardous systematic errors in the safety-related application software employed for a machine. The essential requirement imposed by these standards is the observance of a structured development process: the V-model. The further requirements concerning measures for the avoidance and control of errors during development are also formulated in the standards in the usual very general terms. Furthermore, few examples and proposals for implementation of these requirements have been published to date. Interpretation of the standards during software development in machine construction is therefore often unclear, and presents difficulties during implementation. This situation prompted the launch of a project (FF-FP0319, project term 2011 to 2013) funded by the DGUV and conducted at the Bonn-Rhine-Sieg University of Applied Sciences. In the project, which was conducted in conjunction with machinery construction companies from the region, a development method suitable for application in the field – the IFA matrix method – was formulated and documented in a research report together with a number of examples. This research report forms the core of the present IFA Report. The IFA matrix method described here can be used to specify, validate and document the application software of safety functions in accordance with the standards. The report also provides further information on application programming for safety-related machine controls. Application programming for standard controls typically entails greater effort than for certified safety controls. Several chapters of the report therefore refer to the application of standard controls. In order for the IFA matrix method to be implemented efficiently, the IFA is developing SOFTEMA, a software tool. The examples in the report are available for download and can be viewed by means of SOFTEMA.

Kurzfassung

Sicherheitsbezogene Anwendungssoftware von Maschinen – Die Matrixmethode des IFA

Unternehmen im Maschinenbau realisieren Sicherheitsfunktionen immer mehr durch die Anwendungsprogrammierung von sicherheitsgerichteten Steuerungen. Die aktuellen Normen DIN EN ISO 13849 und DIN EN 62061 definieren erstmals auch Anforderungen an die Softwareentwicklung von Sicherheitsfunktionen. Dadurch sollen gefährliche systematische Fehler in der sicherheitsbezogenen Anwendungssoftware für eine Maschine vermieden werden. Wesentliche Anforderung dieser Normen ist, einen strukturierten Entwicklungsprozess einzuhalten: das V-Modell. Auch die weiteren Anforderungen zu fehlervermeidenden und -beherrschenden Maßnahmen bei der Entwicklung sind in den Normen wie üblich sehr allgemein gehalten. Zudem gibt es bislang wenige publizierte Beispiele und Vorschläge für die Umsetzung dieser Anforderungen. Daher ist die Interpretation der Normen bei der Softwareentwicklung im Maschinenbau oft unklar und bereitet Schwierigkeiten in der Umsetzung. Dies war der Anlass für ein von der DGUV gefördertes und an der Hochschule Bonn-Rhein-Sieg durchgeführtes Projekt (FF-FP0319, Laufzeit 2011 bis 2013). In dem Projekt wurde gemeinsam mit regionalen Maschinenbauunternehmen eine praktisch anwendbare Entwicklungsmethode – die Matrixmethode des IFA – hergeleitet und in einem Forschungsbericht mit vielen Beispielen dokumentiert. Dieser Forschungsbericht bildet den Kern des vorliegenden IFA Reports. Mit der hier dargestellten Matrixmethode des IFA kann Anwendungssoftware von Sicherheitsfunktionen normgerecht spezifiziert, validiert und dokumentiert werden. Darüber hinaus vermittelt der Report weitere Informationen rund um Anwendungsprogrammierung für sicherheitsbezogene Maschinensteuerungen. Der Aufwand für die Anwendungsprogrammierung ist bei Standardsteuerungen typischerweise höher als für zertifizierte Sicherheitssteuerungen. Daher beziehen sich mehrere Kapitel des Reports auf die Anwendung von Standardsteuerungen. Zur effizienten Anwendung der Matrixmethode entwickelt das IFA ein Softwaretool namens SOFTEMA. Die Beispiele des Reports sind zum Download verfügbar und können mit SOFTEMA betrachtet werden.

Résumé

Logiciels d'application pour machines relatifs à la sécurité – La méthode matricielle de l'IFA

Dans le secteur de la construction mécanique, les entreprises ont de plus en plus souvent recours à des logiciels programmables de commandes liées à la sécurité pour réaliser les fonctions de sécurité. Les normes actuelles DIN EN ISO 13849 et DIN EN 62061 définissent pour la première fois également des exigences applicables à la conception des logiciels qui commandent les fonctions de sécurité, le but étant d'éviter des erreurs systématiques dangereuses dans les logiciels d'application relatifs à la sécurité des machines. L'exigence essentielle de ces normes porte sur le respect d'un processus de développement structuré : le modèle en V. Comme de coutume, les autres exigences portant sur les mesures à prendre pour éviter et maîtriser les erreurs lors du développement restent aussi très générales dans les normes. De plus, il n'existe à ce jour que peu d'exemples et de propositions publiés sur la mise en pratique de ces exigences. C'est pourquoi, lors de la conception de logiciels dans la construction mécanique, l'interprétation des normes reste souvent floue et s'avère difficile à mettre en pratique. C'est ce qui a été à l'origine d'un projet subventionné par la DGUV et réalisé par l'université des Sciences appliquées de Bonn Rhein-Sieg (FF-FP0319, durée de 2011 à 2013). Dans le cadre de ce projet, une méthode de développement facile à utiliser dans la pratique – la méthode matricielle de l'IFA – a été élaborée avec des entreprises régionales de construction mécanique, et documentée dans un rapport de recherche comportant de nombreux exemples. Ce rapport de recherche constitue l'essentiel du présent rapport de l'IFA. La méthode matricielle de l'IFA qui y est décrite permet de spécifier, de valider et de documenter les logiciels de commandes liées à la sécurité, et ce en conformité avec les normes. Le rapport fournit en outre d'autres informations concernant la programmation d'applications pour les systèmes de commande de machines relatifs à la sécurité. En règle générale, la programmation d'une application sur un système de commande standard nécessite plus de travail et de coûts que sur les automates de sécurité certifiés. C'est pourquoi plusieurs chapitres du rapport sont consacrés à l'utilisation de systèmes de commande standard. Pour utiliser efficacement la méthode matricielle de l'IFA, celui-ci a mis au point un outil logiciel baptisé SOFTEMA. Les exemples du rapport peuvent être téléchargés et visualisés avec SOFTEMA.

Resumen

Software de aplicación de seguridad para máquinas – El método matriz de la IFA

Cada vez más, las empresas del ámbito de ingeniería industrial realizan funciones de seguridad mediante la programación de aplicaciones de controles relativos a la seguridad. Las normas actuales DIN EN ISO 13849 y DIN EN 62061 definen además por primera vez requisitos a tener en cuenta en el desarrollo de software para funciones de seguridad. De este modo se pretende evitar que se produzcan errores de sistema peligrosos en el software de aplicación de seguridad para una máquina. Un requisito esencial de estas normas es que se cumpla un proceso de desarrollo estructurado: el modelo en V. También el resto de los requisitos para el desarrollo con medidas para evitar y controlar fallos se describe en estas normas, como de costumbre, de manera muy general. Además, hasta la fecha se han publicado pocos ejemplos o propuestas sobre la implementación de estos requisitos. Por tanto, con frecuencia la interpretación de las normas en el desarrollo de software para maquinaria industrial resulta poco clara y genera dificultades a la hora de aplicarla en la práctica. Este es el motivo por el cual la DGUV promovió la realización de un proyecto, que se llevó a cabo en la universidad de Bonn-Rhein-Sieg (FF-FP0319, de 2011 a 2013). En este proyecto se derivó junto con empresas regionales de ingeniería industrial un método de desarrollo aplicable en la práctica, el método matriz de la IFA, que fue documentado en un informe de investigación con un gran número de ejemplos. Este informe de investigación constituye la base del presente informe de la IFA. Con el método matriz de la IFA aquí presentado se puede especificar, validar y documentar el software de aplicaciones para funciones de seguridad conforme a la norma. Además, el informe transmite otras informaciones en torno a la programación de aplicaciones para controles de maquinaria relativos a la seguridad. El trabajo que se invierte en la programación de aplicaciones es por lo general mayor para controles estándar que para controles de seguridad certificados. Por este motivo se dedican varios capítulos del informe a la aplicación de controles estándar. Para aplicar de manera eficiente el método matriz de la IFA, esta organización está desarrollando una herramienta de software denominada SOFTEMA. Los ejemplos del informe se pueden descargar en formato electrónico y se pueden consultar con SOFTEMA.

Contents

1 Foreword

Institute for Occupational Health and Safety of the German Social Accident Insurance (IFA)

Publication of the DIN EN ISO 13849-1 standard for control systems [1] was followed swiftly by BGIA Report 2/2008, "Functional safety of machine control systems" [2]. Like its predecessor, the report generated strong demand. The 15,000 printed copies of the German edition were soon taken. Together with further tools for application of the standard – the widely used SISTEMA software application and the "PLC disc" calculator – the IFA's report considerably facilitates new strategies for assessing and dimensioning the reliability of electronic and programmable control systems. This strategy, which analyses the probabilities of component failure, is enshrined in the DIN EN 61508 series of basic safety standards [3] and is now established in almost all sectors of industry, including machine construction. The preceding DIN EN 954-1 standard [4], with its purely deterministic requirements, has finally been replaced. The performance level is now firmly established in machine construction.

The reports and tools referred to above primarily deal with random failures of control systems, i.e. component defects and wear. However, even when very reliable hardware is used, a modern control system can still fail if care was not taken during development of the application software, such as the PLC program of a safety PLC. Lack of diligence in this area can lead to systematic faults and dangerous failure of safety functions during operation. The normative requirements for application software are however formulated in general terms, and published examples and guidance for their specific application in practice were for a long time difficult to obtain.

For this reason, the IFA also began work some years ago on the topic of "application software". In addition, a valued partner capable of implementing the normative perspective suitably in practice was found in the person of Professor *Norbert Becker* at the Department of Electrical Engineering, Mechanical Engineering and Technical Journalism at Bonn-Rhein-Sieg University of Applied Sciences. Between 2011 and 2013, Project FF-FP0319 concerning standards-compliant development and documentation of safety related application software in manufacturing system engineering [5] was successfully completed by Professor Becker, his laboratory team and numerous partner bodies from the machine construction sector, with funding from the DGUV (and thus also from the individual German Social Accident Insurance Institutions). It was intended from the outset that the research report would be published as the core part of the present IFA Report with the inclusion of numerous software examples. Professor *Norbert Becker* has since presented these research results in numerous lectures and company visits. In the process, he has prepared the ground for this report, for which we are grateful. The IFA matrix method described in this report is the result of project work by *Becker* et al.

Like those before it, this report is intended as a tutorial and a reference work for persons developing application software. It goes without saying that the report is not a substitute for application of the standards. However, the IFA considers the matrix method described here to be an appropriate implementation of the normative requirements. These requirements are very similar in the current standards addressing functional safety in the machinery sector, and this will probably remain so in the future. The IFA matrix method could therefore be applied to good effect with respect to all standards.

The report also contains valuable information on relevant aspects of safety-related application software for machinery control systems. As a follow-up to this report, the IFA is making its free "SOFTEMA" software application available. SOFTEMA is also a user-friendly solution for viewing and understanding the examples in the report. My hope is that like SISTEMA, this tool and the IFA matrix method will become an accepted independent standard by means of which machine control software can be specified, documented and validated reliably and efficiently.

Professor Dr *Dietmar Reinert*
Former Director of the IFA

Bonn-Rhein-Sieg University of Applied Sciences

There is little to add to Professor *Dietmar Reinert*'s foreword. The focus below lies on the history of the IFA matrix method, its application in industry and its technical aspects. How did project FF-FP0319, concerning standards-compliant development and documentation of safety related application software in manufacturing system engineering [5], which was kindly funded by the DGUV, come about?

For some time now, the author of this foreword has been addressing the application of modern safety technology in machine construction and in the process engineering industry. The key points were covered in Automation Technology lecture 2 at the university, as this was considered essential for preparing students for the modern industrial world. Conventional automation technology and modern safety technology have now coalesced.

Furthermore, cooperation with the IFA has been close for many years. In particular, Dr *Michael Huelke* recognized the continued benefit in providing industry with support in implementing the requirements of DIN EN ISO 13849-1 [1] with regard to safety-related PLC application software. This was the IFA's motivation for launching the funded project referred to above and for the sub-department of automation technology at Bonn-Rhein-Sieg University of Applied Sciences to conduct it.

One key objective of the author was for the results of the project to be suitable for direct implementation by industry. Academic vanity was to have no place in the results. For example, it is not practicable to model a machine as a finite automaton in order to be able to generate test cases from it when PLC safety software is implemented in a completely different way. For this reason, it was decided from the outset to create a user group through which local machine construction companies would be involved in the project, thereby enabling their current practices to be analysed, real-case examples from industry to be acquired, and results suitable for direct implementation obtained. We would once again like to thank Kautex Maschinenbau, Hennecke and Kuhne Anlagenbau and the IFA (Dr *Michael Huelke*). The VDMA, manufacturers (Siemens, Pilz, SICK) and the German Social Accident Insurance institutions were involved through a steering committee. The VDMA also facilitated presentation of the IFA matrix method at a user workshop on 8 November 2012. For this, too, I would like to express my thanks. I would also like to thank *Manfred Eggeling* Dipl.-Ing. (FH) for his detailed work on the project.

The IFA matrix method is not entirely new, and is already being applied in modified form in some parts of industry. A matrix lends itself to presenting safety-related switching processes clearly. The result is a very clear definition of all switching operations. Furthermore, this description can also be used directly as a basis for tests during validation, which avoids duplication of effort. Where required, additional test cases can easily be added. Can this description now also be used for specification of the safety-related PLC application software? Since all safety-related switching processes are described by the matrix, the answer is clearly “yes”. One key to the solution is that the safety-related PLC application software can always be structured in three parts: the inputs stage, the processing stage and the outputs stage – and that the first and last of these are already known; only the processing stage is unknown, and it can be specified easily by means of the IFA matrix method. The method presented in this report enables all present and probably all future normative requirements to be met elegantly. Put very simply, the normative requirements state that the path from definition of the safety functions to their verification, implementation and validation is to be structured and its details are to be comprehensible. An important milestone on this path is the software specification. If this milestone is rejected by users (you programme immediately!), there is no way of verifying (checking) the program code against the safety functions and against the software specification. The normative requirements would not then be satisfied.

The project describes the IFA matrix method by means of numerous examples set out in Excel® tables. This is of course very rudimentary. In 2017, the IFA made the free SOFTEMA software tool available, thereby making the method easy to apply. Dr *Michael Huelke* played a major role in driving SOFTEMA forward and implementing it. All project examples are also available for SOFTEMA.

Professor Dr *Norbert Becker*
Sub-department of automation technology, Department of Electrical Engineering, Mechanical Engineering and Technical Journalism, Bonn-Rhein-Sieg University of Applied Sciences

2 Introduction

Manufacturers of machinery are increasingly using application-specific programming of safety controls to implement safety functions. At one time, the requirements concerning the development of safety functions were defined in DIN EN 954-1 [4]. By the late 2000s, however, this standard had ceased to reflect the state of the art, and it was replaced by DIN EN ISO 13849-1 [1] and DIN EN 62061 [6], either of which can be applied. The new standards include definitions of requirements concerning the development of software employed for safety functions. These requirements are intended to prevent hazardous systematic errors in the application software used in a machine. How these new requirements are to be implemented in detail is not clear to the software developers of safety functions. This is partly because, by their nature, requirements in standards are formulated only in very general terms, and up to now virtually no examples of implementation have been published.

2.1 Software quality requirements

Like no other technology, software assumes greater responsibility than ever before. This naturally also applies to those programming it. A key new aspect of DIN EN ISO 13849-1 was the formulation for the first time of requirements concerning software for programmable control systems and the development of such software. The requirements in Section 4.6 of the standard enable safety-related application software to be developed for machinery control systems up to the highest risk level with the required performance level of PL_r e, i.e. including for hazardous machines such as presses or cutting machines.

Software completely without errors unfortunately does not exist in the real world. In contrast to hardware faults, which occur as a result of random component failure, the causes of software faults (errors) are systematic. It is therefore all the more important that all reasonable steps be taken for errors to be avoided during the development of safety-related application software, the purpose of which is, after all, that of minimizing risks. What is considered reasonable is geared firstly to the required performance level PL_r, and thus to the risk arising. At the same time, safety-critical errors are known to creep in during certain particular phases of software development, where they remain undetected until they cause a failure in operation of the machine, with particularly devastating effects. The phases of a machine's life in which this occurs are known to be those of specification, software design and, over the operation time, modification. The requirements of DIN EN ISO 13849-1 are therefore aimed in particular at error avoidance in these phases, as is the IFA matrix method presented here. Sadly, little attention is often paid in practice to these phases of application programming.

2.2 The DGUV's FF-FP0319 research project

In FF-FP0319 concerning standards-compliant development and documentation of safety related-application software in manufacturing system engineering [5] (2011-2013), the project partner, Professor Dr *Norbert Becker* (Bonn-Rhein-Sieg University of Applied Sciences), developed several specific procedures for implementing the requirements set out in the new standards concerning the development of safety-related software for machinery, and evaluated and documented these procedures with reference to industrial examples. The aim was to describe both the procedures and their application in a research report, which was then to be made available to the public as part of the present IFA Report.

Two committees were formed for evaluation of the project results during the project term:

- A user group, comprising local industrial companies (Kautex Maschinenbau, Hennecke, Kuhne Anlagenbau), the IFA and the TÜV Rheinland Academy
- The research support group, comprising control system manufacturers (Pilz, SICK, Siemens), German Social Accident Insurance Institutions, the IFA, the German Mechanical Engineering Industry Association (VDMA), TÜV Rheinland Academy, the Commission for Occupational Health and Safety and Standardization (KAN) and users
- The method was also presented and discussed at a number of industrial companies.

The project was divided into the following tasks:

- Development of a method
- Subsequent presentation of the method and evaluation of it by the user group and the research support group

Several methods for the specification of application software were studied:

- Description of application software as a finite state machine
- Specification by means of checklists
- Specification by means of tables/matrices

Describing the application software of a real-case machine as a finite state machine [7] in which all operating states are considered is generally a very complex process. Furthermore, the subsequent programming of the application software is completely different from its representation as a state machine in a graphical or text-oriented programming language. This

particularly applies to safety-related software, in which the use of certified function blocks is the norm. Procedures in which finite-state machines are described are not common in machine construction. By contrast, finite-state machines are used in the specification of complex safety-related function blocks (library modules) [8], which however was not the primary topic of this research project.

A checklist-based method was also developed. In this method, the safety functions are described by forms based upon checklists. These forms are progressively refined in the course of further specification. Following the demonstrations in the user group and the research support group, it soon became evident that the checklist-based method was also unsuitable for developing and documenting safety-related software in an industrial context. Many companies are however already documenting and specifying safety-related software in the form of tables. Based upon this activity, a matrix-based procedure for specifying and documenting safety software was developed. This met with much greater acceptance when presented to industry.

This procedure, described below as the “IFA matrix method”, was received positively by the user group and the research support group. The discussions resulted in numerous improvements to the presentation. Several examples were developed in this form of presentation, in order for as many cases relevant to practice as possible to be described. In addition, an example of a machine tool on a large scale was implemented, in order to demonstrate the suitability of the IFA matrix method for describing larger installations.

The IFA matrix method was presented, as an interim result of the project, to the public at the VDMA workshop on functional safety/safe application software in machine construction, held on 8 November 2012 in Frankfurt. The project team also subsequently issued several publications [9] and conducted numerous demonstrations in companies. These met with a largely positive reception and resulted in further constructive suggestions.

2.3 The purpose of the present IFA Report

The IFA matrix method presented here can be used for standards-compliant specification and documentation of application software for safety functions. Provided the procedure shown in the example here is followed, it can be assumed that the relevant requirements of DIN EN ISO 13849-1 concerning the safety-related application software (Section 4.6) for the safety functions are met.

Other methods with which the requirements can be met equally well doubtless exist besides this procedure. The IFA matrix method therefore lays no claim to being the only means of satisfying the requirements of the standards.

Application programming for standard controls typically entails greater effort than for certified safety controls. Several chapters of the report refer only to the application of standard controls, and are marked accordingly.

In addition to application of the matrix method, further details must be specified and checked in individual cases, such as:

- Manufacturer-specific parameterization of the peripheral devices used (e.g. variable-frequency drives, sensors)
- Further safety-related functions of the machinery control systems that are not covered by the IFA matrix method
- Additional miscellaneous functions that are not directly relevant to safety, such as special acknowledgement philosophies

3 Standards and the Report: an overview

DGUV project FF-FP0319 [5] and the present report address implementation of the requirements of the DIN EN ISO 13849 series of standards, comprising two parts [1; 10], these being the standards primarily applied in the machinery sector. Further applicable standards also exist, however. Of these, DIN EN 62061 [6] is also harmonized under the Machinery Directive. DIN EN 62061 is limited to electrical, electronic and programmable electronic systems; its suitability for many machines employing hydraulic and pneumatic parts of a control system is therefore limited. However, Section 6.11.3 of DIN EN 62061 formulates similar requirements and procedures for application software to those in DIN EN ISO 13849-1. For application software, the scope of the two standards is similar for control systems up to the highest safety integrity level for the machinery sector (up to PL e or SIL 3). The committees responsible for the two standards have now also examined the equivalence of the requirements and documented this equivalence in their joint report, DIN ISO/TR 23849:2010 [11]. The requirements vary in their details: as a sectoral standard of the DIN EN 61508 [3] series, DIN EN 62061 describes the management of functional safety very comprehensively, for example.

The IFA recommends the procedure described in this report expressly only with regard to compliance with the requirements of the initial version of DIN EN ISO 13849 and its Amendment 1. Nevertheless, this procedure may also be borderline suitable for DIN EN 62061.

3.1 Software categories and language types

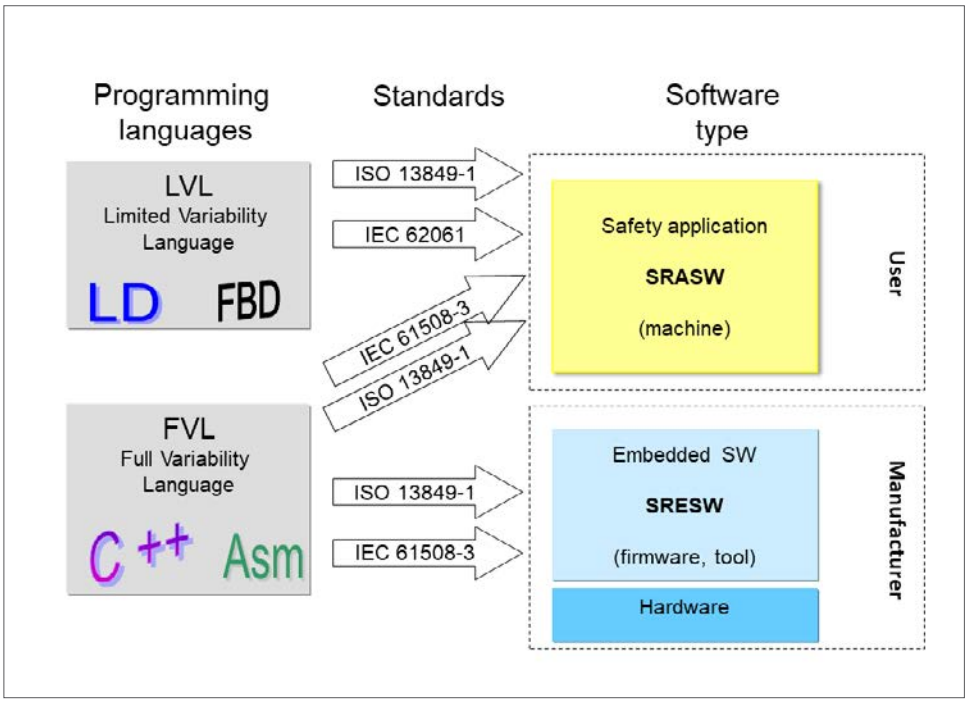
Safety-related software is a specific part of the software of a system (in this report, a machine, as shown on the right in Figure 1) that is used to execute safety-related control functions (in this report: safety functions) in a safety-related system (in this report: parts of a control system of a machine). This safety-related software takes several forms in terms of software types and programming languages. In this respect, the definitions of DIN EN (ISO) 13849 and 62061 are still largely consistent with each other.

Two types of software languages are defined in the standards:

- FVL: full variability language, see DIN EN ISO 13849-1 [1], Section3.1.35

  FVLs are a type of language with the ability to implement a wide range of functions. Examples are C, C++, Assembler. A typical example of systems for the use of FVLs are embedded systems (see below). In the field of machinery, FVLs are used in embedded software and occasionally in application software.

Figure 1:
Relationship between software languages, software types and the standards to be applied. Asm = assembler



- LVL: limited variability language, see DIN EN ISO 13849-1 [1], Section 3.1.34

  LVLs are a type of language with the ability to combine predefined, application-specific library functions to implement the specification of the safety requirements. Typical examples of LVLs (ladder diagram, function block diagram) are stated in DIN EN 61131-3 [2], the standard for PLCs. Programmable logic controllers (PLCs) are a typical example of a system employing LVLs.

Distinction is made as follows between software types:

- Safety-related embedded software (SRESW): see DIN EN ISO 13849-1 [1], Section 3.1.37

  This is software that is supplied by the control system manufacturer as part of the system, and cannot be modified by the user of the machine. SRESW is usually written in an FVL. Typical examples are firmware, operating systems, runtime systems, etc.

- Safety-related application software (SRASW): see DIN EN ISO 13849-1 [1], Section 3.1.36

  This is software implemented in the machine by the manufacturer specifically for the application. It typically contains logic sequences, limit values and expressions for processing and controlling the relevant inputs, outputs, calculations and decisions in order to meet the requirements of the safety-related part of a control system. A typical example is the PLC program of a safety PLC. SRASW is usually written in an LVL.

- Parameterization software: some safety-related parts of a control system require additional parameterization for application. Examples are frequency inverters with integral speed monitoring (SLS), for which the monitored speed must be entered. The part of the control system is usually parameterized by the user by means of dedicated parameterization software provided by the part supplier. This parameterization software also generates its own documentation.

  Safety-related parameters may also be input by means of project-specific solutions, such as a standard control device. In these cases, however, control system users themselves must ensure that parameterization is free of errors, and normative requirements of DIN EN ISO 13849-1 [1], Section 4.6.4 must be met.

The standards to be applied and thus the requirements applicable to software development depend on the type of programming language used (LVL or FVL) and the type of software (SRASW or SRESW) (see Figure 1).

This report focuses on application software of the LVL type and typically programmed in a PLC language (Figure 1 above). The requirements of DIN EN ISO 13849-1, Section 4.6.3, are therefore relevant. This report therefore uses the normative abbreviation SRASW for this application software.

Owing to the higher probability of systematic errors, SRASW programmed in an FVL, such as C, is treated according to the standard (DIN EN ISO 13849-1, Section 4.6.2) in the same way as embedded software.

The simplified V-model described in Section 4.6.1 of the standard is to be used as the development model for both SRESW and SRASW (see Section 5.2 of this report).

3.2 Requirements concerning safety-related application software (SRASW)

The development process is outlined in Section 4.6.1 of DIN EN ISO 13849-1. The normative requirements for SRASW for the software itself, the development tools used and the development activities are described in Section 4.6.3. These requirements also contribute to the fault avoidance. The overhead should be commensurate with the required risk reduction, analogous to that for the hardware of the programmable SRP/CS. The requirements and their effectiveness therefore increase accordingly with rising PL_r of the implemented safety function(s). DIN EN ISO 13849-1 thus states no maximum requirements which would be necessary for all items of software, irrespective of the PL_r. Figure 2 shows that a suitable package of basic measures exists in the first instance for all PL_r for SRASW (as for SRESW). The following basic measures described in Chapter 5 are already required for the development of software for PL_r a or b:

- Development life cycle with verification and validation (Section 5.2)
- Specification of the safety requirements (Section 5.6)
- Documentation of specification and design (Section 5.12)
- Modular and structured programming (Section 5.7)
- Functional tests (Section 5.10)
- Suitable development activities following modifications (Section 5.14)

For software employed for PL_r c to e, the basic measures are supplemented by additional measures for fault avoidance. The latter are required for PL_r c with lower effectiveness, for PL_r d with medium effectiveness and for PL_r e with higher effectiveness. Irrespective of whether the software now acts in only one or in both channels of a desired category, the PL_r of the implemented safety function(s) is always the yardstick for the requirements (see Section 4.3).

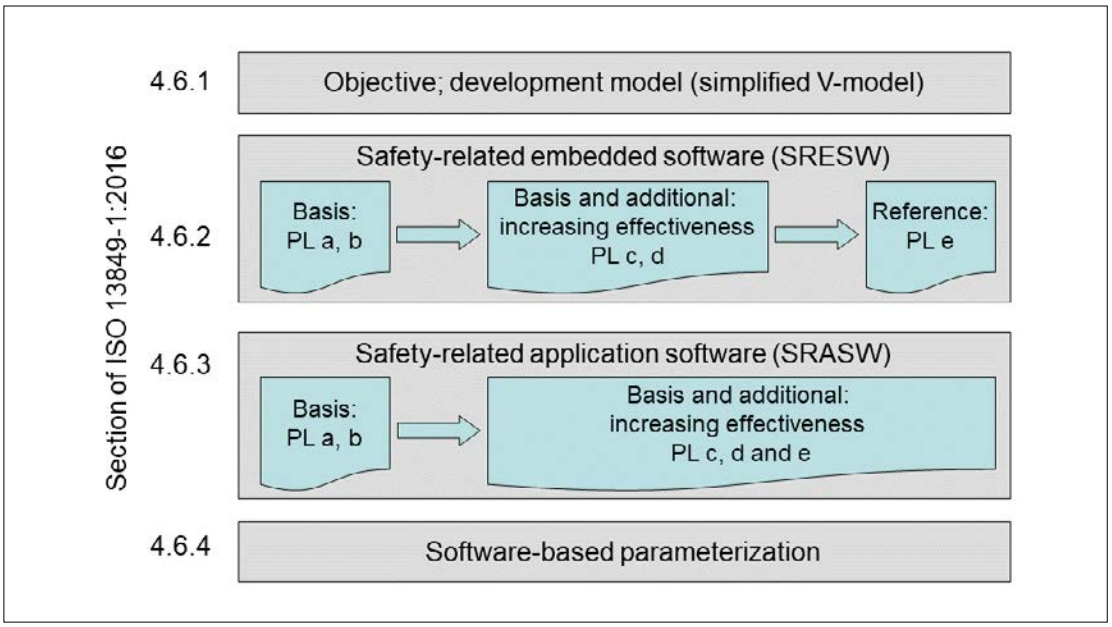
The aspect of increasing effectiveness refers to the increasing effect of fault avoidance. This will be illustrated here by the important task of “specification”. For PL_r c, for example, it may be sufficient for programmers to write the specification themselves and to cross-check it themselves later. Should the

same software be employed for PL_r e, however, a higher level of fault avoidance must be attained. It may then be necessary for the specification to be written by the software project manager, for example, rather than by programmers. In addition, the review of this specification could also be performed jointly by the programmer and an independent person, such as the hardware engineer. More eyes (generally) detect more errors.

Unfortunately, at the time of publication of this report, the authors are not aware of literature concerning forms of requirements with greater or poorer effectiveness. Each party applying the standard is therefore responsible themselves for specifying the form by which the requirements are met.

The normative requirements cited here are presented as part of the IFA matrix method and commented on for each project in a separate document, A4 – Requirements (see Section 5.5).

Figure 2:
Grading of the requirements for safety-related software (DIN EN ISO 13849-1)



3.3 Further informative content of DIN EN ISO 13849-1 concerning SRASW

In addition to the normative requirements referred to above, Part 1 of the standard contains further relevant content in the informative Annexes G and J. In this context, “informative” means explanations of the text of the standard, which in places is abstract in nature. The explanations reflect how the standard developers regard the standard: as a guide to typical implementation, which is to be considered as state of the art.

3.3.1 Annex G: Systematic failure

Errors in SRASW can lead to systematic failures. Informative Annex G discusses this type of failure. However, it essentially refers to control hardware, as the control software is already described in full in the normative Section 4.6.

A distinction is drawn in this annex between the avoidance of systematic failures and their control. Section G.2 specifies a failure control measure employing program sequence monitoring to detect faulty program sequences. In the case of a certified safety PLC, this monitoring is typically present in the firmware, where for example it detects a faulty SRASW sequence. In a standard control system, this measure would have to be reliably implemented in the SRASW itself.

Only the standard measures of functional testing, project management and documentation are referred to as measures for fault avoidance specific to integration of the control system (refer also to Chapter 5 of this report).

3.3.2 Annex J: Software

Figure J.1 in the informative Annex J begins by using a software example to illustrate the modularized three-stage structure employing function blocks: “acquisition of sensor data”, “processing [of the sensor data]” and “actuation of actuating elements”. This design principle is implemented by the IFA matrix method (Section 5.7).

Table J.1 of the annex shows an example compilation of activities and documents for application of the V-model. These activities and documents can also be found in the IFA matrix method.

“Verification of the software specification” in the annex addresses this aspect in detail. The annex evidently assumes that a software specification has been set out in text form. Specifications in text form easily give rise to omissions, inconsistencies and misinterpretation. This contrasts with the IFA matrix method, in which formal verifications are also possible in the tabular presentations.

The final part of Annex J contains examples of programming rules.

Such actual examples are rare, particularly since these rules relate in fact to SRASW.

The programming rules are documented and verified in the IFA matrix method in Table A3, “Measures” (example: see Table 9 in this report).

3.4 Relevant normative content of DIN EN ISO 13849-2:2013 concerning SRASW

Following development of the SRASW in accordance with DIN EN ISO 13849-1, it is validated in accordance with Part 2 of the series of standards [10]. Validation comprises several steps during and at the end of the development process. It is regarded as proof of suitability for purpose, specifically with respect to the actual application scenario. The safety-related parts of the machine control system are therefore examined by means of analyses and tests for their compliance with the safety requirements specified for them. The IFA matrix method also contains validation elements, by which it implements the relevant requirements of DIN EN ISO 13849-2. With the exception of Section 9.5, “Validation of safety-related software”, Part 2 of the standard makes no further direct reference to SRASW. The aspects of SRASW validation are dealt with in Chapter 12 of the present report.

DIN EN ISO 13849-2 also provides, in the form of a “toolbox”, a comprehensive list of measures that are to be applied to prevent systematic failure, such as the observance of basic safety principles and well-tried safety principles. The following safety principles for electrical systems can be applied to SRASW (from Annex D of the standard):

- Avoidance of undefined states: undefined states in the SRP/CS are to be avoided. The control system is to be designed such that its state, e.g. the outputs, can be predicted during normal operation and under all anticipated operating conditions.
- Direction of failure: wherever possible, all devices/circuits should fail to the safe state or condition.
- Reduction of possible faults: separation of safety-related functions from other functions.
- Balance between complexity/simplicity: a balance should be struck between the complexity of the equipment, by which better control is achieved, and simplicity of the equipment, by which its reliability is improved.

Part 2 also contains Table D.21, Faults and fault exclusions – Electronic components – Programmable and/or complex integrated circuits, with the statement that no fault exclusion can be made for faults in all or part of the function including software faults. Practical experience confirms this provision.

Annex E of DIN EN ISO 13849-2:2013 provides a concrete example of validation of a programmable control system, but without validation of the application software (refer to the comment in Annex E.1 of the standard). This example validation of the application software is presented in Section 12.3 of the present report.

4 Risk assessment and safety functions

This chapter describes how the safety functions and the scale of requirements for SRASW relating to fault avoidance are derived from the risk assessment.

4.1 Safety functions for risk reduction

DIN EN ISO 13849-1 is applied when a control system-based protective measure – the safety function – is to be designed and evaluated as part of risk reduction on a machine. Along with the control hardware and the embedded software, the SRASW also contributes to risk reduction, through the program implemented for the safety functions. Development of the SRASW can thus be regarded as part of risk assessment in accordance with ISO 12100 [13] and national and European primary and secondary legislation. The risk assessment process has already been described in BGIA Report 2/2008 [2], Chapter 5 and Annex A.

4.2 Definition of safety functions and their properties

Correct and complete definition of the safety functions and their properties is a prerequisite for creation of the SRASW, and thus for application of the IFA matrix method. Description of this process lies beyond the scope of this report; the reader is therefore referred to SISTEMA Cookbook 6, Definitions of safety functions [14] and to BGIA Report 2/2008 [2], Chapter 5 and Annex A. The outcome of this definition is the specification for the safety functions, which serves as the input document for the matrix method.

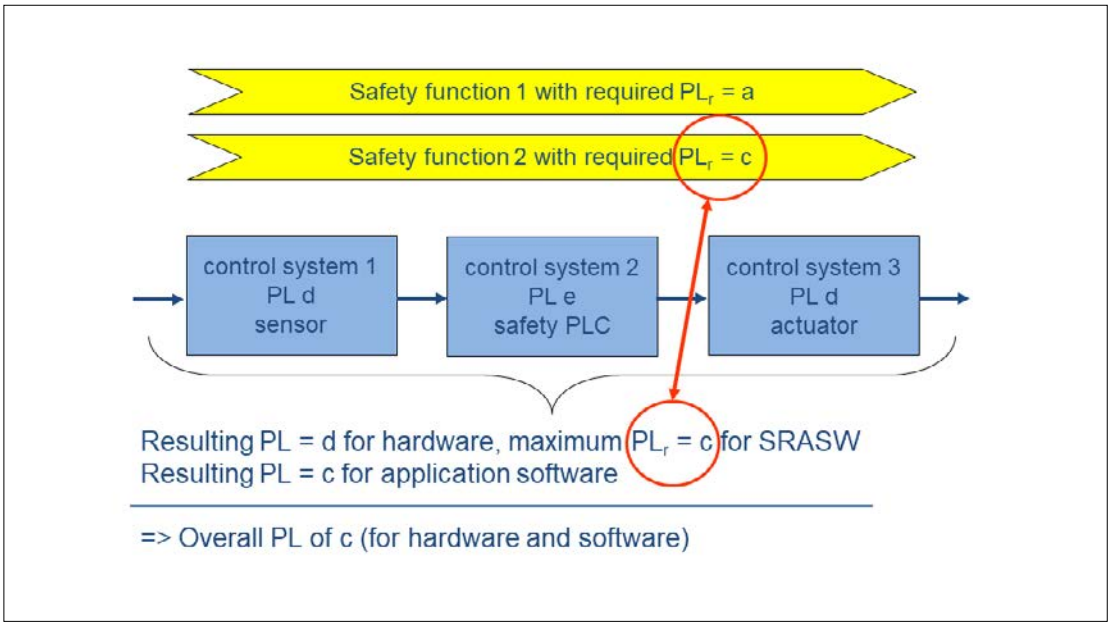
4.3 Influence of the risk assessment on software development

How are the normative requirements for SRASW now selected? DIN EN ISO 13849-1 [1], Section 4.6.3 states: "*For SRASW for components with PL_r from a to e, the following basic measures shall be applied*", followed by "*For SRASW for components with PL_r from c to e, the following additional measures with increasing efficiency [...] are required or recommended.*"

It thus follows that the requirements are based on the PL_r of the implemented safety function and not on the PL (in the sense of component property) of the part of the control system used. If, therefore, a safety PLC (part 2 of the control system) with a typical PL of e is specified, as for example in Figure 3, it does not follow that the SRASW must necessarily also be developed with the highest requirements for PL_r e. If this safety PLC is used to implement safety functions with a lower PL_r of c (medium risk; for safety function 2), the quality of the SRASW used on this PLC and therefore the requirements for its PL_r of c are then also sufficient.

Note: In Figure 3, the SRASW implements the two safety functions 1 and 2. The PL_r for the two safety functions differ (a and c). As the PL_r of safety function 2 is the higher of the two, it determines the requirements for the SRASW.

Figure 3:
Example of derivation of the requirements for SRASW



4.4 Influence of the software structure on software development

In most cases, multiple safety functions, quite possibly differing in their PL_r are implemented on a programmable controller. In theory, each safety function of the SRASW can then also be developed against different quality requirements according to its PL_r. However, this distinction is hardly practicable for typical SRASW, for which the IFA matrix method can be used. In this context, DIN EN ISO 13849-1 states in Section 4.6.3:

"If a part of the SRASW within one component has any impact (e.g. due to its modification) on several safety functions with different PL, then the requirements related to the highest PL shall apply."

How can it be demonstrated in a particular case that parts of the SRASW do not mutually influence each other and that different requirements within the SRASW therefore do apply? In such a case, the following must then be demonstrated and documented:

- The individual safety functions are independent of each other, both spatially and temporally, or
- Any violation of this independence is controlled

The reasoning for this independence must be documented. Typical characteristics for the independence of software modules and the technologies employed are described in Section 5.9.

In the first instance, the IFA matrix method is to be applied independently of the PL_r. In the interests of simplification, technologies and measures for fault avoidance are selected and applied according to the highest PL_r of the safety functions under analysis. This is supported by the SOFTEMA tool (Chapter 14).

4.5 Influence of the software structure on software development

Figure 3 shows that the PL for the entire combination of the three parts of a control system is always derived from two aspects:

- The reliability of the hardware: in this example, this results in a PL derived from the combination of the three parts of a control system with PLs of d/e/d. This yields an overall PL of d for the hardware in this example.
- The reliability of the software: assuming that the SRASW of parts 2 and 3 of the control system were to be developed against the requirements for a PL_r of c (i.e. would no longer satisfy PL_r d), the result would be the following situation for the safety function: $PL_{hardware}$ d with $PL_{software}$ c, resulting in a downgrade to an overall PL of c owing to the "poorer" software.

Should it subsequently transpire in this example that a further safety function with a higher PLr d needs to be implemented, this may have consequences for the application programs that have already been implemented. Owing to the requirement cited above, if the software safety functions are not separated from each other, they may all require "uprating" at a later stage. For this reason, the quality objective, i.e. the PLr specified for software development, should not be selected too narrowly. In the example in Figure 3, it would be advantageous for the software quality also to attain a PLr of d, consistent with the hardware quality. This would enable a safety function with a PLr of d to be added at a later stage without major overhead.

5 Measures for fault avoidance

When failures occur in a control system during operation, they are caused either by random faults resulting from ageing or physical phenomena, or by systematic faults that arose before commissioning, e.g. specification, implementation or manufacturing faults. In application software, only systematic faults occur. Should these systematic faults have negative consequences during operation, measures in the embedded software for the control of faults (e.g. cycle time monitoring by means of a hardware watchdog) may be able to bring about a safe state. However, it is better to prevent systematic faults from occurring in the first place. Measures for fault avoidance are used for this purpose during software development. This chapter describes and comments on the most important measures and techniques for this purpose set out in DIN EN ISO 13849-1. The following section first describes the typical procedure for combined project planning of control hardware and software.

5.1 Typical project procedure

The risk reduction process must consider a machine's entire life cycle. This includes the process for development of safety-related control systems. For safety-related software, this process is in fact defined very specifically by the "V-model" (Sections 5.2 to 5.5 of the present report). The sections from 5.6 onwards present further fundamental development aspects within the V-model for SRASW.

By definition however, software is always used in conjunction with control hardware to implement complete safety functions. The standard does not explicitly state how this combined development of hardware and application software with their resulting mutual influences should be performed. This section presents a bold description of a possible project process for the implementation of safety functions. Figure 4 shows an overview of the typical project steps.

Since the focus lies on implementation of safety functions, the sequence plan begins with specification of these functions. The system's design data and the specification for the safety requirements for the machine as a whole (often referred to as the safety concept) form the basis for the specification of the safety functions. This also includes the A2.1 System sketch. All document types referred to here, i.e. Ax.x, Bx.x, etc., are described in more detail in Section 5.5.

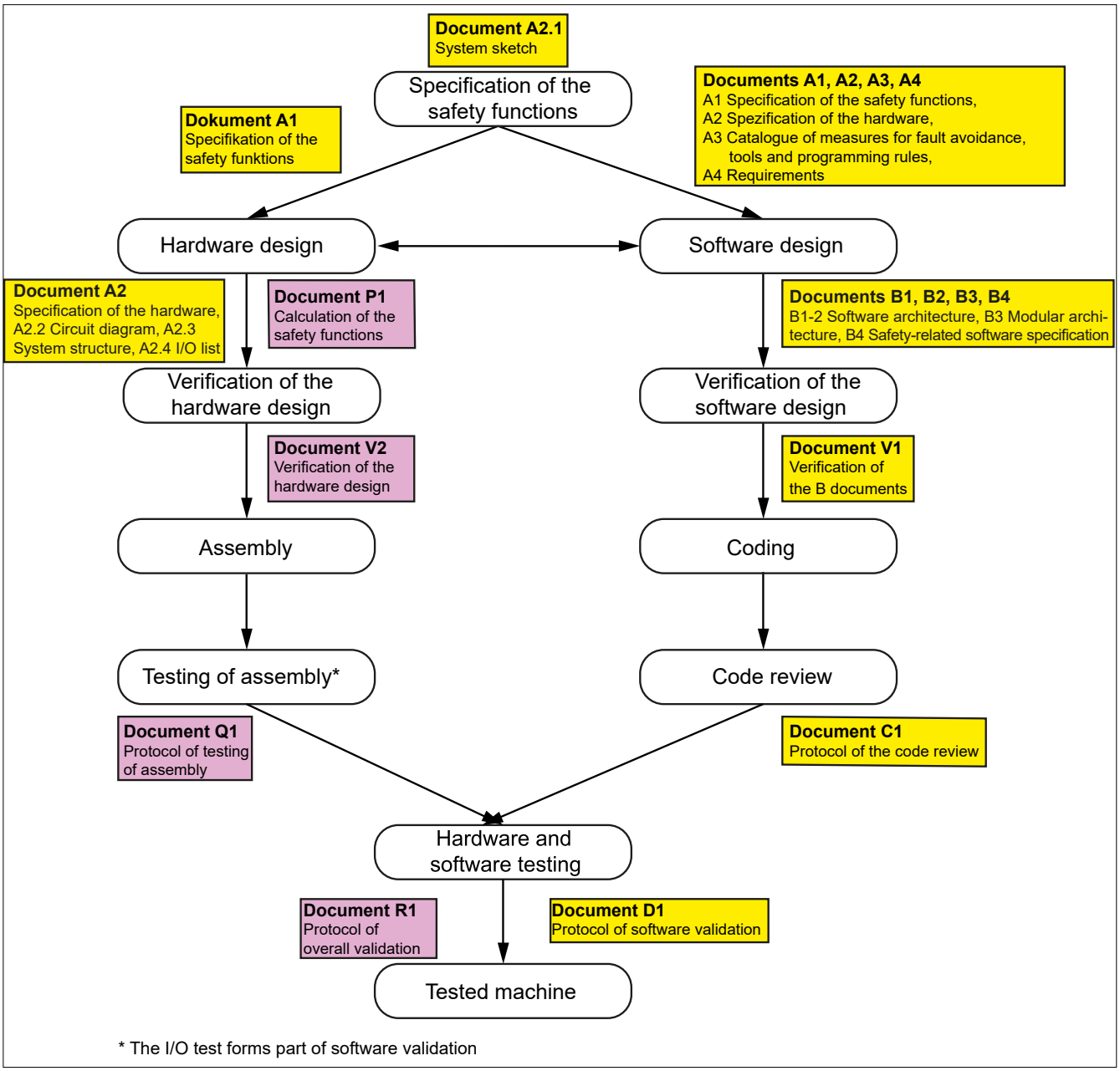
The project process is then divided into a hardware-oriented and a software-oriented branch. The documents required for application programming, A to D and V1, are highlighted in yellow. The additional documents generated during hardware design, which are not described in further detail by way of examples in this report, are highlighted in pink.

Development of safety-related parts of a control system begins with specification of the safety functions (document A1). Based on this specification, the circuit diagram (document A2.2), the system structure (document A2.3) and the I/O list (document A2.4) are created during hardware design. This is followed by calculation of the performance levels of the safety functions (document P1), for example by means of the IFA SISTEMA software [15] (refer also to BGIA Report 2/2008 [2]). Hardware planning is then verified against the specification of the safety functions (document V2). The verified hardware design documents form the basis for assembly. Assembly is checked and the result documented (document Q1).

The specification for the safety functions (document A1) and the hardware design documents A2.1 to A2.4 also form the basis for software planning. The catalogue of measures for fault avoidance, the tools and programming rules (document A3) and normative requirements (document A4) are also available and serve as planning information. The following documents are generated during software planning: safety program architecture (document B1), standard program architecture (document B2), modular architecture (document B3) and safety-related software specification with validation plan (document B4). These documents must then be verified against the specification for the safety functions (document V1). This is followed by the programming phase and checking of the program code during the code review (document C1). Some of the documents stated are optional.

Following inspection of assembly and review of the code, the hardware and software can be tested (documents D1 and R1). This is protocoled in test documents. It is followed by the overall validation, which is not shown separately in Figure 4. The individual verifications and tests should preferably be carried out by a second person with relevant qualifications, in accordance with the two-man rule (Section 5.15).

Figure 4:
Typical sequence plan for planning of projects involving programmable control systems

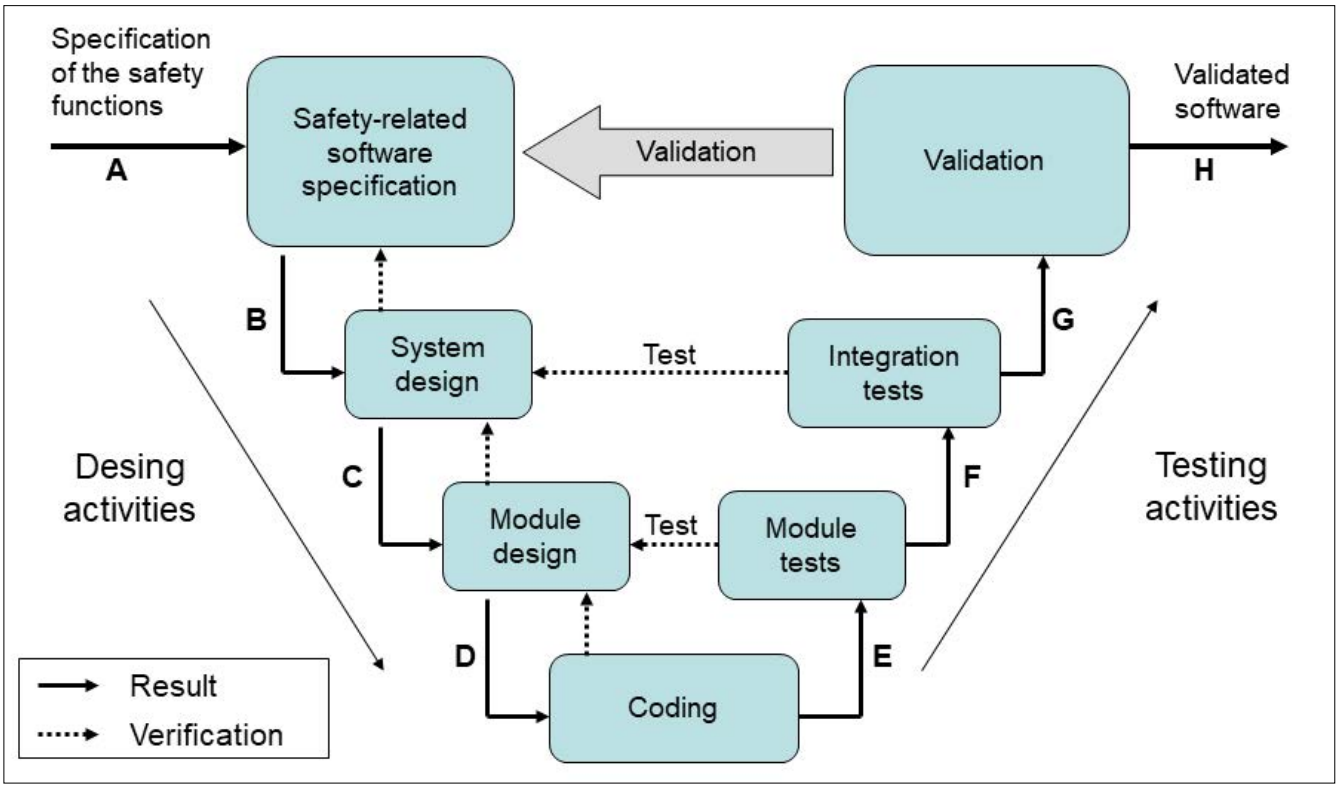


5.2 V-model development model

In the area of functional safety, the V-model presents a development process for safety-related software. It is consistent with several standards. It has existed for some time, having been introduced into software engineering by *Boehm* in 1979 [16]. The IEC 61508 series of basic safety standards has adopted this model and presented it in very detailed form. This was necessary, since the scope of this basic standard is unavoidably very broad: it extends from simple control devices for a manually guided machine to highly complex process control systems for process engineering plants. The latter require this level of detail in the development model.

However, where dedicated parts of a control system with integrated development environments – such as safety PLCs – are used in the machinery sector, the complex form of the V-model taken in IEC 61508 can be simplified. The "software architecture" phase can be omitted from application programming, as the software architecture is already determined by the control system's operating system and development tool. This could, for example, be an architecture compliant with the relevant PLC standard (DIN EN 61131-3 [12]), implemented by the control system manufacturer and certified by test bodies. Accordingly, the "integration tests" phase relating to the hardware can be omitted in the ascending, testing branch of the V-model. The application programmer still has the task of testing integration of the software modules with each other and within the specified software architecture. This is also the background to the "simplified V-model" in DIN EN ISO 13849-1, Section 4.6.1 (Figure 5).

Figure 5:
The simplified V-model in DIN EN ISO 13849-1, Section 4.6.1



5.3 Description of the V-model

This section summarizes the requirements of DIN EN ISO 13849-1, Section 4.6. A core requirement of the standard is that development of safety-related application software is to be carried out according to the simplified V-model (Figure 5).

Application of the V-model has two objectives:

- Avoidance of software faults (systematic faults)
- Development of readable, understandable, testable and maintainable software

The V-model can be characterized as follows:

- The V-model consists of design (Figure 5, left, including programming) and review activities, also termed “phases” (Figure 5, right).
- The result of each design phase must be verified against the specifications of the previous phase. The results are marked A to H in Figure 5.
- Each design phase has a corresponding review phase. The corresponding test plan has already been developed in the design phase.

A Clean documentation of the software created during all phases of the V-model serves to minimize errors in software development. The specification and the software design must be documented, to enable the program itself to be verified at a later stage against this design. This specification is to be accessible to all parties involved in software development. The specification includes a definition of the safety functions in which the required performance level PL_r and the associated operating mode are specified.

Programming should be modular and structured. Wherever possible, use should be made of validated function blocks provided by the manufacturer. During programming, great importance should be attached to code being readable and understandable. For this reason, symbolic variable names should be used and programming guidelines observed (see also Section 5.7). Understandable code makes testing of the program easier, and is conducive to the avoidance of further software errors during subsequent modifications.

Some requirements contained in the standard are met directly by the certified PLC programming environments:

- Use of limited variability programming languages (LVLs)
- Separation of safety software and purely functional software
- Provision of validated function blocks

Should changes be made to the software, appropriate development measures must be taken. This includes documentation of all modifications, and an impact analysis. The impact analysis can be used to estimate the overhead required for the activities that must be repeated according to the V-model. All modifications must be documented in a change history.

Verification comprises the analyses and functional tests for SRP/CSs and their sub-aspects, which determine whether the results attained in a development phase meet the specifications for the phase concerned. For example, the code review checks whether the application software complies with the specifications for the measures for fault avoidance.

The purpose of software validation is to demonstrate that the application software meets the specifications. Software validation consists of analyses and functional tests (see Chapter 12). The subsequent overall validation of the safety functions (not shown in the V-model) also examines whether the correct safety functions have been defined for the machine ("Has the right thing been built?").

Programmers who do not routinely develop safety-related application software are likely to consider these requirements tedious initially. However, they provide programmers with certainty of having developed the software to an adequate standard.

5.4 Simplification of the V-model for typical SRASW

Development of application software in accordance with the V-model as required by the standard can be simplified further still. The dashed lines in Figure 6 show the areas that can be grouped. These simplifications are subject to the boundary conditions stated below. Safety-related application software for machinery is usually created with a typical structure. It is divided into a inputs stage, processing stage and outputs stage (see Page 33). The inputs and outputs stages often make use of certified function blocks provided by the control system manufacturers in the form of libraries. Only the processing stage of the function blocks of the outputs stage need be specified and tested in more detail. The interconnection of all function blocks with the other peripheral signals is clear and simple. The system design is largely predetermined by this specified structure. This is the reason why the first and last phases are grouped, as the "system design" and its corresponding test phase, the "integration test", are not required in practice.

It is also advantageous for development by machine manufacturers of their own function blocks, where required, to be placed in a separate V-model. These function blocks, developed and validated in-house, are used in a later stage during programming of safety functions and treated in the same way as the library blocks provided by the control system manufacturers.

Grouping in this way results in two small V-models: one for the software development of complete safety functions, the other for the software development of individual software modules (function blocks) of SRASW (Figure 7). Individual software modules must often be developed in an application where a required functionality cannot be achieved by use of the library modules provided by the controller manufacturer. In this case, it is advantageous for this functionality to be encapsulated in a function block within a library, thereby enabling it to be re-used multiple times.

Figure 6:
The V-model with possible groupings (dashed lines)

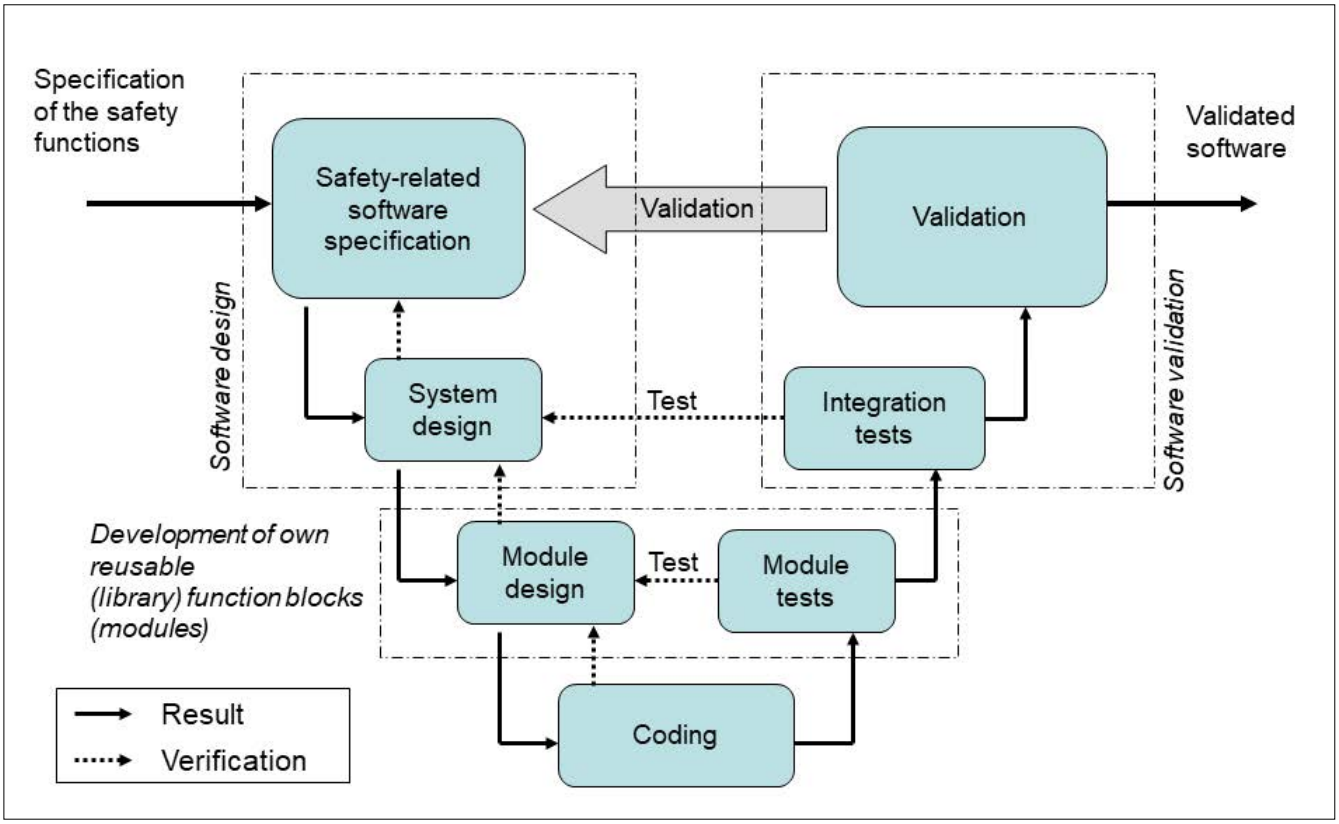
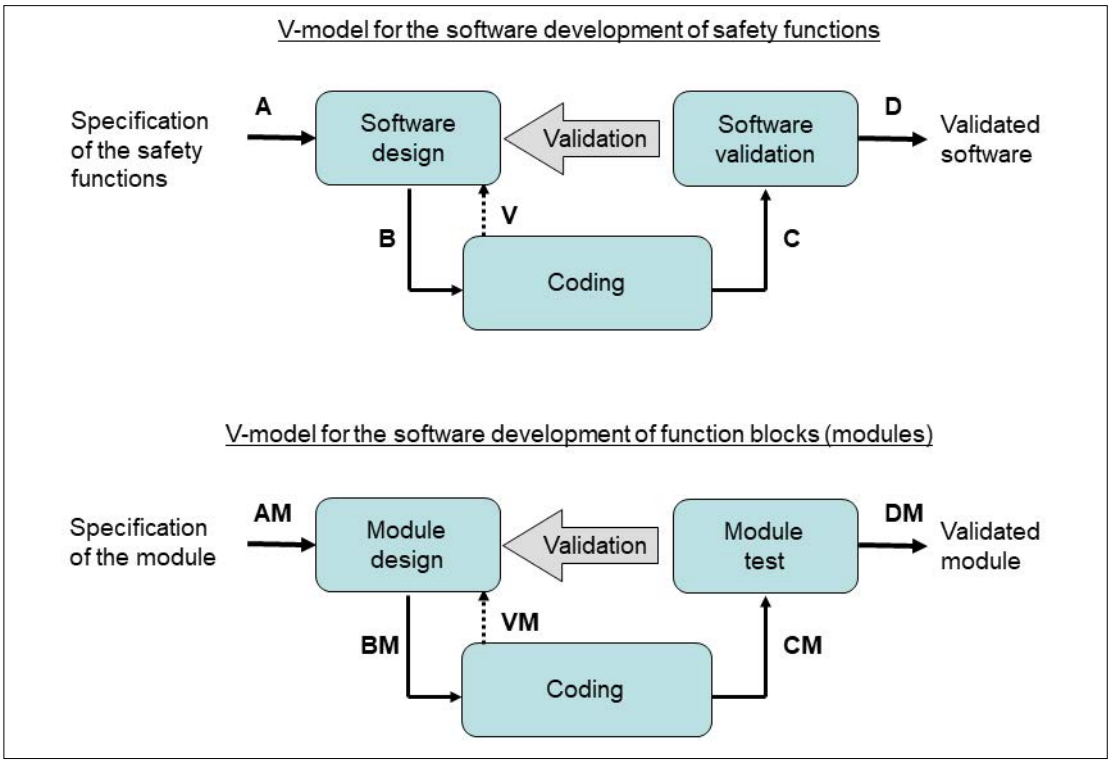


Figure 7:
Grouped V-models for safety functions (top) and modules (bottom)



5.5 Document types for the simplified V-model

Where software is developed in accordance with a V-model, it is practicable to define document types for the inputs and outputs of the phases. In the IFA matrix method, the letters A to D and V at the inputs and outputs of the phases in Figure 7 facilitate traceability of these document types. In the V-model for module development, the letter "M" is added in each case to avoid confusion. Tables 1 and 2 define the document types for the two V-models.

Table 1:
Document types for the V-model for software development of safety functions

Abbre-viation	Document	Comment
A1	Specification of the safety functions	Present
A2.1	Specification of the hardware, system sketch	Optional
A2.2	Specification of the hardware, circuit diagram	Present
A2.3	Specification of the hardware, system structure	Present
A2.4	Specification of the hardware, I/O list	Present
A3	Catalogue of measures for fault avoidance, tools and programming rules	Reusable
A4	Requirements	Reusable
B1	Architecture of a safety program	Not applicable for simple applications
B2	Architecture of a standard program	Not applicable for simple applications, optional
B3	Modular architecture	
B4	Safety-related software specification and validation plan	
B5	Program sketch	Optional
V1	Protocol of verification	
C1	Protocol of code review	
D1	Protocol of software validation	

Table 2:
V-model document types for software development of modules (function blocks)

Abbre-viation	Document	Comment
AM1	Interface definition and function description	
AM2	Catalogue of measures for fault avoidance, tools and programming rules	Reusable
AM3	Requirements	Reusable
BM1	Module specification and test plan	
BM2	Program sketch	Optional
VM1	Protocol of verification	
CM1	Protocol of code review	
DM1	Protocol of module test	

The documents that must always be created for a project, e.g. for hardware design, are marked “present” in Table 1. “Reusable” means that the document concerned can also be used in other projects; it need not therefore be created again for each project. In the present report, the optional documents are required for comprehensible explanation of the procedure; in the project itself, however, they can usually be dispensed with.

The documents are described briefly below:

- A1 – Specification of the safety functions: table of safety functions showing important properties such as the required performance level, response times, operating modes, priority, etc.
- A2.1 – System sketch: broad overview of the system; may also contain the design data
- A2.2 – Circuit diagram: electrical circuits, in particular for safety-related components
- A2.3 – System structure: overview of the safety components and their connection (topology showing the network)
- A2.4 – I/O list: list of all safety-related and where applicable other relevant inputs and outputs, together with their addresses and variable names; the I/O list also contains check fields for the code review and validation of the software
- A3 – Catalogue of measures, tools and programming rules for fault avoidance
- A4 – Requirements: listing of the normative requirements in accordance with
  DIN EN ISO 13849-1, Section 4.6.3, with project-specific comments
- B1 – Architecture of the safety program: overview of the safety program structure (call hierarchy of modules)
- B2 – Architecture of the standard program: overview of the standard program structure (call hierarchy of modules)
- B3 – Modular architecture: overview of the modules used (function blocks) with the interconnected inputs and outputs and the logic signals for the processing stage
- B4 – Safety-related software specification and validation plan: matrix-based specification of the software and test plan for software verification and validation
- B5 – Program outline: presentation of the software, e.g. in the function block diagram; usually corresponds to the future program listing
- V1 – Verification protocol: not a document in its own right, but check fields in documents B3 and B4, as these documents must be verified against the specification of the safety functions (A1)
- C1 – Protocol of code review: the software code must be checked for errors by review. The verification steps are listed and errors are to be documented in the protocol of code review
- D1 – Protocol of software validation

For module software development:

- AM1 – Interface definition and function description: description of the input and output variables of a module (function block) and its function
- AM2 – Catalogue of measures, tools and programming rules for fault avoidance
- AM3 – Requirements: list of the normative requirements of DIN EN ISO 13849-1, Section 4.6.3, together with project-specific comments

- BM1 – Module specification and test plan: corresponds to B4. Matrix-based specification of the function block with test plan (possible only with simple function blocks)
- BM2 – Program sketch: presentation of the software, e.g. in a function block diagram; usually corresponds to the future program listing
- VM1 – Verification protocol: not a document in its own right, but refers to check fields in document BM1, which document verification of the module specification BM1 against the function description AM1
- CM1 – Protocol of code review: the software code must be reviewed for errors. The verification steps are listed and errors are to be documented in the protocol of code review
- DM1 – Module test protocol: corresponds to D1. Protocol of validation of a function block

5.6 Specification of safety requirements and safety functions

Design and integration of safety-related parts of a control system must be geared towards products being developed that are as fault-free as possible and satisfy the safety requirements, and to these products being used as intended.

The life cycle of the safety-related parts of a control system (Figure 4) and thus of the SRASW begins with specification of the safety requirements for a machine. Section 6.1.1, Box 6.1 of BGIA Report 2/2008 [2] shows a general structure for a safety requirements specification.

The specification of the safety functions forms part of these safety requirements and is the first important document for SRASW development (refer also to SISTEMA Cookbook 6, Definitions of safety functions [14]). In addition to specific aspects of various safety functions, Chapter 5 of DIN EN ISO 13849-1 also lists general aspects that must be included in such a specification as a minimum requirement. Document “A1 – Specification of safety functions“ already contains this information, and can be expanded to include additional project-specific information. When the IFA’s SOFTEMA tool (Chapter 14) is used, any changes and extensions to specification A1 are inherited by the subsequent SRASW development documents.

A specification of this kind sets out, at the beginning of the development process (Figure 4), the framework for all parties involved. It constitutes a set of requirements specifications; in no way is it a product description produced following development.

5.7 Programming guidelines

Code that is written must be readable and clear, in order to facilitate testing and error-free modification at a later stage. This is guaranteed by suitable and binding programming guidelines and their observance during programming. The guidelines should be the existing and accepted rules of a recognized organization [8; 15; 17]. DIN EN ISO 13849-1, Annex J.4 also lists examples of rules. Alternatively, a company can compile suitable programming rules itself on the basis of these documents. In 2013, Promotional Committee 2 of PLCopen[1] launched a further initiative: the Software Construction Guidelines working group. These guidelines are already available.

Programming guidelines govern the use of critical language constructs, the scope and interfacing of software functions, the formatting and commenting of the code, symbolic names of functions and variables, etc. A separate table (document A3) is provided in the IFA matrix method for presentation and verification of the rules (for examples, see Section 6.5).

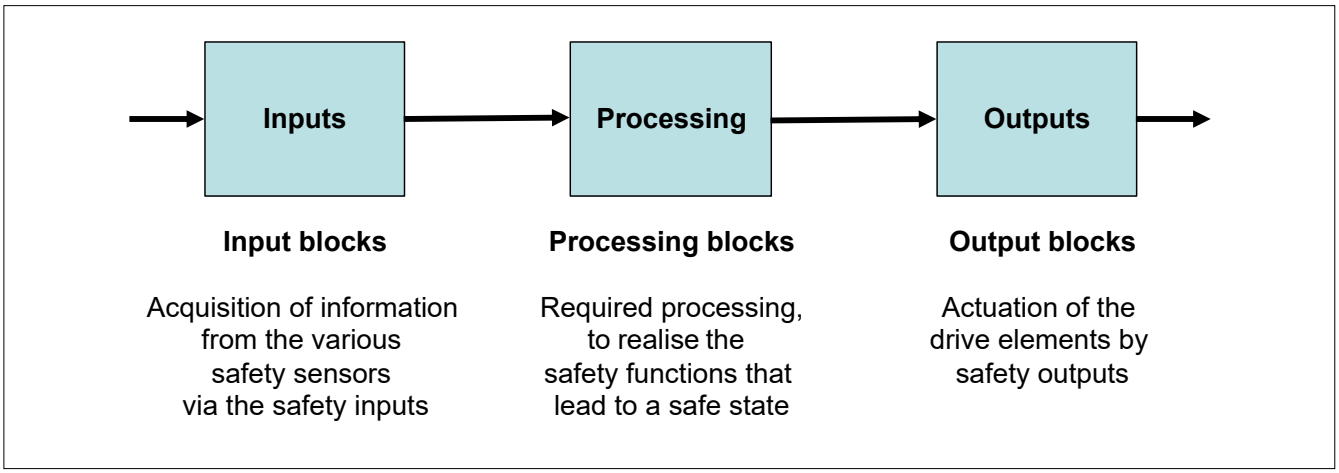
In an actual project, these generic rules are supplemented by manufacturer-specific programming rules and configuration measures addressing the particular features of the control system family and software architecture used (for examples, see Section 6.5).

5.8 Modular and structured programming

Modularity and structuring are fundamental principles for the avoidance and control of faults during the development of software for safety-critical systems. For this reason, they are specified as a basic measure by DIN EN ISO 13849-1, which also provides a software architecture model in three stages: Inputs -› Processing -› Outputs (Figure 8). These three stages are in turn to be implemented in most cases by function blocks.

[1] PLCopen is a global, non-proprietary and product-independent organization for the automation industry (www.plcopen.org)

Figure 8:
General architecture model for software (to DIN EN ISO 13849-1:2007, Figure 7)



How then are these two principles defined? The reader is referred here to the DIN EN 61508 [3] basic safety standard series, Part 7 of which provides an overview of procedures and measures and their definitions. From EN 61508:

Aim [of structured programming]*: To design and implement the program in a way that it is practical to analyse without it being executed. The program may contain only an absolute minimum of statically untestable behaviour.*

The following principles should be applied to minimise structural complexity:

- *divide the program into appropriately small software modules, ensuring they are decoupled as far as possible and all interactions are explicit;*
- *compose the module software control flow using structured constructs, that is sequences, iterations and selection;*
- keep the possible number of paths through a software module small, and the relation between the input and output parameters as simple as possible;
- *avoid complicated branching and, in particular, avoid unconditional jumps (goto) in higher level languages;*
- *where possible, relate loop constraints and branching to input parameters;*
- *avoid using complex calculations as the basis of branching and loop decisions.*
- *Features of the programming language which encourage the above approach should be used in preference to other features which are (allegedly) more efficient, except where efficiency takes absolute priority (for example some safety critical systems.* (DIN EN 61508-7:2011 [3])

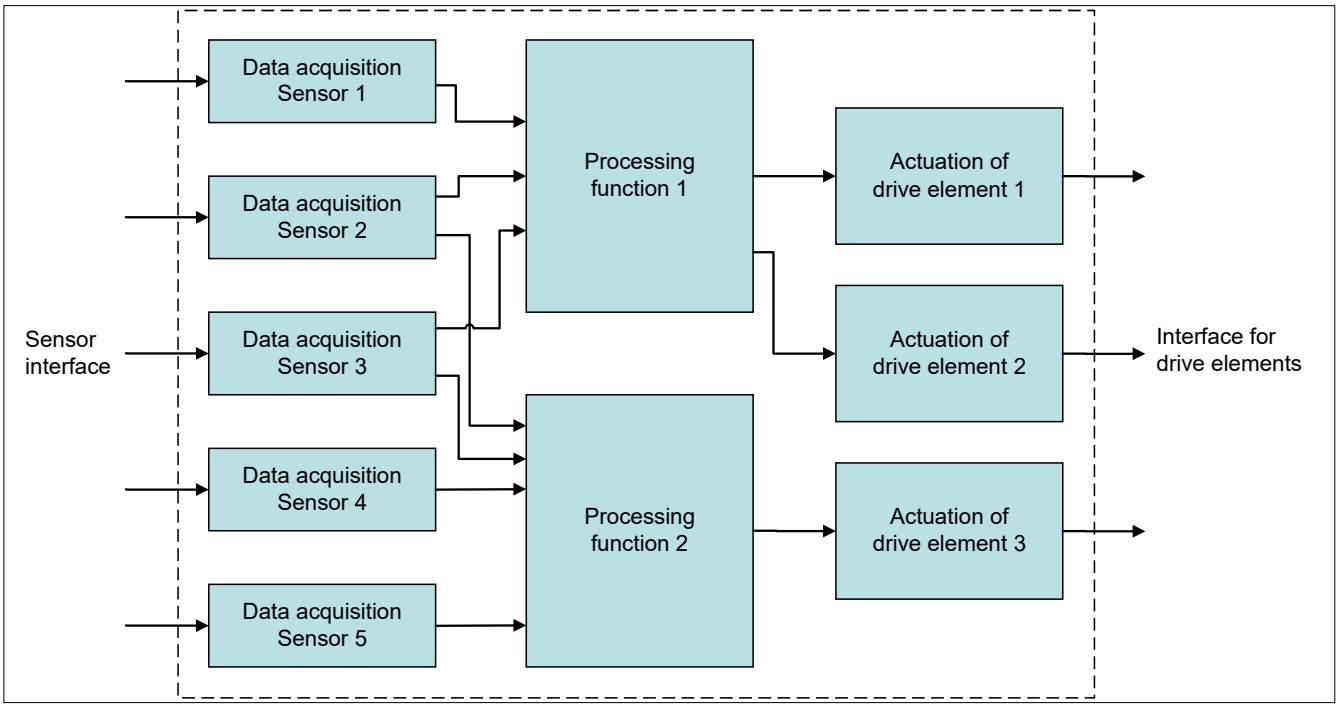
The principle of modularization is superordinate and relates more to the interfaces between the software modules:

Aim [of modularization]*: To reduce complexity and avoid failures, related to interfacing between subsystems.*

Description: Every subsystem, at all levels of the design, is clearly defined and is of restricted size (only a few functions). The interfaces between subsystems are kept as simple as possible and the cross-section (i.e. shared data, exchange of information) is minimised. The complexity of individual subsystems is also restricted. (DIN EN 61508-7:2011 [3])

These principles are assured when a PLC language is used on a safety PLC in observance of appropriate programming guidelines. With its three-stage structure and the use of small software modules (function blocks), the IFA matrix method promotes modularization and structured programming. The processing functions (or processing stage, see Section 6.1) should then also be broken down into individual modules, each relating to an actuator (as shown in Figure 9).

Figure 9:
Design of a software example (to DIN EN ISO 13849-1, Annex J))



5.9 Separation of safety-related and non-safety-related software

An important aspect of the structure of safety-related application software is its separation from the non-safety-related, functional process software. The latter may be modified more frequently, which should be possible without any dangerous impact on the safety functions (and possibly the need for validation or certification of the SRASW to be repeated). When certified safety controls are used, this separation is usually assured. The explanations below therefore primarily concern the use of standard components.

DIN EN ISO 13849-1 [1] refers in Section 4.6.3 d) to the following additional measure for fault avoidance:

"*d) Where SRASW and non-SRASW are combined in one component:*

1) SRASW and non-SRASW shall be coded in different function blocks with well-defined data links.

2) there shall be no logical combination of non-safety-related and safety-related data which could lead to downgrading of the integrity of safety-related signals, for example, combining safety-related and non-safety-related signals by a logical "OR" where the result controls safety-related signals."

The purpose of these requirements is that software modules must be executed independently of (or separately from) each other, both spatially and temporally.

Temporal separation means that a module must not consume excessive available processing time and thereby impair the function of another module. Nor must it prevent execution of the other module by blocking shared resources (memory, semaphores, communication, etc.). This requirement must be met by appropriate prioritization of the safety-related modules, sufficient resources, static memory allocation, etc., and must be monitored if necessary by measures for fault control (e.g. cycle time monitoring).

Spatial separation means that the data used in a safety-related module must not be changed impermissibly by another module, and in particular not by a non-safety-related module.

When is spatial separation assured? What techniques are typically used? Complete separation is not always possible, such as where actuator energization signals from the process software must be enabled in the SRASW. The options for coupling software modules that are intended to be independent are limited[2]. In such scenarios, the safety-related module must always retain full control over actuators giving rise to a hazard:

- Interface coupling: the safety-related module or its data is accessed only through subroutines of the module implemented for this purpose (e.g. a demand for a drive to be energized). The safety-related module alone decides what response is to be taken to the demand.

- Data coupling via a parameter list: the safety-related module or its data is accessed only through variables in the form of parameters of a subroutine. Each read/write operation of a variable can be recognized by the safety-related module and processed further by it without giving rise to hazardous situations.

[2] For further information, see DIN EN 61508-3 (VDE 0803-3):2011 [18], informative Annex F, "Techniques for achieving non-interference between software elements on a single computer".

The following forms of coupling are not recommended and should be used only in exceptional cases and in observance of the programming guidelines:

- Coupling through global data: safety-related modules employ global data to which other modules also have direct access. Understanding how the modules interact and assessing the impact of changes to the code may present difficulties.

- Control coupling: coupling that grants the non-safety-related module direct control over the safety-related module, for example by transmitting a bit that is capable of energizing a drive giving rise to a hazard. In this context, DIN EN ISO 13849-1 states explicitly: "*there shall be no [...] safety-related and non-safety-related signals by a logical "OR" where the result controls safety-related signals.*"

Two modules cease to be independent when their content is linked. For code changes to be evaluated, both modules must be understood. This means, for example, that:

- A jump occurs directly from one module to another

- One module influences the branch targets of another module

- One module is able to access the data of another module directly

Correct coupling and module separation can be evaluated only by a reading and understanding of the code, and must be documented as part of the code review (see Section 6.8). If sufficient separation from or independence of the non-safety-related module cannot be verified, it must also be treated as SRASW and included in the development process.

5.10 Functional test and expanded test

With regard to validation, the requirements of DIN EN ISO 13849 Parts 1 and 2 concerning SRASW differentiate between functional tests (as a basic measure) and expanded functional tests (as an additional measure). What is the difference?

During a functional test, the safety functions are checked as specified. For example, a safeguard is triggered and a check performed to ensure that the correct drive stops. Faults in the peripherals or errors in the software are not assumed or incorporated into the test. This is reasonable, since this basic measure is required for PL_r a and b and these PL_r are usually implemented with untested category B architectures. Where a test has not been programmed, it cannot be tested.

Expanded functional tests are a measure supplementing the functional tests and are required for the higher PL_r c to e. These PL_r can be attained only by the use of category 2, 3 and 4 programmable systems with diagnostic functions. Diagnostic functions are generally also implemented in the SRASW (direct/indirect monitoring, cross-monitoring, plausibility check, etc.).

The definition from in the EN 61508-7:2011 [3] basic safety standard is as follows:

Aim [of expanded functional testing]: To reveal failures during the specification and design and development phases To check the behaviour of the safety-related system in the event of rare or unspecified inputs.

Expanded functional testing reviews the functional behaviour of the safety-related system in response to input conditions which are expected to occur only rarely (for example major failure), or which are outside the specification of the safety-related system (for example incorrect operation). For rare conditions, the observed behaviour of the safety-related system is compared with the specification. Where the response of the safety-related system is not specified, one should check that the plant safety is preserved by the observed response.

In the expanded functional test, faults are typically generated or simulated in the control system itself or in its peripheral devices. This enables the diagnostic functions in the SRASW, which have been implemented in the input and output blocks (Figure 8), to be tested, together with their fault response. In this context, it may be difficult to determine that the safe state of the control system was in fact brought about by the diagnostic function of the SRASW under test, and not by the embedded software.

For PL_r d and e, it is recommended that expanded test cases based on limit value analyses be performed in addition. This is generally advantageous only when the input values are analog values and different ranges of these input values are formed. For example: a temperature sensor supplies values between -20 and +100 °C. Relevant outputs are driven according to the temperature range. Three ranges are specified:

- Too cold: -20 to +9.99 °C

- Temperature OK: +10 to +29.99 °C

- Too hot +30 to +100 °C

The limit value analysis of this specification yields the extreme values of -20 and 100 °C and range boundaries of 10 and 30 °C. Tests should then be carried out with input values at or around these extreme values and boundary limits, as this is where processing errors may be suspected.

Where applicable, borderline timing situations should also be tested, such as time monitoring with borderline delays close to the nominal monitoring time.

In addition to the functional tests of the safety functions, the matrix method also enables other test cases to be defined and validated in accordance with the methods referred to above.

5.11 Test coverage

How much must be tested and for how long? The criterion here is the test coverage, which is an important measure of software quality. The test coverage states the proportion of the software

executed by the entirety of the test cases. A higher number of judiciously selected test cases can improve the test coverage and thus also the software quality. However, this also increases the overhead of testing. 100% test coverage of all software elements would be ideal; in practice, the required test overhead is geared to the required safety integrity, the PL_r.

The DIN EN 61508-3:2011 [3] basic safety standard provides very specific guidance on this aspect in Table B.2 and will be used here for interpretation in relation to the IFA matrix method (Table 3). These are test methods that require knowledge of the program code and do not therefore correspond to the black box testing required by DIN EN ISO 13849. They nevertheless permit conclusions regarding an attainable test coverage for processing in the SRASW (Figure 8), depending on the scope of the test cases.

Table 3:
Recommendations for test coverage

Test coverage based on IEC 61508-3	PL_r SIL	Scope of test cases in black box testing of the SRASW
100% of the inputs	a, b and c SIL1	A demand must be made at least once to each safety-related input, i.e. to all safeguards and thus the function blocks of the inputs stage, and the re-start tested. Objective: to ensure that each subroutine of the SRASW, including the function blocks of the processing stage and outputs stage, has been called at least once. However, the scope of testing stated in the next row for PL_r d is also recommended for this PL_r.
100% of the program statements	d SIL2	Demand all safety functions in all operating modes and test the re-start. Objective: to ensure that all statements in the processing stage of the SRASW (Figure 7) have been executed at least once. Includes PL_r a, b, c.
100% of the program branches	e SIL3	Demand all safety functions in all operating modes, test the re-start and test all diagnostic functions by means of fault simulation. Objective: to test both possibilities of each branch in the processing stage of the SRASW. Includes PL_r a, b, c, d.

Certified manufacturer's function blocks and function blocks that have already been validated no longer need to be tested; testing of their combination and parameterization is however still required. Function blocks that have not yet been validated must be tested and validated separately during module development (see Section 5.10).

The test coverage attained can be documented in the IFA matrix method and in SOFTEMA in the test columns of the matrix presentations (B4, B5).

5.12 Documentation

Before the manufacturer issues the EC declaration of conformity for a machine, he must draw up its technical documentation. However, this should not be regarded merely as a tedious duty. Proportionate internal documentation is also beneficial to the company's own sustainable development work and serves as a safeguard for the event of legal conflicts. DIN EN ISO 13849-1 is brief but clear regarding documentation of SRASW:

"*all lifecycle and modification activities shall be documented; documentation shall be complete, available, readable and understandable.*"

The IFA matrix method and SOFTEMA tool support adequate development documentation, as described in Chapter 6 and illustrated by the numerous examples in Chapter 7. Chapter 13 deals separately with the topics of "technical documentation" and "user information".

5.13 Configuration management

What is meant by configuration management? It is evident, and must therefore be a requirement, that development of safety-related software, in particular, should be transparent to all parties involved and for subsequent audits:

- Who specified, programmed, commissioned, verified and validated it, and when?
- What was used for its development, e.g. what tools and settings, re-used software modules and identification, programming guidelines?
- What manuals are suitable for the development tools?
- Which program versions are loaded in which SRP/CSs?

This and other necessary information, including all relevant development documents, must be archived for later use, for example for the event of modification after five years in operation. Use of the matrix method and SOFTEMA tool results in all this information being protocoled and archived in a standard format (Microsoft Excel). This means that data suitably and permanently archived can still be accessed years later.

5.14 Modifications

Experience has shown that even after testing, SRASW will still undergo not inconsiderable extension and adaptation during commissioning of an installation or machine. This procedure is termed "modification". These changes are often so extensive

that not only the code, but even the original specification is no longer appropriate and should in fact be revised. Changes to safety functions at one point of the installation or machine may have an impact on the remaining safety functions, which have not yet been modified. Modifications may also reveal deficits in the safety concept. This possibility should be examined, and the necessary phases of the V-model repeated if appropriate.

Practical experience also shows, however, that even after it has been installed, a machine or installation may still occasionally require addition of an emergency stop button or guard door, for example. Frequently, the machining process itself is improved, in turn requiring adaptation of the safety concept. The existing software must be "modified". Note: this may be the case on SRP/CSs that have already been in operation for a longer period of time and for the most part without failures being caused by software faults – which could equally mean that a fault that is present but "hidden" has simply not yet taken effect. Following a modification, however, this situation may change, for example if the software was not adequately structured and individual modules/functions are not therefore entirely without reciprocal influence.

In the situations described, "Murphy's Law" often takes effect: the program was written many years previously and the original programmers now have more pressing tasks or have since changed employer. Under such circumstances, it is in the interests of the machine's availability as well as safety for the software to possess the properties stated above: readability, structure, intelligibility, and also conduciveness to straightforward, non-error-prone modification – irrespective of whichever programmer happens to be available.

In principle, a modification means that the development process must be restarted, i.e. in the V-model (Figure 5), at the point in the process at which a change was made, for example:

- When the code has been changed, the code review, integration test and validation may have to be repeated.
- If required changes extended to the specification, it too must be verified again, for example by review by another person, in order to ensure that no faults have crept in at a different point in the specification. Accordingly, all development and verification measures and also validation of the affected safety functions must be repeated.

The IFA matrix method supports these modifications (Section 6.15) and SOFTEMA automates the relevant changes in the development documents.

5.15 Two-man rule and degrees of independence

Checking activities such as verification and validation are intended to assure conformity of the design of the SRP/CS with the Machinery Directive. These activities should be begun as early as possible during development, in order for faults to be detected and eliminated in time. The outcome of programming should not, however, be checked by the same person who wrote the code; the two-man rule applies here. In practice, this often raises questions. When is the two-man rule needed, if ever? How independent must the second person be? Can the task be assumed by someone from the same team?

DIN EN ISO 13849-2:2013, Section 4.1 makes the following recommendation: "*Validation should be carried out by persons who are independent of the design of the safety-related part(s).*" These may be independent persons, or persons from independent departments or independent organizations (see definitions in Table 4). This report also introduces the term "different person". This merely means a person other than the one whose outcome (specification, code, etc.) is to be checked.

The degree of independence should be commensurate with the risk, i.e. the required performance level PL_r. Part 2 of the standard adds in a comment: "*Independent person does not necessarily mean that a 3rd party test is required.*" Unless specified to the contrary by statutory requirements, the review is therefore typically still conducted "in-house". Nevertheless, external test bodies are often consulted for advice and evaluation of technical and organizational measures. Table 4 shows the IFA's recommendation for the degrees of independence.

DIN EN 61508-1 (VDE 0803-1) [3], Section 8.2.16 (1), Note 1 states:

Depending upon the company organization and expertise within the company, the requirement for independent persons and departments may have to be met by using an external organization. Conversely, companies that have internal organizations skilled in risk assessment and the application of safety-related systems, that are independent of and separate (by ways of management and other resources) from those responsible for the main development, may be able to use their own resources to meet the requirements for an independent organization.

Table 4:
Definition of the degrees of independence for SRASW (based on DIN EN 61508-4:2011)

	Definition (example) as a recommendation of the IFA
Different person	Person who was not the one who performed the activities to be reviewed, but who may be involved in the project or bear responsibility (yes: project manager, supervisors, participants in the activity under review)
Independent person	Person who is not involved in the activities to be reviewed and has no direct responsibility for these activities (no: project manager, superiors, persons involved in the activity to be reviewed; yes: hardware designers for review of SRASW development; commissioning engineers for review of project planning activities)
Independent department	Department not in contact with the project/development departments responsible for the SRASW activities (person from the QA department/CE department/panel building, etc.)
Independent organization	Organization not in contact with the development organizations, owing to its management and other resources (different business unit, different company, test body)

With these definitions, the IFA provides a detailed recommendation in Table 5 for the minimum degree of independence, which is dependent upon the relevant PL_r of the SRASW development. The recommendation is based on the DIN EN 61508-1:2011 basic safety standard, Table 5, but takes into account that the safety functions falling within the scope of DIN EN ISO 13849-1 typically exhibit lower complexity and thus a lower probability of failure than those within the typical scope of DIN EN 61508-1. For this reason, the independence required for SRASW in Table 5 is reduced by one degree from that in DIN EN 61508-1.

Table 5:
Degree of independence for verification and validation (SRASW)

Minimum degree of independence for verification and validation for SRASW *, IFA recommendation	Relevant PLr for SRASW development *			
	a and b	c	d	e
Different person**	Recommended for all SRASW	For all SRASW	Standard SRASW	Not sufficient
Independent person	Possible, but not necessary	Possible, but not necessary	Complex or new SRASW	Standard SRASW
Independent department	Possible, but not necessary	Possible, but not necessary	Possible, but not necessary	Complex or new SRASW
Independent organization***	Possible, but not necessary	Possible, but not necessary	Possible, but not necessary	Possible, but not necessary

* Based on the DIN EN 61508-1 basic safety standard [3], Table 5

** Lowest degree of independence

*** Highest degree of independence

The green fields in Table 5 refer to development of SRASW with the use of PLC languages on a safety PLC where experience exists with the type of design and the technology used and the programs do not exhibit a higher degree of complexity – i.e. as is the case for typical users of the matrix method. For the "standard SRASW" considered in the present IFA report, this means that for PL_r c and d, a "different person" is sufficient to meet the two-man rule; for PL_r e, an "independent person" is recommended. It should be borne in mind that a fault may pose a threat to human life, even with PL_r c. The two-man rule is recommended even for for PL_r a and b.

The yellow table fields refer to all other error-prone situations, in which, for example, very large or highly complex programs are developed, or a new type of software architecture or new technology is used (FVL languages, unfamiliar controllers, software architectures developed in-house, etc.). The IFA matrix method will also not usually be suitable in such situations.

The grey table fields represent degrees of independence that are possible, but not necessary. If a lower degree of independence is selected than in Table 5, the reasoning for this should be stated.

5.16 Project management

For the development of SRESW (embedded software) for a PL_r of c or higher, DIN EN ISO 13849-1 requires a "project management and quality management system comparable to, e.g. IEC 61508 or ISO 9001". Conversely, in-house project management is not explicitly required in connection with the requirements for SRASW; the V-model is however specified for process organization. At another relevant point, project management – not specified in greater detail – is recommended: in Annex G.4, Measures for avoidance of systematic failures during SRP/CS integration. Here too, however, SRASW is not the direct focus.

From this it can be inferred that implementation of the V-model as a development life cycle for SRASW, with verification and validation (Section 5.2) in observance of the two-man rule (Section 5.15), is an appropriate error-avoidance measure for quality assurance. Application of the matrix method supports this quality assurance. Beyond this measure, a "*project management and quality management system comparable to, e.g. IEC 61508 or ISO 9001*" is not required for SRASW.

Nor does DIN EN ISO 13849-1 provide any definition of its own for "project management". For the sake of completeness, the reader is therefore referred here to a description in the DIN EN 61508-7 basic safety standard [3], Section B.1.1:

Aim: To avoid failures by adoption of an organisational model and rules and measures for development and testing of safety-related systems.

Description: The most important and best measures are

- *the creation of an organisational model, especially for quality assurance which is set down in a quality assurance handbook; and*
- *the establishment of regulations and measures for the creation and validation of safety-related systems in cross-project and project-specific guidelines.*

A number of important basic principles are set down in the following:

- *definition of a design organisation:*
  - *tasks and responsibilities of the organisational units,*
  - *authority of the quality assurance departments,*
  - *independence of quality assurance (internal inspection) from development;*
- *definition of a sequence plan (activity models):*
  - determination of all activities which are relevant during execution of the project including internal inspections and their scheduling,
  - *project update,*
- *definition of a standardised sequence for an internal inspection:*
  - *planning, execution and checking of the inspection (inspection theory),*
  - *releasing mechanisms for subproducts,*
  - *the safekeeping of repeat inspections;*
- *configuration management:*
  - *administration and checking of versions,*
  - *detection of the effects of modifications,*
  - *consistency inspections after modifications;*
- introduction of a quantitative assessment of quality assurance methods:
  - *requirement acquisition,*
  - *failure statistics;*
- *introduction of computer-aided universal methods, tools and training of personnel.*

5.17 External testing of SRASW

The Machinery Directive and the relevant national product safety legislation oblige manufacturers, importers and dealers of machinery to comply with comprehensive occupational safety and health requirements. Failure to meet these requirements may have far-reaching consequences, such as product liability cases or subsequent demands by the labour inspectorates.

Among the requirements of the Machinery Directive for conformity assessment of particularly dangerous machinery, i.e. safety-related components listed in Annex IV, is that a notified testing and certification body designated by an EU member state must be charged with the task. For the vast majority of machines, however, external testing by a test body is performed at the manufacturer's own volition, and provides certainty that the products and technical documentation comply with the national and European health and safety requirements.

This external testing also has advantages for operators. Who is able to assess the safety of a product exactly at the purchase stage? Often, there is little alternative to trusting the manufacturer. This may prove expensive if safety deficits are encountered during subsequent operational use. Tested products guarantee that the customer is purchasing a product that is safe from a technical perspective – for the benefit of both the company and its employees.

The testing and certification system of the German Social Accident Insurance – DGUV Test – comprises 16 testing and certification bodies (www.dguv.de/dguv-test). Most of these bodies test and certify machinery, which therefore includes safety-related application programs. Owing to the programs' complexity, most tests must be coordinated with the test bodies during the design and development phases. This also applies to the form taken by specification, documentation and validation, for example in accordance with the IFA matrix method, which can reduce the testing and certification overhead.

6 Development of safety-related application software

This chapter forms the core of the present report and is drawn essentially from DGUV research project FF-FP0319 (see Section 2.2 of the report). The IFA matrix method is introduced below and illustrated by examples.

6.1 Matrix-based specification and documentation

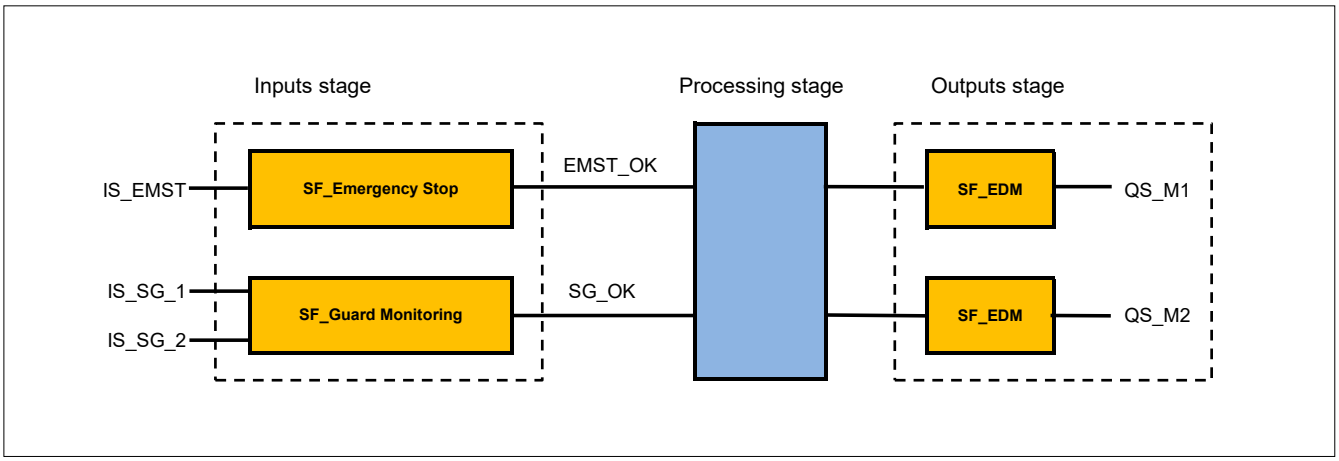
The IFA matrix method relies on a compact specification and documentation of SRASW being created, in addition to the usual documents such as the design documents for a machine and the program printout of the SRASW.

This method requires the software to possess a certain structure. This structure is also an essential requirement for the simplification provided by the V-model (see Section 5.4). Figure 10 shows an example of the software structure.

As the figure shows, the structure is divided clearly into three parts. On the left-hand side is the inputs stage, in which the inputs to the PLC are processed primarily by means of library blocks supplied with the PLC by the manufacturers. The outputs stage is shown on the right-hand side: here, the outputs from the PLC are actuated through library blocks, or directly.

In the middle (shown in blue) is the actuator processing stage, which has yet to be specified. In this example, the outputs EMST_OK and SG_OK of the library blocks used in the inputs stage are connected logically in the processing stage. This logic comprises solely the operations AND, OR and NOT. Time conditions must be implemented at the inputs and outputs stages.

Figure 10:
Example of a specified software structure. The function blocks are compliant with PLCopen [8].



This structure corresponds to the sensor ⇢ logic ⇢ actuator architecture for safety functions from DIN EN ISO 13849-1 (see Section 5.7).

Structured naming of variables is also important for the IFA matrix method. This provides a better overview of the program and supports the avoidance or swift location of errors, as modules associated with each other always share the same name elements.

The matrix method essentially consists of the following steps:

1. Definition of the safety functions (document A1)
2. Listing of the variable names and addresses of safety-related inputs and outputs in the I/O list (document A2.4)
3. Selection of measures for fault avoidance from the catalogue (document A3)
4. Specification of the normative requirements (document A4)
5. Specification of the safety program architecture (document B1)
6. Specification of the modular architecture with modules at the inputs and outputs stages (document B3)
7. Creation of the matrix with the entries for the software specification (document B4)
8. Verification of documents B1, B3 and B4 against the specification of the safety functions (document V1)
9. Programming of the software
10. Code review (document C1), performed if possible by a second person

11. Validation of the software, i.e. analysis and functional test (document D1), if possible by a second person

12. Archiving of the software and documentation, and of all required test documents

Note: The matrix method presented here can also be used for compact presentation of switching operations that are not safety-related.

With reference to successive examples each based upon the last, the following chapters describe application of the IFA matrix method. The examples serve as a common thread. The method can be adapted to the relevant requirements during its application. The contents of some documents (e.g. A3) depend strongly upon the project. Application of the method is supported by the SOFTEMA tool, which is described in Chapter 14.

Care was taken in presentation of the control system examples to make them non-proprietary.

6.2 Example of matrix-based specification and documentation

For compact and clear documentation of the examples discussed here, all documents associated with each example have been compiled in an Excel file. This enables the reader to navigate easily to each document by clicking on the spreadsheet tabs. These Excel documents can be found in a single archive file in the download area of the present IFA report.

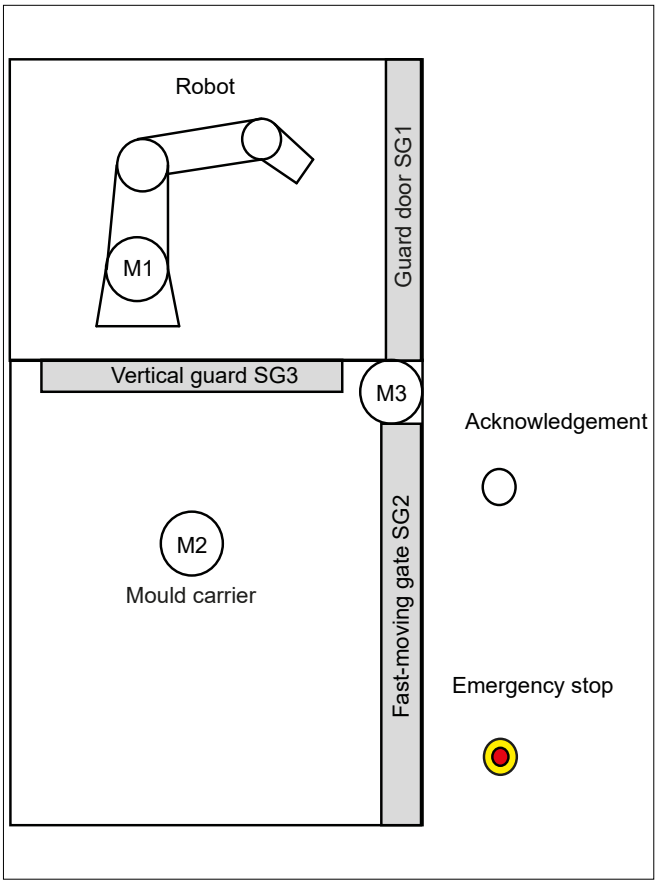
The complete structure of the Excel file is explained with reference to a straightforward example. The format of the examples available for download (Chapter 7) was updated from that used in the research project. This format is better suited to automated processing by a software tool such as SOFTEMA.

In this section, the matrix method is illustrated by the example of a robot production cell, as described in Section 7.1 and shown in Figure 11. The figure constitutes the system sketch (document A2.1). This document is marked as optional in Table 1 (Page 23); however, the example can be explained more easily with this system sketch.

The function of the manufacturing cell can be described as follows:

- The robot (M1) loads material into a mould on the mould carrier. For this purpose, it reaches through the automatic vertical guard SG3.
- After loading the material, the robot withdraws and the vertical guard closes again automatically.
- Following a curing time, a machine operator opens the fast-moving gate SG2 (motor M3), retrieves the finished part from the mould, cleans the mould and closes the fast-moving gate again.

Figure 11:
System sketch (document A2.1), example of a robot production cell



- As soon as the fast-moving gate has been closed, the robot can load material into the mould again.
- The fast-moving gate SG2 has an edge protection sensor SL_SG2 (not shown) to prevent crushing at the closing edge.
- The guard door SG1 provides access to the robot for maintenance.
- The emergency stop EMST and acknowledgement actuator are the only control devices shown in the illustration, as the remaining controlgear is not relevant to safety.
- Automatic mode is the only operating mode for this production cell.

6.3 Specification of the safety functions

The risk assessment yields five safety functions for this case. Table 6 shows the detailed definition of an example safety function (the emergency stop) in the safety specification. The structure of the definition is based on SISTEMA Cookbook 6 [14] and the information in Box 6.1 of BGIA Report 2/2008 [2].

The software engineering of all safety functions in the example is illustrated in compact form in Table 7 (document A1).

Table 6:
Definition of the emergency stop safety function

	SF1	Emergency stop
1	Description	When the emergency stop button EMST is pressed, the drives M1, M2, M3 are halted.
2	Triggering event	Pressing the emergency stop button
3	Safety-oriented response	Stopping: speed = 0, stop category 0 (STO)
4	Dangerous machine part	Axes M1, M2, M3
5	Response to a fault in the SF	Stopping: speed = 0, stop category 0 (STO)
6	PL_r	d
7	Operating mode	All operating modes
8	Parameters/faults	8.1 Parameter: discrepancy time T_{dis} = 50 ms for the contacts of the emergency stop button -> fault
		8.2 Parameter: discrepancy time $T_{dis\_}Q$ = 1 s for contactor monitoring
9	Run-on	100 ms
10	Behaviour in the event of a power failure	Stopping: speed = 0, stop category 0
11	Re-start criteria	No fault, emergency stop button closed, acknowledgement button actuated
12	Priority	1 (highest priority)
Version		
Date		
Name		

Table 7:
Specification of the safety functions (document A1)

No	Description	PL_r	Response time in ms	Priority
SF1	When the emergency stop EMST is actuated, M1, M2 and M3 are switched off.	d	100	1
SF2	When the guard door SG1 is opened, M1 is switched off.	d	100	2
SF3	When the fast-moving gate SG2 is opened, M2 is switched off.	d	100	2
SF4	When the fast-moving gate SG2 is opened and the vertical guard SG3 is opened, M1 is switched off.	d	100	2
SF5	When the edge protection sensor SL_SG2 of the fast-moving gate SG2 is actuated, M3 switched off.	d	100	2

Certain information has been omitted in this compact form. The triggering event and the response of the safety functions are shown in the Description column. The remaining parameters in Table 6 have been included in the more general A3 catalogue of measures for fault avoidance. This improves the clarity of this document. The operating mode is not stated in this example, as automatic mode is the sole operating mode. The safety functions, including multiple operating modes, are shown in the later example which includes set-up mode (Section 6.11).

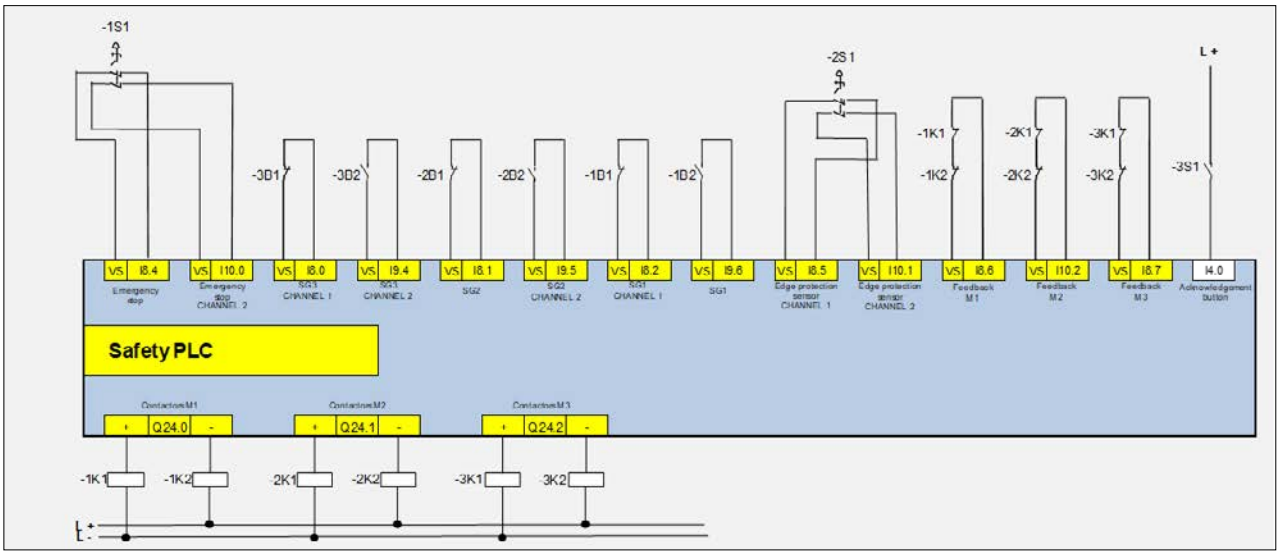
6.4 Specification of the control hardware

Figure 12 shows the circuit diagram (document A2.2) for the example. The diagram is shown here in highly simplified form; normally, circuit diagrams from an electrical CAE program are available for this document.

The circuit diagram shows that all safety functions are implemented in two-channel form owing to the required performance level of PL_r d. The drives are also switched by two contactors connected in series. Each of these contactors is connected to only a single binary output; the desired two-channel implementation appears to be lost as a result. However, each binary output is switched internally within the control system by two switches (not shown); the solution shown is therefore permissible.

Figure 13 shows the system structure. Specifically, this shows the hardware structure of the safety PLC and where applicable the connection of further safety-related components.

Figure 12:
Circuit diagram of the example (Document A2.2)



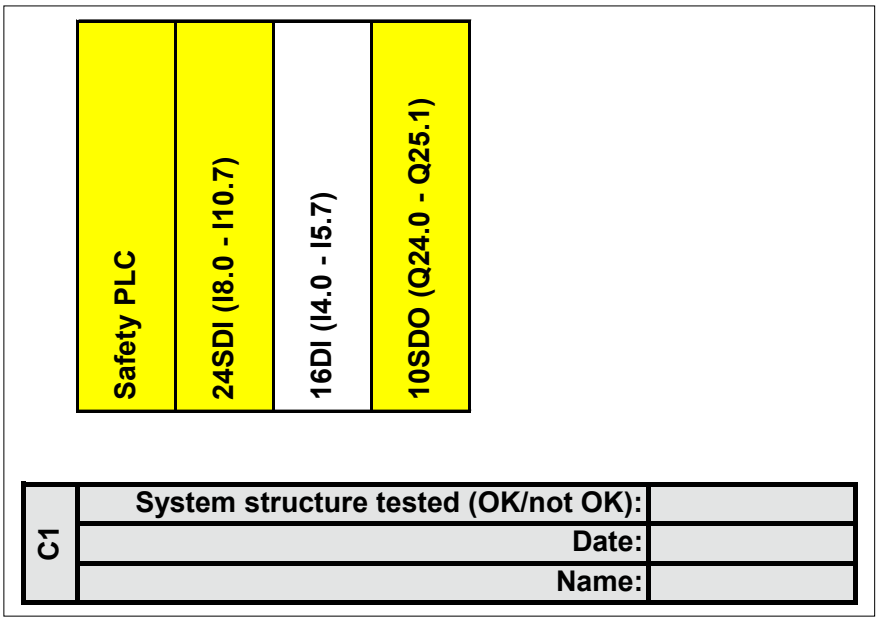


Figure 13:
System structure (document A2.3)

The safety PLC and safety input and output cards are highlighted in yellow. The standard input card is not highlighted. A check field for the code review (C1) is highlighted in grey.

Table 8 shows the I/O list (document A2.4). Variables beginning with the syntax “IS_...” or “QS_...” (serving here as an example) originate from the safe I/O area of the safety PLC. For the exchange of data over networks, communication parameters can also be entered here in place of the I/O addresses.

The I/O list contains additional check fields (grey area): these are part of the validation protocol D1 and the code review C1, in which the wiring of the sensors and actuators (column D1) and the logical relationship between the variables and the function blocks in the software (column C1) are checked for correctness.

Validation in columns D1 is performed on the machine in assembled form, verification in column C1 with reference to the program listing or in the program editor following programming.

For two-channel inputs that are tested for discrepancy by an I/O card, only correct wiring of the inputs need be tested.

Correct parameterization on the I/O card for the specified discrepancy time
(e.g. 50 ms) must be checked. This is one of the manufacturer-specific tests.

Table 8:
I/O list (document A2.4); NC = normally closed contact, NO = normally open contact

Signals	Variable	Address	D1 Validated (OK/not OK)	C1 Correct connections verified in the software (OK/not OK)
Inputs				
Emergency stop EMST, two-channel (NC) (1S1)	IS_EMST	%I8.4		
Contact 1 robot guard door SG1 (NC) (1B1)	IS_SG1_1	%I8.2		
Contact 2 robot guard door SG1 (NO) (1B2)	IS_SG1_2	%I9.6		
Contact 1 fast-moving gate SG2 (NC) (2B1)	IS_SG2_1	%I8.1		
Contact 2 fast-moving gate SG2 (NO) (2B2)	IS_SG2_2	%I9.5		
Contact 1 vertical guard SG3 (NC) (3B1)	IS_SG3_1	%I8.0		
Contact 2 vertical guard SG3 (NO) (3B2)	IS_SG3_2	%I9.4		
Edge protection sensor of SG2, two-channel (NC) (2S1)	IS_SL_SG2	%I8.5		
Feedback contactors M1 (NC) (1K1, 1K2)	IS_SM1	%I8.6		
Feedback contactors M2 (NC) (2K1, 2K2)	IS_SM2	%I10.2		
Feedback contactors M3 (NC) (3K1, 3K2)	IS_SM3	%I8.7		
Acknowledgement button (NO) (3S1)	I_ACK	%I4.0		
Outputs				
Contactor motor M1 (1K1, 1K2)	QS_M1	%Q24.0		
Contactor motor M2 (2K1, 2K2)	QS_M2	%Q 24.1		
Contactor motor M3 (3K1, 3K2)	QS_M3	%Q 24.2		
	Date:			
	Name:			
	Software signature:			

Remarks on the circuit diagram and the I/O list:

- The emergency stop button (IS_EMST) and edge protection sensor (IS_SL_SG2), which are two-channel elements, possess only one address, as the two channels are already monitored on the I/O card for discrepancies. The card then relays a single signal with logical compression (see above). The advantage here is that the program need no longer monitor the discrepancy.

- Discrepancy monitoring on the I/O card is not necessary for the guard door contacts, as the software module SF_GuardMonitoring performing processing requires the two items of contact information separately.

- The mirror contacts of a pair of contactors are connected in series and each read into a binary input (IS_SM1, 2, 3). This enables the contactor to be monitored for failure by means of the SF_EDM function block.

- If the contactors are in fact contactor relays serving to switch the control voltage to the contactors proper for the motor, the mirror contacts of all contactors and contactor relays involved must be connected in series and read in to permit monitoring.

6.5 Catalogue of measures for fault avoidance

Table 9 shows by way of example the catalogue of higher-level measures for fault avoidance; Table 10 shows the measures for fault avoidance specific to the control system used (both compiled in document A3). Further information on the measures can be found in Section 5.7.

The measures shown here serve solely as examples. This catalogue must be suitably adapted and extended, according to the company's requirements and those of the project, for programming of a specific application and for the control system used.

The measures specified for fault avoidance are numbered consecutively (Rx) in document A3 (as shown in Tables 9 and 10) and can be marked as implemented ("OK") or not implemented ("not OK") in column C1 during code review.

Table 9:
Example catalogue of higher-level measures for fault avoidance (document A3)

	C1	
	Abbreviation	(OK/not OK)
Variables		
Prefix for binary inputs: I_... (not safety-related) / IS_... (safety-related)	R1	
Prefix for binary outputs: Q_... (not safety-related) / QS_... (safety-related)	R2	
Prefix for instances: Timers: “T_”; rising edge detection: “R_”; flip-flops: “FF_”; SF_Guardmonitoring: “MON_”; SF_EmergencyStop: “EMST_”; SF_EDM: “EDM_”	R3	
Prefix for global variables: “G_” (not safety-related)/ “GS_” (safety-related).	R4	
Variable names: The variable name after the prefix should be self-explanatory, e.g. using the name of the equipment (...SG1... for SG1).	R5	
Variable declaration: Initialize with a safe state. Comment required.	R6	
Comments: Each network and each variable declaration is commented.	R7	
Signal processing		
Software architecture: The software is to be divided into a inputs stage, processing stageand a outputs stage. The inputs stage is to be implemented in sequential networks. Each binary output is to be implemented in a network together with the processing stageand the outputs stage.	R8	
Assignment: Variables are to be assigned only at a single point.	R9	
Cyclical processing: Each part of the software is processed cyclically without conditions.	R10	
Monitoring of two-channel pusbuttons: Two-channel pusbuttons are monitored on the input card with a discrepancy time of 100 ms.	R11	
Monitoring of contactors: Contactors are monitored with a discrepancy time of 1 s.	R12	
Monitoring of guard doors: Guard door contacts are monitored with a monitoring time of 1 s.	R13	
Automatic re-start: Permitted only for automatic guards.	R14	
Peripheral error: Acknowledgement is required.	R15	
Activation of safety functions: Safety functions are activated by a FALSE signal.	R16	
Library modules used		
Use: Preference should be given to the use of library modules.	R17	
Guard doors: SF_GuardMonitoring	R18	
Emergency stop: SF_EmergencyStop	R19	
Contactors: SF_EDM	R20	
Automatic re-start: The parameters “S_StartReset” and “S_AutoReset” of the library blocks are FALSE. S_AutoReset = TRUE may be set only for automatic guards.	R21	
Activation: The “Activate” input parameter of SF_GuardMonitoring, SF_EmergencyStop and SF_EDM is always TRUE.	R22	
Modules developed in-house: Where possible, logic that is used multiple times should be encapsulated in a module in the form of a function or function block. Development follows the V-model. Password protection and library management are required.	R23	
Activities following changes		
Documentation: All changes must be documented in the change history.	R24	
Validation: Following changes to software, its validation must be repeated.	R25	
	Date:	
	Name:	

Table 10:
Example catalogue of control system-specific measures for fault avoidance (document A3)

		C1	
		Abbreviation	(OK/not OK)
Program editor/programming language			
Program editor used	Safety Editor V10.1	R26	
Programming language	Function Block Diagram (FBD)	R27	
Software library	Safety Library V3.2	R28	
Signal processing			
The safe I/O cards must be operated with ACK_XYZ, i.e. the I/O cards are not automatically reintegrated after a fault has been rectified.		R29	
I/O cards that have been passivated must be acknowledged when they are re-integrated.		R30	
Safety mode (not test mode) must be activated in the PLC.		R31	
If test mode is active, this should be displayed and the machine should switch off automatically.		R32	
		Date:	
		Name:	

6.6 Architecture of the safety program and the standard program

Figure 14 shows the architecture of the safety program (document B1). The call-up hierarchy may differ from that shown here, depending on the control system. However, it can be seen that the main safety program (FB_Main) calls up the safety library blocks. The diagram also includes a brief explanation of the significance of the individual modules and a check box for the code review (C1).

The architecture of the standard program (document B2) is optional, as it is not usually relevant for execution of the safety program. It may, however, facilitate an understanding of the overall function of the installation.

The architecture of the safety program (document B1) can be omitted for simple applications.

Figure 15 shows the modular architecture (document B3). This shows the interaction within the entire safety software. As the illustration shows, the structure is divided clearly into three parts. The inputs stage, in which the inputs are processed primarily by means of library blocks, is shown on the left-hand side.

The outputs stage, in which outputs are controlled directly or by means of library blocks, is shown on the right-hand side. The structure of these two parts is determined by the periphery. The actuation logic for the actuators, which has yet to be specified, is shown in the centre in blue (ACT module). Below this is a field for verification (V1) of the modular architecture. The modular architecture is verified against the specification of the safety functions (document A1). It can also be presented in tables in the form of a module list, rather than graphically as shown in Figure 15. This reduces the overhead for description. The SOFTEMA tool (Chapter 14) uses the list form.

The essential functionality of the certified function blocks in accordance with PLCopen [8] shown in Figure 15, i.e. SF_EmergencyStop (emergency stop function block), SF_GuardMonitoring (guard door function block) and SF_EDM (contactor monitoring function block), is explained in Section 6.17. Only where function blocks are important for the ensuing process are their signals shown. The outputs of the inputs stage employ negative logic (low-active, active low).

The part of Figure 15 that is still unknown at this point is the ACT actuation module for the actuators. In the general case, ACT consists only of the elementary basic logic operators AND (&), OR (≥1) and NOT (o). The symbols for the logical operators used here correspond to IEC 60617-12. The basic structure of the ACT module for actuating an actuator is shown in Figure 16.

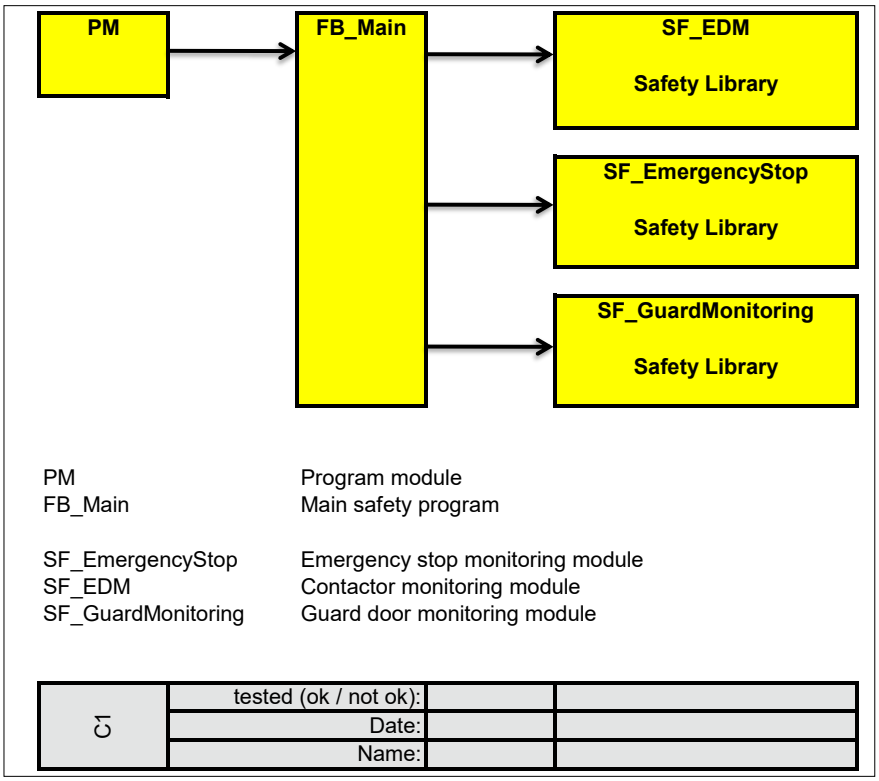
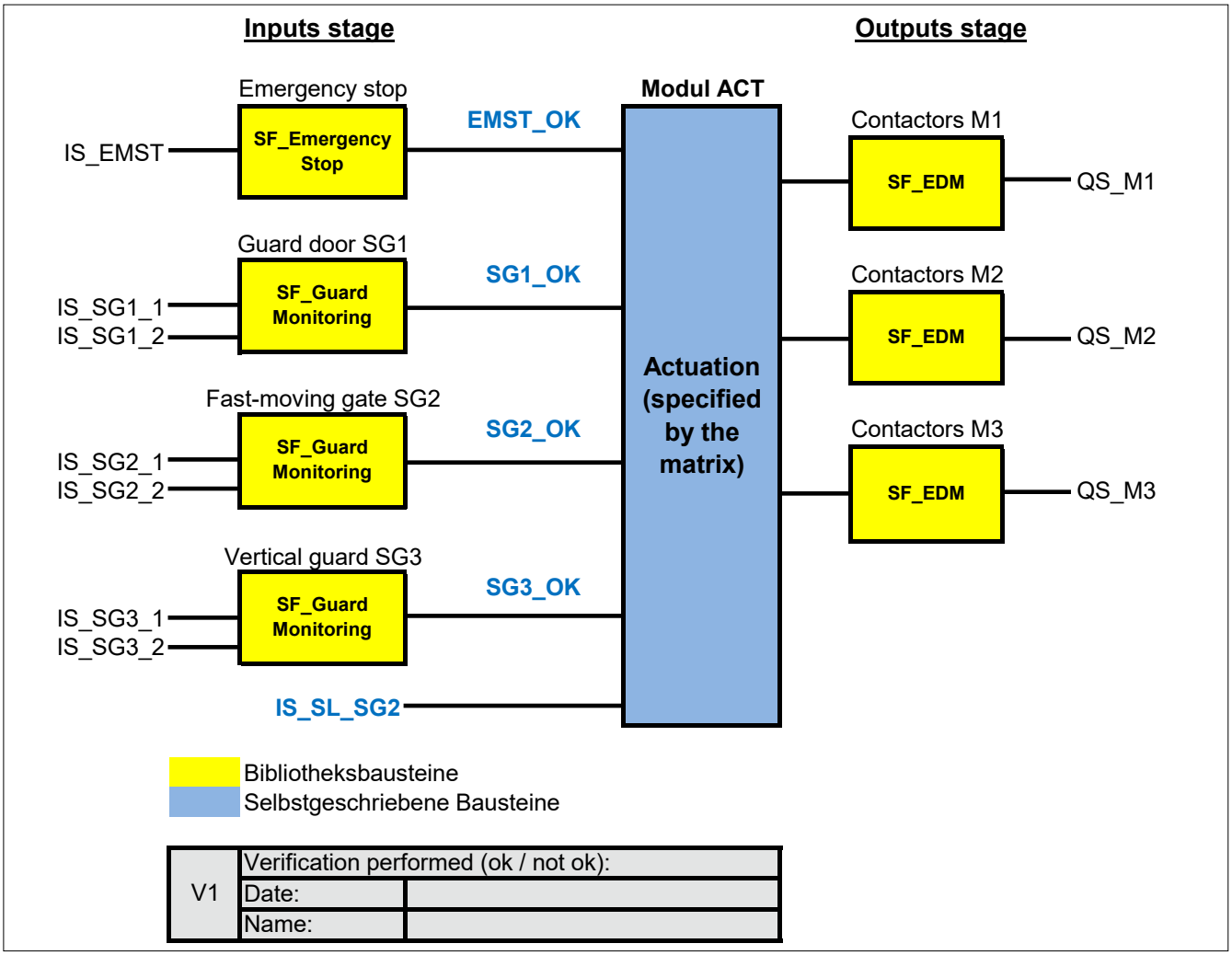


Figure 14:
Architecture of the safety program
(document B1), shown here as a call-up hierarchy

Figure 15:
Modular architecture (document B3)



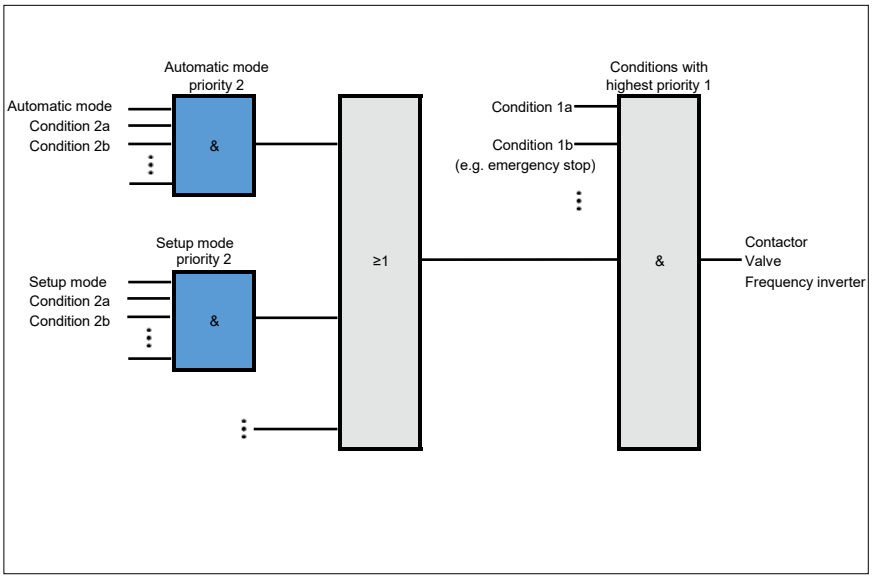


Figure 16:
Structure of the ACT module for an actuator

Safety functions are generally activated with the 0 signal in accordance with the closed-circuit current principle, i.e. the hazard is eliminated. If the output of the right-hand AND element is FALSE, a contactor is de-energized, a valve moves to its safety position or an integral safety function in the variable frequency drive is activated. Figure 16 also shows the significance of the priorities (1 is the highest priority here). Priorities are usually linked to operating modes. Signals with a priority of 1 always activate the safety function with a 0 signal. The emergency stop is a good example. Signals that are assigned to automatic mode or set-up mode, for example, all have priority 2.

Should, in rare cases, priority 3 signals be required, they would be implemented as an input on one of the blue priority 2 AND elements with the same OR-AND logic.

The above example, however, has only a single operating mode. The right-hand AND element of each actuator thus takes effect here, and priorities 1 and 2 are the same.

6.7 Software specification with the cause and effect matrix

The logic described above for the ACT actuation module must be specified in the next step for programming. This specification is also to be suitable for use for verification and validation of the program. A cause and effect (C&E) matrix is created for this purpose. The C&E matrix is a graphical presentation of causes leading to or significantly influencing effects. The C&E matrix is also known as a cause and effect diagram, cause and effect chart or cause and effect table. These forms of presentation are described in IEC 62881. Figure 17 shows an example of the C&E matrix for the installation described here.

Figure 17:
Cause and effect (C&E) matrix

Cause									Effect		
Involved inputs								Safety functions	Outputs		
IS_EMST (I8.4)	IS_SG1_1 (I8.2)	IS_SG1_2 (I9.6)	IS_SG2_1 (I8.1)	IS_SG2_2 (I9.5)	IS_SG3_1 (I8.0)	IS_SG3_2 (I9.4)	IS_SL_SG2 (I8.5)		QS_M1 (Q24.0)	QS_M2 (Q24.1)	QS_M3 (Q24.2)
1	1	1	1	1	1	1	1	ALL_OK	ON	ON	ON
0	-	-	-	-	-	-	-	SF1: Emergency stop actuated	OFF	OFF	OFF
-	0	0	-	-	-	-	-	SF2: SG1 open	OFF	NOP	NOP
-	-	-	0	0	-	-	-	SF3: SG2 open	NOP	OFF	NOP
-	-	-	0	0	0	0	-	SF4: SG2 and SG3 open	OFF	NOP	NOP
-	-	-	0	0	-	-	0	SF5: Edge protection sensor actuated	NOP	NOP	OFF

This is the subject of the discussions below. In a C&E matrix, the triggering events (causes, i.e. the control system's signal inputs) are listed below each other in rows on the left-hand side and their effects (the signal outputs) in columns on the right-hand side. The example installation has an initial state from which all safety functions are tested. This state is referred to as "ALL_OK" in the matrix. In the ALL_OK state, all enabling signals for the actuators QS_M1 ... QS_M3 are TRUE. Beginning in this initial state, the switching operations of the safety functions are entered with the associated input signals.

This matrix shows the switching behaviour of the safety functions clearly, and therefore serves as the basis for testing during software validation. It is created during the specification phase, independently of subsequent programming.

As the matrix clearly shows the switching behaviour of the safety functions, it is advantageous to extend it to cover specification of the software for the safety functions. The simple presentation shown in Figure 17 is extended for this purpose. Fields for code review and validation are added, together with entries that clearly specify the ACT actuation module. This yields document B4, the safety-related software specification and validation plan. Figure 18 shows the complete matrix.

Figure 18:
Extended C&E matrix (document B4, Safety-related software specification and validation plan); check fields (grey fields) and control information (blue variables)

			Cause									Effect				D1		
			Involved inputs								Safety functions	Outputs						
Mode of operation	Preceding state in the test	State	IS_EMST (I8.4)	IS_SG1_1 (I8.2)	IS_SG1_2 (I9.6)	IS_SG2_1 (I8.1)	IS_SG2_2 (I9.5)	IS_SG3_1 (I8.0)	IS_SG3_2 (I9.4)	IS_SL_SG2 (I8.5)		QS_M1 (Q24.0)	QS_M2 (Q24.1)	QS_M3 (Q24.2)	Acknowledgement I_ACK (I4.0)	Tested (ok / not ok)	Name	Date
											C1: Software consistent with the matrix documentation							
		1	1	1	1	1	1	1	1	1	ALL_OK	EIN	EIN	EIN				
	1	2	0	1	1	1	1	1	1	1	SF1: Emergency stop actuated	EMST_OK OFF	EMST_OK OFF	EMST_OK OFF	ON			
	1	3	1	0	0	1	1	1	1	1	SF2: SG1 open	SG1_OK OFF	NOP	NOP	ON			
	1	4	1	1	1	0	0	1	1	1	SF3: SG2 open	NOP	SG2_OK OFF	NOP	ON			
	1	5	1	1	1	0	0	0	0	1	SF4: SG2 and SG3 open	SG2_OK v SG3_OK OFF	NOP	NOP	ON			
	1	6	1	1	1	0	0	1	1	0	SF5: Edge protection sensor actuated	NOP	NOP	IS_SL_SG2 OFF	ON			
V1	Verification performed (OK / not OK):													Software signature:				
	Date:																	
	Tester:																	

In the additional columns on the left-hand side, the operating mode, a serial number (state) and the preceding state during testing of the safety function are stated for each safety function. "Preceding state" refers to the state set for the test before a demand is made the safety function under test.

The "software signature" shown at the bottom of Figure 18 is a unique identifier for a version of the SRASW. It changes whenever the program is modified, even if at only one point.

The "Verification" field (V1) on the left at the bottom of Figure 18 serves to verify the switching information (OFF, ON, NOP) and the variables (in blue) against the specification of the safety functions (document A1).

Acknowledgement indicates both the acknowledgement input and the need for acknowledgement. If, for example, the safety function SF1 has been tripped, the ALL_OK state is not restored until the EMERGENCY STOP button has been disengaged and then acknowledged with "ON" (corresponding to TRUE on the acknowledgement input).

The variables relevant to the software specification are shown in blue in Figure 18. The variables are the output variables from the inputs stage and are thus the input variables of the ACT actuation module (Figure 15). These variables employ negative logic.

The following rule applies to generation of the software specification:

Step 1: For each individual safety function triggering a switching operation on an actuator (from the preceding state, in this case: ALL_OK), the logical combination of the ACT input variables triggering the switching operation is entered in the relevant cell of the table. This switching process is indicated here by "OFF" or "ON". Where a safety function does not trigger a switching operation on an actuator, "NOP" must be entered in the cell.

Example from Figure 18 for the safety function SF1 and its effect upon the output QS_M3:

The cell contains “EMST_OK“, and below that “OFF”. This entry means: “If variable EMST_OK = FALSE, output QS_M3 should = FALSE.”

NOP does not trigger a switching operation.

Example for the safety function SF4 and output QS_M1:

SF4 triggers a switching operation when – and only when – guard door s SG2 and SG3 are open. The OR operation “SG2 v SG3” must therefore be entered. This means: “If the expression (SG2 _OK OR SG3_OK) = FALSE, output QS_M1 should = FALSE.”

Step 2: The complete logic combination for each actuator (i.e. across all safety functions) is then derived from the AND operation of the variables shown in blue in the column for the actuator.

Figure 19 shows the final result for ACT in this example.

Where a corresponding option is offered in the programming environment of a safety PLC, this code could be generated automatically from the entries shown in blue in Figure 18. This would also enable programming of the ACT module to be completed in the same work step by specification of the ACT module in the PLC’s programming environment.

The following applies to application of the matrix:

- Programmers read the relevant column (the entries in blue) for each actuator in the C&E matrix in order to program the logic for it.
- Testers read the rows of the matrix to test individual safety functions.

The C&E matrix makes the entire switching behaviour of the safety functions transparent and verifiable. The entries in blue specify the software of the ACT module formally and unambiguously. This also makes the software clear, comprehensible and verifiable at the code review stage.

The complete safety Program sketch (document B5) for this example is presented in Figure 20. This outline serves to make the example comprehensible. It is generally of no relevance during practical application of the matrix method, as the program is written directly against specification B4 rather than with reference to the outline.

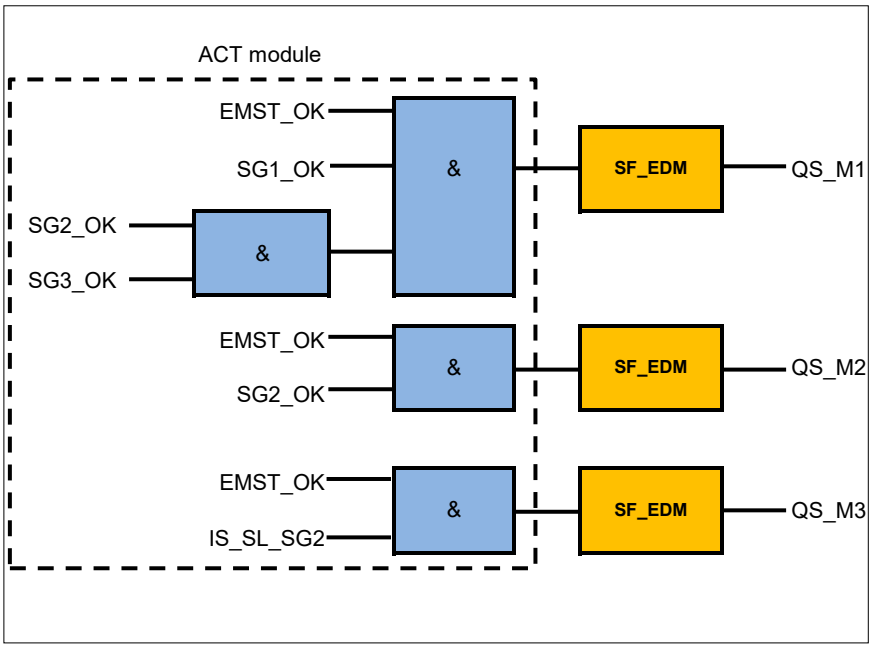
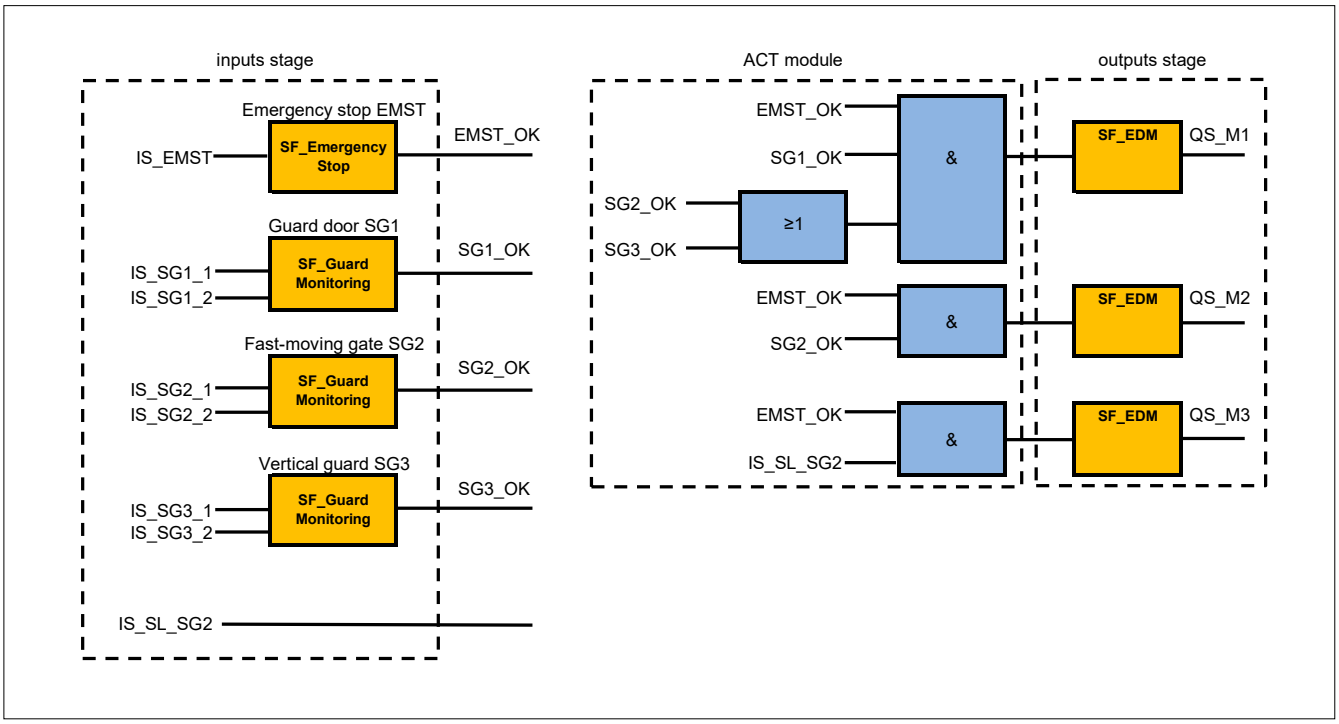


Figure 19:
Logical structure of the ACT module (in the dashed frame)

Figure 20:
Program sketch (document B5)



6.8 Verification and validation in the IFA matrix method

The C&E matrix shown in Figure 18 also contains the following fields:

- V1 Verification:
  Is the switching behaviour entered consistent with the safety functions specified in A1?

- C1 Code review:
  Is the program code of the ACT actuation module consistent with the switching behaviour entered? (Proceed column by column for each output.)

- D1 Software validation:
  Does the application software meet the safety requirements? (Proceed row by row for each safety function.) All safety functions are tested live in the installation. The associated states and addresses of the inputs and outputs are also listed. The initial state from which the test is to be performed is stated. Note that further rows for testing with further test cases can easily be added to the matrix. For PL_r c, d and e, a preliminary simulation test is recommended in accordance with the measures for fault avoidance set out in the standards.

Table 11 shows an example of a code review (document C1). The “Reference” column of the code review refers to relevant documents. Each of these documents contains a C1 field, column or row for the code review.

Table 11:
shows an example of a code review (document C1).

Have the activities been completed?	Reference:	Completed (yes/no)
1. Were the agreed fault avoidance measures, tools and programming rules observed during programming?	A3 Measures (column C1)	
2. Has the hardware system structure been implemented?	A2.3 System structure (field C1)	
3. Has the interconnection of I/O signals been implemented correctly in the code?	A2.4 I/O list (column C1)	
4. Is the architecture of the safety program being followed?	B1 Architecture of the safety program (field C1)	
5. Has the modular architecture been respected?	B3 Modular architecture (field C1)	
6. Has the specification from the matrix been implemented in the code?	B4 C&E matrix (row C1)	
Date:		
Name:		
Software signature:		

Table 12:
Example Protocol of software validation (document D1)

Have the activities been completed?	Reference:	Completed (yes/no)
Verification of the modular architecture	B3 Modular architecture (field C1)	
Verification of the matrix	B4 C&E matrix (field V1)	
Code review	C1 Code review:	
I/O-check	A2.4 I/O list (column D1)	
Testing of the peripheral devices	Refer to the manufacturer's specifications	
Validation of the C&E matrix	B4 C&E matrix (column D1)	
Validation of the safety functions	A1 Safety functions (column D1)	
Validation of the requirements	A4 Requirements (column D1)	

Is documentation complete?	Present (yes/no)
V-model documents from this Excel document	
PDF printout of all safety-related software, including checksum	
PDF printout of the hardware configuration (with all settings), including checksum	
Archiving of PDF manuals for all system components	
PDF printout of the configuration of peripheral devices, including checksums	
Manufacturer's approval regulations (e.g. for parameterization of safety components)	
Relevant standards	
Date:	
Name:	
Software signature:	

The Protocol of software validation (document D1, see Table 12 for an example) cross-references the fields, columns or rows marked V1 or D1 in the reference documents.

For the user, this raises the question of the test coverage of the D1 columns in Figure 18. Testing is limited here to testing of the serviceability of the safety functions (5 in number) (as already stated, rows containing additional test cases can be added if necessary).

As can be seen from Figure 15, the ACT module in this case has five binary input variables (i.e. the enable signals) and three binary output variables (actuator signals). The switching behaviour of the ACT could therefore be described by a switching table with 32 rows (2^5 input variables) and three columns. The logic in Figure 19 could also be obtained by the use of mathematical transformations. Working through this switching matrix would yield full test coverage for the ACT module. In practice, the overhead would just about be justifiable in this case.

The situation is different if, for example, ten safety functions are present, resulting in the ACT having approximately ten input variables. Ten safety functions or more are quite conceivable in real-case applications. The switching matrix would then consist of 1,024 rows and corresponding columns for the actuators. The overhead of specification and testing entailed by use of the switching matrix would not be justifiable in this case. Completion of the switching table alone would probably result in a few errors creeping in. If a matrix similar to that in Figure 18 were used, ten rows would be required. Completion of these, including the entries with blue highlighting, would involve little effort. The authors advocate a pragmatic approach to software validation in practice: an approach which, although it fails to provide full test coverage, can be carried out by and is comprehensible to a wide group of persons. This is possible, for example, by means of the D1 validation columns proposed in Table 8 and Figure 18. This also provides full transparency of what has been checked, when, and by whom. Information on appropriate test coverage can be found in Section 5.11.

Software validation (document D1) comprises the following parts:

- Analysis: of verification (documents B3, B4), code review (C1), normative requirements (A4), safety functions (A1)
- Real-case functional test: of the I/O list and the C&E matrix (documents A2.4, B4)

Review activities should preferably be performed in observance of the two-man rule (Section 5.15).

In addition to the activities described here for specification and review of the application software, the manufacturer-specific parameterizations of the system components (safety PLCs, I/O cards, frequency inverters, sensors, etc.) must also be specified and tested during software validation. For example, correct safety-related parameterization of frequency inverters must be checked and documented (see documents in Table 12).

Validation, and also external audit of the software, require all necessary documents to be created and retained.

Specifically, the documentation should contain:

- Documents of the V-model from the Excel file
- PDF printout of all safety-related software, including checksum
- PDF printout of the hardware configuration (with all settings), including checksum
- Archiving of the manuals for all system components, as changes may occur over time, resulting in the installed status no longer lending itself to documentation at a later date
- PDF printout of the manufacturer-specific configuration of peripheral devices, including checksums
- Manufacturer's approval regulations (e.g. for the parameters of a safety PLC or a safety variable frequency drive); these are usually found in the system manuals
- Specifications to be complied with from Type C standards for special machines (e.g. presses)
- Further relevant standards and documents

For documentation purposes, PDF is proposed here as a data format owing to its ubiquity, as external auditors do not necessarily have access to all system software applications.

6.9 Compact software specification

The presentation of the C&E matrix in Figure 18 may not be suitable for greater numbers of actuators (such as several hundred actuators in rolling mills) and sensors. For this reason, alternative, compact presentations are described here. For better management of larger numbers of actuators in a table, transposition to the presentation shown in Figure 18 is recommended. This transposition is performed automatically when SOFTEMA (Chapter 14) is used. However, the larger the number of actuators, the greater the number of relevant input variables. Transposing the C&E matrix is therefore likely to make presenting the input variables difficult. For this reason, this presentation shows the information on the input variables involved only in very rudimentary form. The assumption here is that an experienced tester knows specifically how to handle these input variables during software validation. They must be aware that a guard door , for example, is to be opened during the test. They must know that an emergency stop button, for example, is to be pressed during the test. Figure 21 shows the transposed and reduced presentation of the C&E matrix in this example.

Figure 21 is clearly much more compact than Figure 18. It also contains less detailed information; this, however, is not necessarily a drawback in industrial practice. The entries in blue font fully specify the processing stage of the actuators on the left. The input variables involved are summarized below.

In this new presentation, however, the direct relationship between the fields of the matrix describing the de-energizations and the safety functions defined in Table 7 is lost, as the latter are formulated with respect to the sensors. The "SFs involved" column is intended to provide an overview and establish the relationship to the safety functions concerned.

An even more compact presentation of Figure 21 involves omission of the entries for the input signals (Figure 22). This is conditional upon experienced testers being able to determine from the variable names which sensors/input signals are to be actuated.

Figure 21:
Transposed and reduced presentation of the C&E matrix (document B4)

Actuators		Switch offs		C1			D1		
			SFs involved	Software consistent with the matrix documentation			Function tested		
Output	Description	Automatic mode of operation		OK/not OK	Name	Date	OK/not OK	Name	Date
QS_M1	Motor M1	**EMST_OK&SG1_OK & (SG2_OK v SG3_OK)** EMST, IS_SG1,2,3_1,2	1, 2, 4						
QS_M2	Motor M2	**EMST_OK&SG2_OK** EMST, IS_SG2_1,2	1, 3						
QS_M3	Motor M3	**EMST_OK&IS_SL_SG2** EMST, IS_SL_SG2	1, 5						
V1	Verification performed (OK / not OK):			Software signature:					
	Date:								
	Tester:								

Figure 22:
Further compressed matrix (document B4)

Actuators		Switch offs		C1			D1		
			SFs involved	Software consistent with the matrix documentation			Function tested		
Output	Description	Automatic mode of operation		OK/not OK	Name	Date	OK/not OK	Name	Date
QS_M1	Motor M1	EMST_OK&SG1_OK & (SG2_OK v SG3_OK)	1, 2, 4						
QS_M2	Motor M2	EMST_OK&SG2_OK	1, 3						
QS_M3	Motor M3	EMST_OK&IS_SL_SG2	1, 5						
V1	Verification performed (OK / not OK):			Software signature:					
	Date:								
	Tester:								

The processing stage is described in full with the output signals of the outputs stage. If more complex signals are formed or several signals grouped in the inputs stage, it may be difficult to list all the inputs involved. In this case, the very compact presentation as shown in Figure 22 is recommended.

6.10 Notes on the inputs stage

Note here that the inputs stage cannot always be presented as simply as in Figure 20. In more complex cases, the project planning task consists of combining signals in the inputs stage logically such that the output variables of the logic can be used directly in the ACT module to drive actuators. The specified negative logic must be observed for inputs modules for safeguards:

- Output variable = TRUE/1: the safeguard is not tripped; hazards are present, but safeguarded
- Output variable = FALSE/0: the safeguard is tripped, hazards must be de-energized

An example of such signal grouping is shown in Figure 23. In this example, monitoring of the stationary state (safe operating stop) of several axes is grouped in a single stationary state signal. The example of the „Machine tool“ (Section 7.5) shows a practical application for this grouping.

Figure 24 shows an example of a time-controlled door release being enabled in the inputs stage. The demand for the door to be unlocked (pushbutton I_OP_SG) sets a flip-flop. Once the parameterizable time TIME has elapsed, the unlocking signal S_UNLOCK is output. Complete opening of the door resets the F_TON timer. It would be useful to encapsulate the logic in Figure 24 in a reusable library module.

The transition time until the door is opened cannot be read in the matrix. During the functional test, this information must be read from the definition of the safety function.

Transferring timing aspects and the combination of complex signals to the inputs or outputs stages usually enables the ACT processing logic to be limited to purely binary logic (AND, OR, NOT).

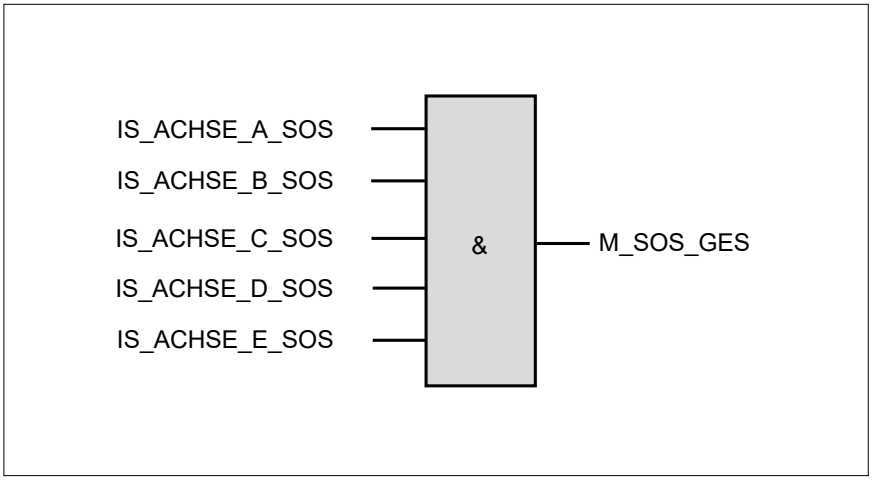


Figure 23:
Summary of stationary-state signals in the inputs stage

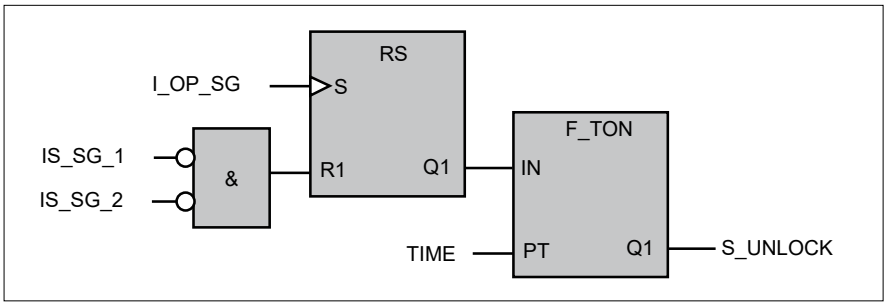


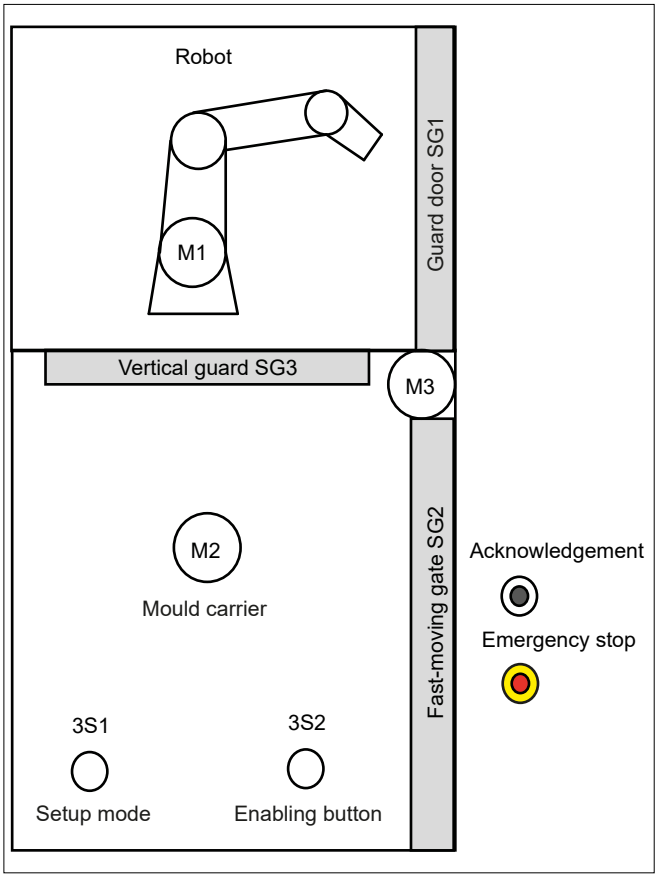
Figure 24:
Example of a timer-controlled door release in the inputs stage

6.11 Consideration of multiple operating modes and function blocks developed in-house

The above example will now be extended with the addition of a setup mode (as in the example in Section 7.2). This example clearly shows the difference between the operating modes. In addition, the example implements use of a function block developed in-house for driving a motor with safely limited speed (SLS). Development of such a function block is described in Section 6.12.

Figure 25 shows the system sketch (document A2.1) of the robot cell with integrated setup mode for axis M2.

Figure 25:
System sketch (document A2.1) of a robot cell with setup mode



In addition to the function from the previous example, the mould carrier (M2) can be moved in setup mode at a safely limited speed with the fast-moving gate open and the vertical guard closed, by means of two enabling buttons arranged at different points. As soon as the speed of the motor becomes too high, it switches off safely.

The risk assessment thus yields seven safety functions. The first five are unchanged from the previous example. Table 13 shows the safety functions of the robot cell with setup mode.

Figure 26 shows the circuit diagram for this example. The only change with respect to the previous example is the addition of the two push buttons 3S1 and 3S2. The motor M2 is now controlled by a frequency inverter with integral safety functions.

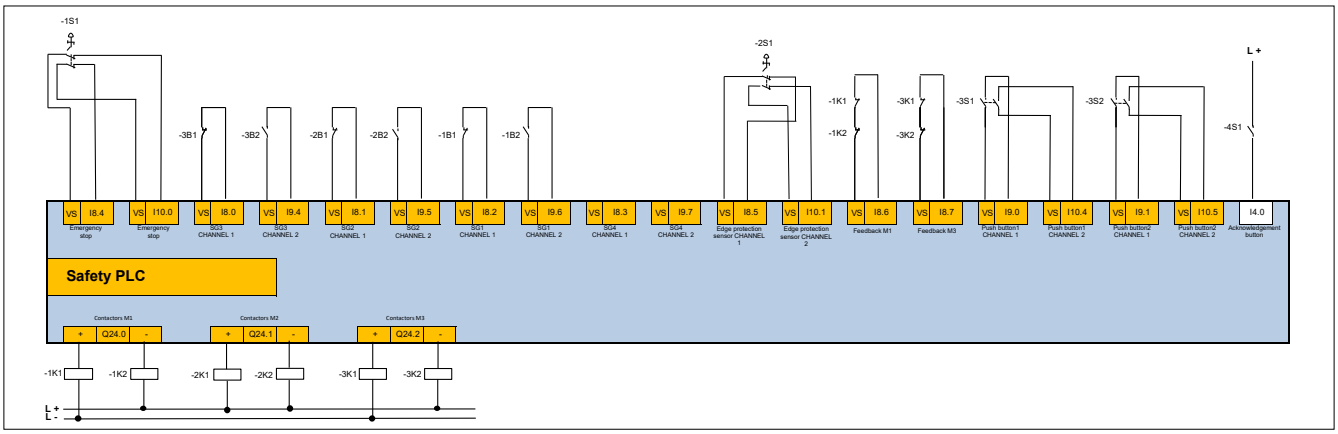
Use of a drive with integral safety functions in this example has resulted in significant changes to the system structure (document A2.3). A drive with field bus interface has now been added. Figure 27 shows the system structure.

Table 14 (Page 50) shows the symbol table for this example. Changes with respect to the previous example arise from the two push buttons at the inputs and actuation of the frequency inverter safety functions at the outputs.

Table 13:
Safety functions (document A1) of the robot cell with setup mode

No	Description	PL_r	Response time in ms	Priority	Operating mode
SF1	When the emergency stop EMST is actuated, M1, M2 and M3 are switched off.	d	100	1	All
SF2	When the guard door SG1 is opened, M1 is switched off.	d	100	1	All
SF3	When the fast-moving gate SG2 is opened, M2 is switched off.	d	100	2	Automatic mode
SF4	When the fast-moving gate SG2 is opened and the vertical guard SG3 is opened, M1 is switched off.	d	100	1	All
SF5	When the edge protection sensor SL_SG2 of the high-speed guard SG2 is actuated, the motor M3 is switched off.	d	100	1	All
SF6	When the fast-moving gate SG2 is open, the vertical guard SG3 is closed and the push button 3S1 is depressed, SLS is enabled for M2 (SLS active).	d	100	2	Setup mode
SF7	When the high-speed guard SG2 is open, the vertical guard SG3 is closed and the push button 3S2 is depressed, SLS is enabled for M2 (SLS active).	d	100	2	Setup mode

Figure 26:
Circuit diagram (document A2.2) of the robot cell with setup mode



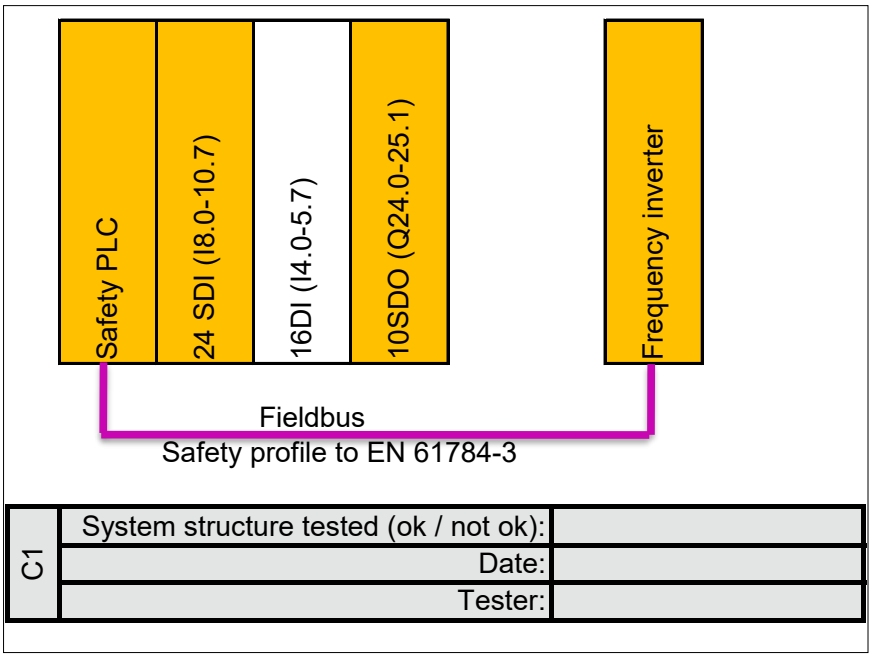


Figure 27:
System structure (document A2.3) of the robot cell with setup mode

Table 14:
Symbol table (document A2.4) of the robot cell with setup mode

Signals	Variable	Address	D1	C1
			Validated (OK/not OK)	Correct connections verified in the software (OK/not OK)
Inputs				
Emergency stop, two-channel (NC) (1S1, 1S2, 1S3)	IS_EMST	%I8.4		
Contact 1 guard door SG1 (NC) (1B1)	IS_SG1_1	%I8.2		
Contact 2 guard door SG1 (NO) (1B2)	IS_SG1_2	%I9.6		
Contact 1 fast-moving gate SG2 (NC) (2B1)	IS_SG2_1	%I8.1		
Contact 2 fast-moving gate SG2 (NO) (2B2)	IS_SG2_2	%I9.5		
Contact 1 vertical guard SG3 (NC) (3B1)	IS_SG3_1	%I8.0		
Contact 2 vertical guard SG3 (NO) (3B2)	IS_SG3_2	%I9.4		
Edge protection sensor of SG2, two-channel (NC) (2S1)	IS_SL_SG2	%I8.5		
Feedback contactors M1 (NC) (1K1, 1K2)	IS_SM1	%I8.6		
Feedback contactors M3 (NC) (3K1, 3K2)	IS_SM3	%I8.7		
Push button 1 SLS, two-channel (NO) (3S1)	IS_TIP_1	%I9.0		
Push button 2 SLS, two-channel (NO) (3S2)	IS_TIP_2	%I9.1		
Safety error message FI	IS_Err_FU	%I32.7		
Acknowledgement button (NO) (4S1)	I_ACK	%I4.0		
Outputs				
Contactor motor M1 (1K1, 1K2)	QS_M1	%Q24.0		
Contactor motor M3 (3K1, 3K2)	QS_M3	%Q24.2		
Activation of STO for M2	QS_M2_STO	%Q32.0		
Activation of SLS for M2	QS_M2_SLS	%Q32.4		
Activation of SS1 for M2	QS_M2_SS1	%Q32.1		
Activation of SS2 for M2	QS_M2_SS2	%Q32.2		
Activation of SOS for M2	QS_M2_SOS	%Q32.3		
Acknowledgement of FI safety error	QS_M2_ACK_FI	%Q32.7		
		Date:		
		Name:		
		Software signature:		

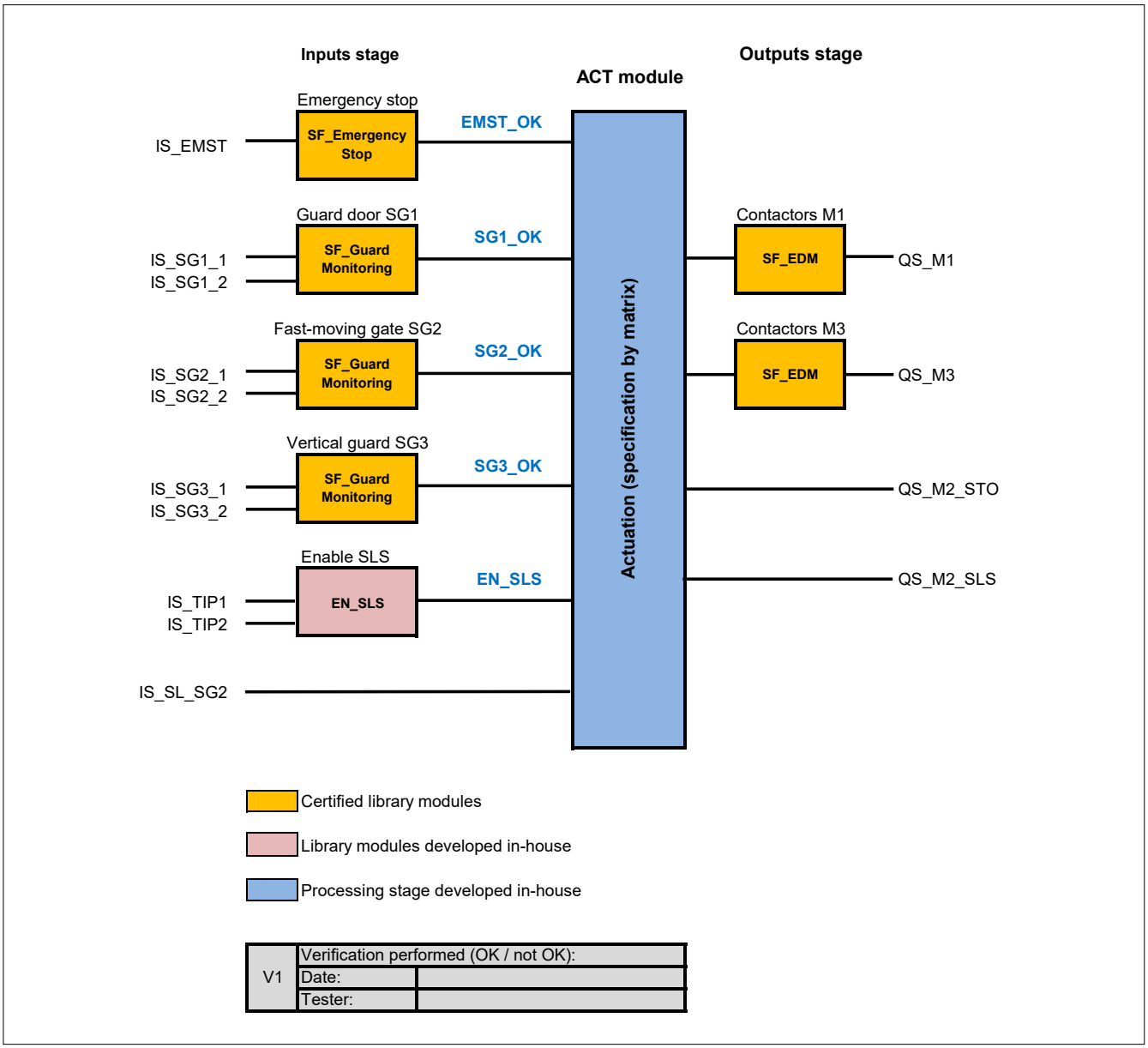
The catalogue of measures for fault avoidance (document A3), the normative requirements (A4), the architecture of the safety program (B1), the architecture of the standard program (B2) and the code review (C1) are unchanged with respect to the previous example.

Figure 28 shows the modular architecture (document B3) of the example. Of the examples shown here, this is the first containing a function block (EN_SLS) which is to be developed by the programmer themselves.

The outputs for the safe frequency inverter for the motor M2 can be seen in the modular architecture. These commands are explained briefly below (see DIN EN 61800-5-2):

- STO (safe torque off): When the STO command is activated, supply of power to the drive is switched off safely and the drive coasts to a halt.
- SLS (safely limited speed): The SLS command is used to monitor a specified maximum speed. Monitoring is independent of the direction of the motor's rotation. If this speed is exceeded, STO is automatically activated in the frequency inverter.
- SS1 (safe stop 1): The SS1 command brings about a category 1 stop in accordance with DIN EN 60204-1. The drive is braked to a halt within a parameterizable time and the power supply then switched off with STO.
- SS2 (safe stop 2): The SS2 command also causes the drive to be braked to a halt within a parameterizable time. In this case, however, the safe standstill of the drive is then monitored with SOS.
- SOS (safe operating stop): The SOS command is used for safe monitoring of the drive's stationary state. Should the motor still turn, the power supply to the drive is switched off with STO.
- All safety functions are activated with a FALSE/0 signal (negative logic): should communication with the control system fail, all safety functions are active and the drive stops. For this reason, all safety functions that are not in use (SS1, SS2, SOS) are deactivated with the TRUE signal.
- The output QS_M2_ACK_FI is used to acknowledge faults in the frequency inverter.

Figure 28:
Modular architecture (document B3) of the robot cell with setup mode



The function block EN_SLS must be specified. The EN_SLS signal activates the QS_M2_SLS output with the FALSE signal. Since the safety functions of the frequency inverter are all activated with FALSE, it is also advantageous to define the output of the function block in negative logic.

Figure 29 shows the C&E matrix (document B4) of the robot cell with setup mode. In the interests of clarity, the states of the input signals are not shown. The first five safety functions are the same as before. An initial state is defined for setup mode (state 7, highlighted in white). From this state, the motor M2 can be moved at a safely limited speed. The matrix also contains a test row with a test case (state 7).

Any number of rows containing test cases can be added to the C&E matrix for further tests (highlighted in yellow in this example).

This matrix is the first to show negations of the variables, which are indicated by a preceding slash.

Attention is drawn again to the completion rules for the ACT actuation module:

Step 1: For each individual safety function triggering a switching operation on an actuator (from the previous state), the logical combination of the ACT input variables triggering the switching operation is entered in the relevant cell of the table according to the operating mode. This switching process is indicated here by “OFF” or “ON”. Where a safety function does not trigger a switching operation on an actuator, “NOP” must be entered in the cell.

Figure 29:
C&E matrix (document B4) of the robot cell with setup mode

			Cause	Effect						
				Outputs					D1	
Mode of operation	Preceding state in the test	State	Description	QS_M1 (Q24.0)	QS_M2_STO (Q32.0)	QS_M2_SLS (Q32.4)	QS_M3 (Q24.2)	Acknowledgement (I_ACK (I4.0)	Tested (OK/not OK)	Name	Date
			C1: Software consistent with the matrix notation								
		1	**ALL_OK**	ON	ON	ON	ON				
	1	2	SF1: Emergency stop actuated	**EMST_OK** OFF	**EMST_OK** OFF	NOP	**EMST_OK** OFF	ON			
	1	3	SF2: SG1 open	**SG1_OK** OFF	NOP	NOP	NOP	ON			
	1	4	SF3: SG2 open	NOP	**SG2_OK& EN_SLS** OFF	NOP	NOP	ON			
	1	5	SF4: SG2 and SG3 open	**SG2_OK v SG3_OK** OFF	NOP	NOP	NOP	ON			
	1	6	SF5: Edge protection sensor actuated	NOP	NOP	NOP	**SL_SG2_OK** OFF	ON			
		7	SG2 open, SG3 closed, IS_TIP1, 2 not actuated	NOP	OFF	ON	NOP				
	7	8	SF6: SG2 open, SG3 closed, IS_TIP_1 actuated	NOP	**/EN_SLS** ON	**EN_SLS** OFF	NOP	ON			
	7	9	SF7: SG2 open, SG3 closed, IS_TIP_2 actuated	NOP	**/EN_SLS** ON	**EN_SLS** OFF	NOP	ON			
	7	10	SG2 open, SG3 closed, IS_TIP_1, 2 actuated	NOP	OFF	ON	NOP				
V1	Verfication performed (OK/not OK):					Software signature:					
	Date:										
	Name:										

Automatic mode
Setup mode
All

Several examples of the safety functions SF3, SF4 and SF5 from Figure 29 are described below.

- When SF3 is activated in the automatic operating mode by the opening of SG2, M2 is switched off by STO from the previous All_OK state, unless safely limited speed (SLS) has been activated for M2. SG2_OK & EN_SLS must therefore be entered in this cell.

- When SF4 is activated in the “all” operating mode by the opening of SG2 and SG3, M1 is switched off. SG2_OK v SG3_OK must therefore be entered in this cell. NOP must be entered for QS_M2_STO, as M2 is not affected by this safety function in this operating mode.

- If SF6 is activated in setup mode by push button IS_TIP_1 being pressed whilst guard door SG2 is open and guard door SG3 is closed, QS_M2_STO is switched on (ON: i.e. STO deactivated in this case) and QS_M2_SLS is switched off (OFF: i.e. SLS enabled in this case). The negated variable /EN_SLS must therefore be entered for QS_M2_STO and the variable EN_SLS (enabled by EN_SLS = FALSE) for QS_M2_SLS.

Step 2: The inputs of the AND elements in Figure 16 are derived as follows from Figure 29: for each output and each operating mode, only the variables entered must be linked by AND.

Figure 30 shows the program sketch. The structure of the actuation module shown in Figure 16 can be clearly seen in the ACT module at the output QS_M2_STO. On the right-hand AND, (EMST_OK) is the condition for all operating modes. Before it, to the left, is an OR condition with the automatic operating mode (SG2_OK & EN_SLS) and the setup operating mode (NOT (EN_SLS)).

The SLS safety function can also be used in combination with the SS2 safety function. The drive then remains under closed-loop control and is monitored for safe standstill with SOS if no push button is actuated. For the sake of simplicity, this has not been implemented here.

For the sake of completeness, Figure 31 shows the compact presentation of the matrix.

An additional table showing test cases has been created here to permit further useful tests with the compact form of presentation. Figure 32 shows the entries for additional test cases.

Figure 30:
Program sketch of the robot cell with setup mode

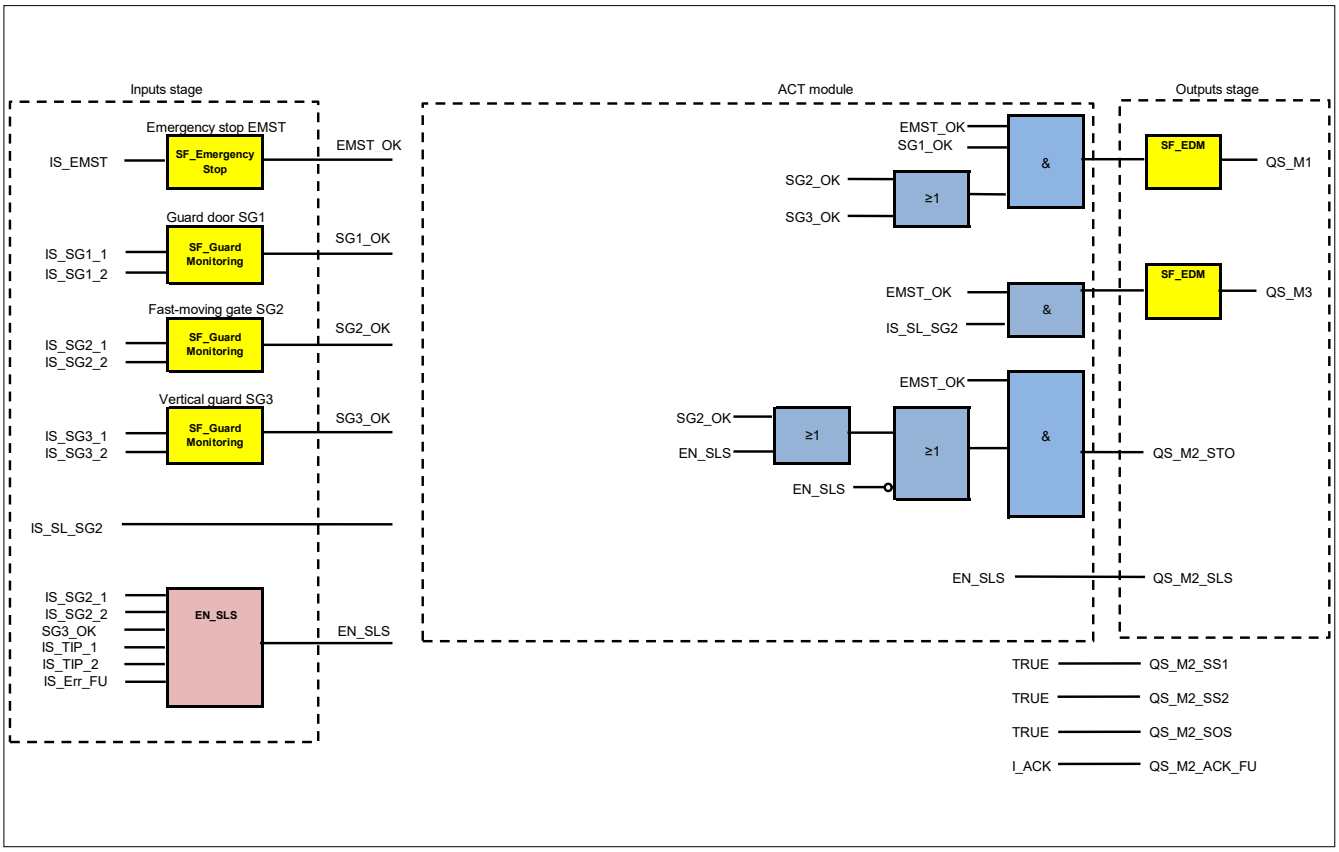


Figure 31:
Compact matrix presentation for the example of the robot cell with setup mode

Actuators		Switch offs			SFs involved	C1 Software consistent with the matrix documentation			D1 Function validated		
Output	Description	Mode of operation, all	Automatic mode of operation	Setup mode of operation		OK/not OK	Name	Date	OK/not OK	Name	Date
QS_M1	Motor M1	EMST_OK & SG1_OK & (SG2_O v SG3_OK)			1, 2, 4						
QS_M2_STO	Motor M2 STO	EMST_OK	SG2_OK & EN_SLS	/EN_SLS	1, 3, 6, 7						
QS_M2_SLS	Motor M2 SLS			EN_SLS	6, 7						
QS_M3	Motor M3	EMST_OK & IS_SL_SG2			1, 5						
V1	Verification performed (OK / not OK):					Software signature:					
	Date:										
	Name:										

Figure 32:
Additional test cases for the example of the robot cell with setup mode

Additional test cases		D1 Validated		
Test case	Reaction	OK/not OK	Name	Date
SG2 open, SG3 closed, both Push buttons IS_TIP_1 and IS_TIP_2 actuated	SLS not enabled			
SG2 open, SG3 closed, both Push buttons IS_TIP_1 and IS_TIP_2 not actuated	SLS not enabled			

6.12 Addressing of configurable safety controls

Besides the use of freely programmable PLCs with a typical PLC language (FBD, LD, e.g. in accordance with DIN EN ISO 61131-3 [12]), safety functions can increasingly also be implemented with compact, graphically configurable control systems. The question then arises as to the applicability of the normative requirements for SRASW to such systems, as DIN EN ISO 13849-1, for example, describes requirements of its own for “software-based parameterization” in Section 4.6.4. This Section, however, was intended by the standards committee to address control devices such as sensors or drive controls whose functions can be parameterized, but not logically linked.

Programming tools with a simple graphical user interface are typical for this type of configurable controller. Function blocks can be placed and linked to each other, i.e. with inputs and outputs, in a graphical editor. The function blocks must then be parameterized: in exactly the same way, in fact, as in a conventional PLC, but with no facility for the use of text-based programming languages. Although these controllers are advertised as being quick and easy to configure, they are still susceptible to logic errors in the same way as their “larger” counterparts.

The programs of such controllers must therefore be regarded as SRASW, and can be specified, validated and documented by means of the IFA matrix method. An example is shown in Section 7.10.

6.13 Matrix-based documentation of function blocks developed in-house

The procedure for use of the IFA matrix method for creating specifications for function blocks developed in-house will now be described. The description is based on the example of the robot cell with setup mode with reference to the EN_SLS function block.

The simplified V-model for development of function blocks (Figure 7, see Section 5.4) is used to create the specification for the EN_SLS function block. The associated documents are listed in Table 2, Section 5.5.

Figure 33 shows the description of the interface (document AM1) for the function block. It contains a description of the function, and all inputs and outputs are stated with symbolic name, data type and comment.

The catalogue of measures for fault avoidance for a function block developed in-house is somewhat more compact than that for the full application program. Tables 15 and 16 show examples of general and controller-specific measures for the development of function blocks (document AM2).

Figure 33:
Description of the interface (document AM1) of the EN_SLS function block

FB name: EN_SLS			
Description of the function			
The EN_SLS function block generates the EN_SLS enable signal for the safely limited speed (SLS). The enable is issued with a logical 0. For the enable to be issued, the Guard door SG1 must be open (0 signal at bSG1_1 and bSG1_2), the Guard door SG2 must be closed and acknowledged (1 signal at bSG2), there must be no error in the frequency inverter (0 signal at bERROR), only one enable push button (bTIP_1 or bTIP_2) is actuated. The positive edges of the two buttons are evaluated to prevent an automatic restart.			
Inputs			
Name	**Data type**	**Initial value**	**Description, parameter values**
bSG1_1	SAFEBOOL	FALSE	Contact 1 of SG1. FALSE: Door open. TRUE: Door closed.
bSG1_2	SAFEBOOL	FALSE	Contact 2 of SG1. FALSE: Door open. TRUE: Door closed.
bSG2	SAFEBOOL	FALSE	Outputs of SF_Guardmonitoring FB for SG2. FALSE: Door open. TRUE: Door closed.
bTIP_1	SAFEBOOL	FALSE	Enable button 1. FALSE: Button not actuated. TRUE: Button actuated.
bTIP_2	SAFEBOOL	FALSE	Enable button 2. FALSE: Button not actuated. TRUE: Button actuated.
bERROR	SAFEBOOL	FALSE	Frequency inverter error. FALSE: No error. TRUE: Error.
Outputs			
bEN_SLS	SAFEBOOL	TRUE	Enable signal for SLS. FALSE: Release granted. TRUE: No release for Enable SLS.
Notes: None			

Table 15:
Example catalogue of general measures for fault avoidance for in-house development of function blocks

	CM1	
	Abbreviation	Implemented (y/n)
A. Variables		
Prefixes for Boolean variables: “b”.	RMA1	
Prefixes for instances: timers: “T_”; rising edge detection: “R_”; flip-flops: “FF_”	RMA2	
Variable names: the variable name following the prefix should be self-explanatory, e.g. containing the name of the relevant functional unit. Example: ..SG1.. for the guard door SG1.	RMA3	
Variable declaration: initialization with the safest value. Each declaration contains a comment.	RMA4	
Interface: each function block communicates with the environment solely through the input/output variables.	RMA5	
Global variables: not permitted.	RMA6	
B. Signal processing		
Assignments: variables are assigned in a single program statement only.	RMB1	
Comments: each network contains one comment.	RMB2	
Protection: the function block is protected by a password.	RMB3	
C. Library modules		
Use: wherever possible, library modules should be used.	RMC1	
Date:		
Name:		
Software signature:		

Table 16:
Example catalogue of controller-specific measures for fault avoidance for in-house development of function blocks

		CM1	
		Abbreviation	Implemented (y/n)
Program editor/programming language			
Program editor used	Safety Editor V10.1	RS1	
Programming language	Function Block Diagram (FBD)	RS2	
Software library	Safety Library V3.2	RS3	
	Date:		
	Name:		
	Software signature:		

Figure 34 shows the module specification with the grey verification and validation fields (document BM1) for the function block EN_SLS. The behaviour of the output is described here again briefly and presented in the form of a C&E matrix. Here too, the C&E matrix serves as a basis for testing and for specification of the software. The model of the function block EN_SLS consists of a “set dominant” flip-flop, which is set/reset by the OR combinations alluded to in Figure 34.

For programming, the logical operations shown by the variables highlighted in colour are linked by OR operations for the set and reset input of the flip-flop.

Figure 35 shows the program sketch (document BM2) of the EN_SLS module, from which the system structure can clearly be seen. Table 17 shows the code review (document CM1) for a function block developed in-house.

The module test protocol (document DM1) is derived from the validation columns on the right in Figure 34 (document BM1).

This method can be used to define and test simple function blocks (i.e. consisting essentially of flip-flops and AND and OR operations). The test is usually performed during simulation. The set and reset logic of EN_SLS is relatively simple and can therefore be described by a matrix notation as shown in Figure 34 (including test plan). Specification by means of a matrix is not possible with more complex function blocks; for such cases, refer to the specification methodology developed by PLCopen [8].

Figure 34:
Module specification and test plan of the function block EN_SLS (document BM1), EQ = EQUIVALENT, / = NOT, › = rising edge

Description														
EN_SLS is realized by a "set dominant" Flip-flop. The Flip-flop is set by an OR connective. The reset is again realized by an OR connective. The inputs of these OR links are specified below by the colored variables listed in the "effect" column. The expected value of the function block's Outputs is also shown in the "effect" column.														
			Cause							Effect				
			Inputs						Description	Output	DM1			
Flip-flop	Preceding state	State	bSG1_1	bSG1_2	bSG2	bTIP_1	bTIP_2	berror		bEN_SLS	Correct (Y/N)	Name	Date	
									C1: Software consistent with matrix specification					
		1	0	0	1	0	0	0	bTIP_1, 2 FALSE, SG1 open, SG2 closed, berror FALSE	ON				
Reset									bTIP_1 TRUE,	>bTIP_1				
	1	2	0	0	1	1	0	0		OFF				
			OR						bTIP_2 TRUE	>bTIP_2				
	1	3	0	0	1	0	1	0		OFF				
Set									bSG1_1 TRUE	bSG1_1				
	2	4	1	0	1	1	0	0		ON				
			OR						bSG1_2 TRUE	bSG1_2				
	2	5	0	1	1	1	0	0		ON				
			OR						bSG2 FALSE	/bSG2				
	2	6	0	0	0	1	0	0		ON				
			OR						berror TRUE	berror				
	2	7	0	0	1	1	0	1		ON				
			OR						bTIP_1 TRUE,FALSE; bTIP_2 TRUE, FALSE	bTIP_1 EQ bTIP_2				
	2	8	0	0	1	0	0	0		ON				
									VM1	Software signature:				
										Verification correct (Y/N):				

Figure 35:
Program sketch (document BM2) of the module EN_SLS

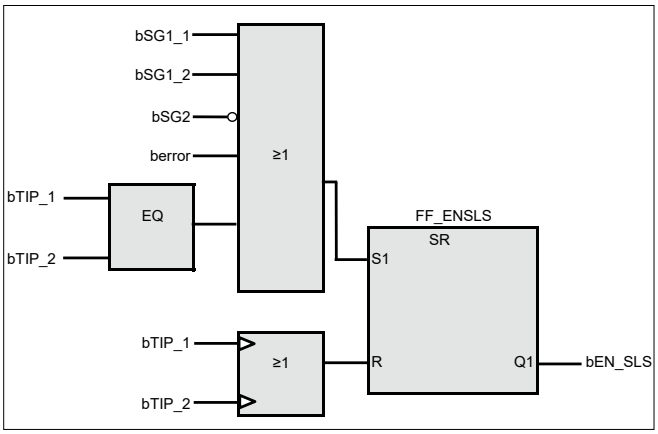


Table 17:
Code review (document CM1) for function blocks developed in-house

Verifications	Reference:	Yes/no
1. Have the agreed programming rules been observed?	Measures AM2	
2. Have the agreed tools been used?	Measures AM2	
3. Is the code consistent with the matrix specification?	3. Is the code consistent with the interface description AM1/module specification BM1?	
Date:		
Name:		
Software signature:		

6.14 Summary of the matrix-based documentation

The essential observations of this chapter are as follows:

- A pragmatic and transparent procedure is stated for working through the further simplified V-model. This serves as a common thread.
- This procedure is independent of the controller, the specified PL_r and the (suitable) programming language employed.
- The key points of the procedure are:
  - Division of the software into inputs, processing stage and outputs stages.
  - Description of the processing logic by means of a C&E matrix (or a transposed and reduced C&E matrix, which is particularly suitable for larger numbers of safety functions).
  - The matrix presentations make the individual operating modes transparent.
  - The matrix presentation can be used to create a specification for the application program, and to verify and validate it comprehensibly.
  - Additional test cases can be added to the matrix to extend the test coverage.
  - In principle, the matrix can be used to generate code and make working with a safety PLC much easier.
- Software validation consists of analyses (verification, code review) and functional tests (I/O list, C&E matrix). In practice, full test coverage is difficult to achieve, and a pragmatic approach must be adopted (Section 5.11).
- Simple function blocks (library modules) developed in-house can be specified and tested by means of similar matrix methods. For more complex function blocks, refer to PLCopen [8].
- The specification and the test plans for further manufacturer-specific details (e.g. for parameters of I/O cards, parameters of frequency inverters, etc.) must also be created and edited.

Figure 36 shows once again the relationships between the documents and activities in the V-model, in this case in summarized form. The figure shows the documents of the V-model and their relationships to the activities to be reviewed. The hardware is reviewed in documents A2.3 and A2.4 and the program in documents A2.4, A3, B1, B3 and B4. “A” stands for the analyses and the “T” for functional test. “C1” or “D1” on the arrows indicates directly whether it concerns a part of the code review or software validation, respectively. As can be seen from the dashed arrow, documents B3 and B4 must be verified against the specification of the safety functions (A1).

Note: The matrix method shown here can also be used for compact presentation of switching operations that are not relevant to safety.

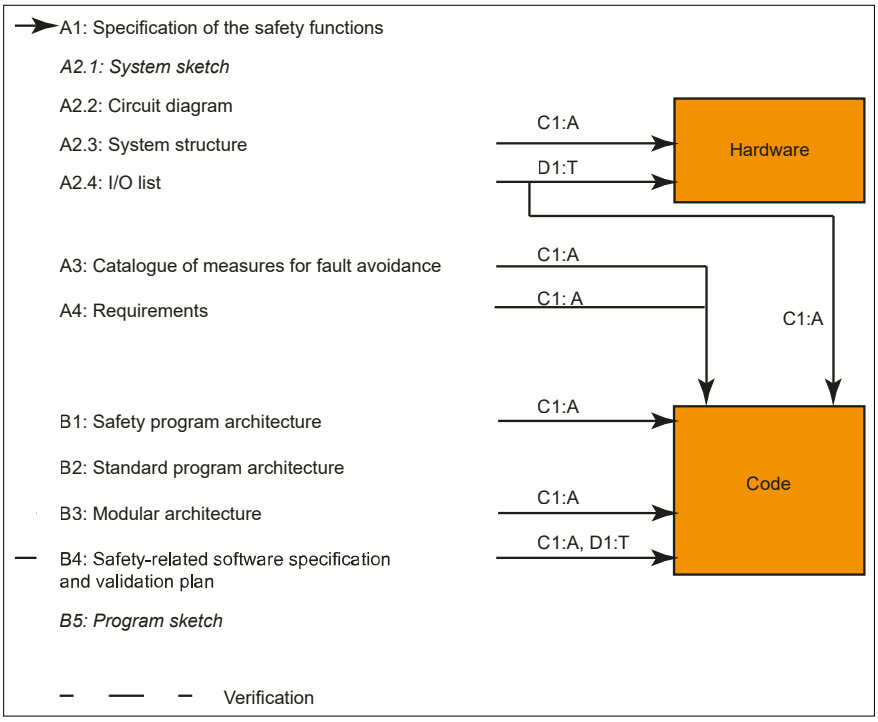


Figure 36:
Relationship between the documents of the V-model (italics = optional); A = analysis, T = test, C1 = Protocol of code review, D1 = Protocol of software validation

6.15 Procedure for modifications

Process software may require adaptation, for example following modifications to a machine's functions or to the hardware. This applies in equal measure to the safety software: safety concepts are modified, additional safeguards are installed, or a safeguard's influence on the machine's work process is altered.

Modifications to software require steps to be retraced in the software development V-model. The more far-reaching the change, the further back in the V-model the process must be recommenced. A method for modifying software is shown here with reference to an example.

Change management comprises the following steps:

1. Impact analysis: entry of the change on the "Change history" form (see Table 18 for an example), with a description in plain text and listing of the affected documents.
2. The changes in the documents stated are marked, for example in colour.
3. If at all possible, the changes should be verified by a second person.
4. Programming of the change.
5. If possible, the code review should be performed by a second person.
6. The changes are validated (I/O-check in the event of new signals; functional test).
7. Following successful validation, the colour markings are removed again.
8. Version archiving of all affected documents.

This procedure will now be demonstrated with reference to the example of the robot cell described in the previous chapter. A modification is made in the form of installation of an additional guard door to facilitate access for the performance of maintenance work (corresponding to the example in Section 7.3). Table 18 shows the entry in the change history, including analysis of which documents are affected. Since addition of a further guard door requires two new safety functions to be integrated, the V-model for the modified part must be followed again in full. The names of the documents correspond to those in Table 1 for the V-model documents.

In the interests of clarity, the modified system sketch (document A2.1) is first shown in Figure 37. The guard door SG4 is a new addition.

The new guard door brings with it two new safety functions, SF6 and SF7. These ensure that the motor M2 behind the guard door is switched off when the guard door is opened, and that the motor M1 is switched off when the guard door s SG4 and SG3 are open.

Table 19 lists the safety functions (document A1) with the new safety functions SF6 and SF7 (in red). The operating mode was not stated, as automatic mode is the only operating mode in this example.

The new guard door SG4 has resulted in the addition of two further inputs to the circuit diagram, for monitoring the guard door . Figure 38 shows the modified circuit diagram with colour-coded contacts (document A2.2).

These two new inputs result in changes to the I/O list (document A2.4). They are added to Table 20, marked in red.

Table 18:
Example of a change history (based on the example of the robot cell)

No	Date	Modification performed by	Document	Modification
1	09.01.2013	M. Muster	A1, A2.1, A2.2, A2.4, B3, B4, B5	Guard door 4 added for performance of maintenance work on the mould carrier
2				
3				
4				
5				

Figure 37:
System sketch (document A2.1) of the robot cell with additional guard door

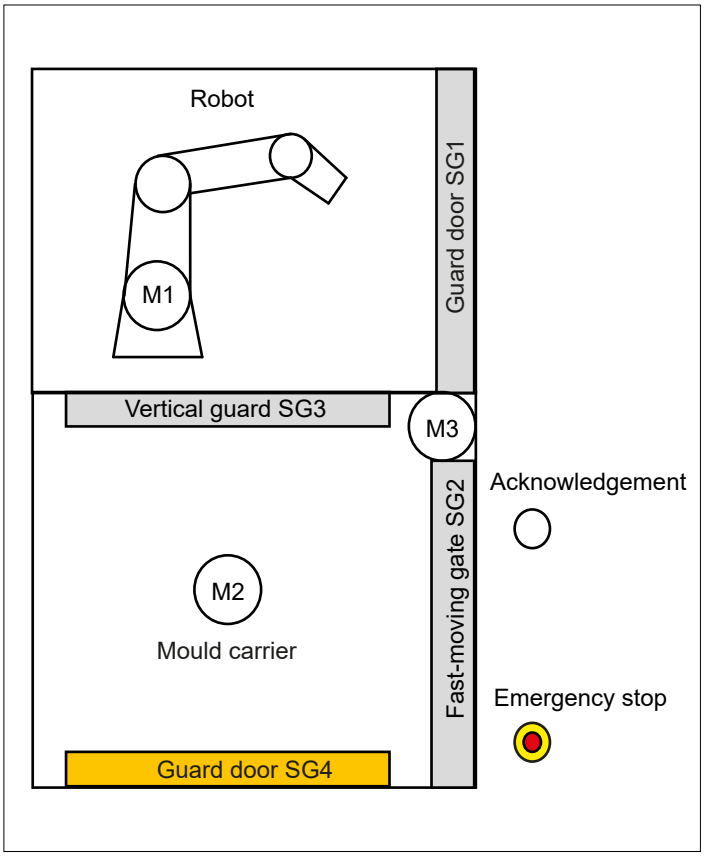


Table 19:
Safety functions (document A1) of the robot cell with additional guard door

No	Description	PL_r	Response time in ms	Priority
SF1	When the emergency stop EMST is actuated, M1, M2 and M3 are switched off.	d	100	1
SF2	When the guard door SG1 is opened, M1 is switched off.	d	100	2
SF3	When the fast-moving gate SG2 is opened, M2 is switched off.	d	100	2
SF4	When the fast-moving gate SG2 is opened and the vertical guard SG3 is opened, M1 is switched off.	d	100	2
SF5	When the edge protection sensor SL_SG2 of the high-speed guard SG2 is actuated, the motor M3 is switched off.	d	100	2
SF6	When the guard door SG4 is opened, M2 is switched off.	**d**	**100**	**2**
SF7	When the guard door SG4 is opened and the vertical guard SG3 is opened, M1 is switched off.	**d**	**100**	**2**

Figure 38:
Circuit diagram (document A2.2) of the robot cell with additional guard door

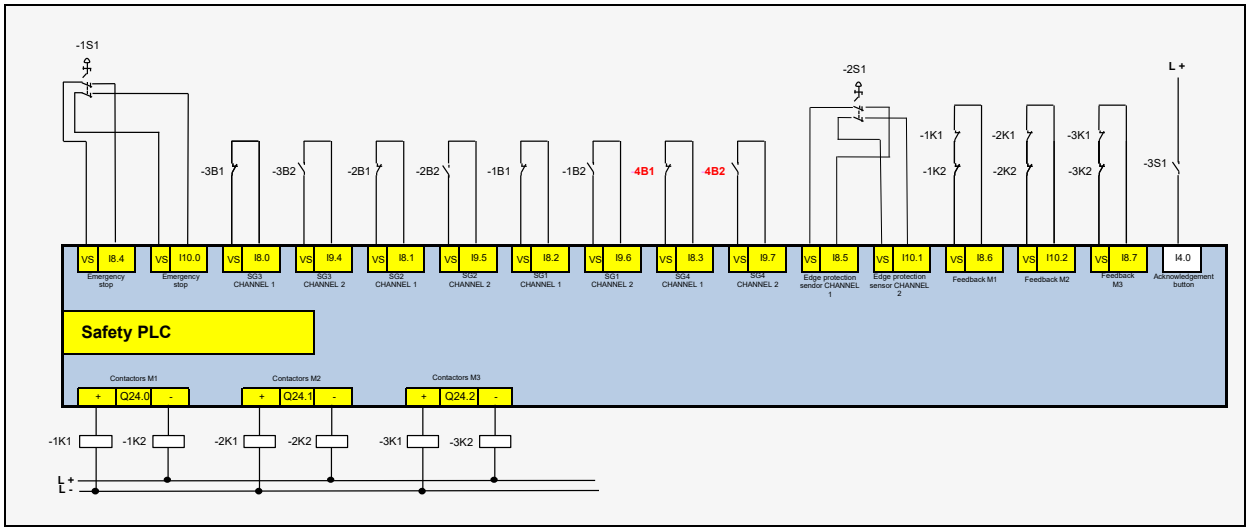


Table 20:
I/O list (document A2.4) of the robot cell with additional guard door

Inputs	Variable	Address	D1 Validated (OK/not OK)	C1 Correct connections verified in the software (OK/not OK)
Inputs				
Emergency stop, two-channel (NC) (1S1)	IS_EMST	%I8.4		
Contact 1 robot guard door SG1 (NC) (1B1)	IS_SG1_1	%I8.2		
Contact 2 robot guard door SG1 (NO) (1B2)	IS_SG1_2	%I9.6		
Contact 1 fast-moving gate SG2 (NC) (2B1)	IS_SG2_1	%I8.1		
Contact 2 fast-moving gate SG2 (NO) (2B2)	IS_SG2_2	%I9.5		
Contact 1 vertical guard SG3 (NC) (3B1)	IS_SG3_1	%I8.0		
Contact 2 vertical guard SG3 (NO) (3B2)	IS_SG3_2	%I9.4		
Contact 1 guard door SG4 (NC) (4B1)	**IS_SG4_1**	**%I8.3**		
Contact 2 guard door SG4 (NO) (4B2)	**IS_SG4_2**	**%I9.7**		
Edge protection sensor of SG2, two-channel (NC) (2S1)	IS_SL_SG2	%I8.5		
Feedback contactors M1 (NC) (1K1, 1K2)	IS_SM1	%I8.6		
Feedback contactors M2 (NC) (2K1, 2K2)	IS_SM2	%I10.2		
Feedback contactors M3 (NC) (3K1, 3K2)	IS_SM3	%I8.7		
Acknowledgement button (NO) (3S1)	I_ACK	%I4.0		
Outputs				
Contactor motor M1 (1K1, 1K2)	QS_M1	%Q24.0		
Contactor motor M2 (2K1, 2K2)	QS_M2	%Q24.1		
Contactor motor M3 (3K1, 3K2)	QS_M3	%Q24.2		
		Date:		
		Name:		
		Software signature:		

The additional guard door module is also listed in the modular architecture (document B3). This expanded modular architecture is shown in Figure 39. The C&E matrix (document B4) also changes: as shown in Figure 40, it has two additional rows for the new safety functions.

In the compact form of the matrix representation (Figure 41), the changes are shown in only two cells.

In the Program sketch in Figure 42 (document B5), the change is shown clearly by the colour highlighting.

The code review document (C1) remains unchanged. However, the code review must be repeated for the changes. The parts of the program that have changed must also be reviewed. The I/O list (document A2.4) for the new signals and the new rows of the C&E matrix (document B4) or modified rows of the compact matrix presentation must also be verified and validated.

This procedure is intended to make changes as transparently comprehensible as possible, thereby preventing errors from arising when software components are modified. If a structured procedure for software development is observed, such as the IFA matrix method described above, the effort entailed by a change can be reduced to that described. If not, the entire program may have to be checked again following a modification.

Figure 39:
Modular architecture (document B3) of the robot cell with additional guard door

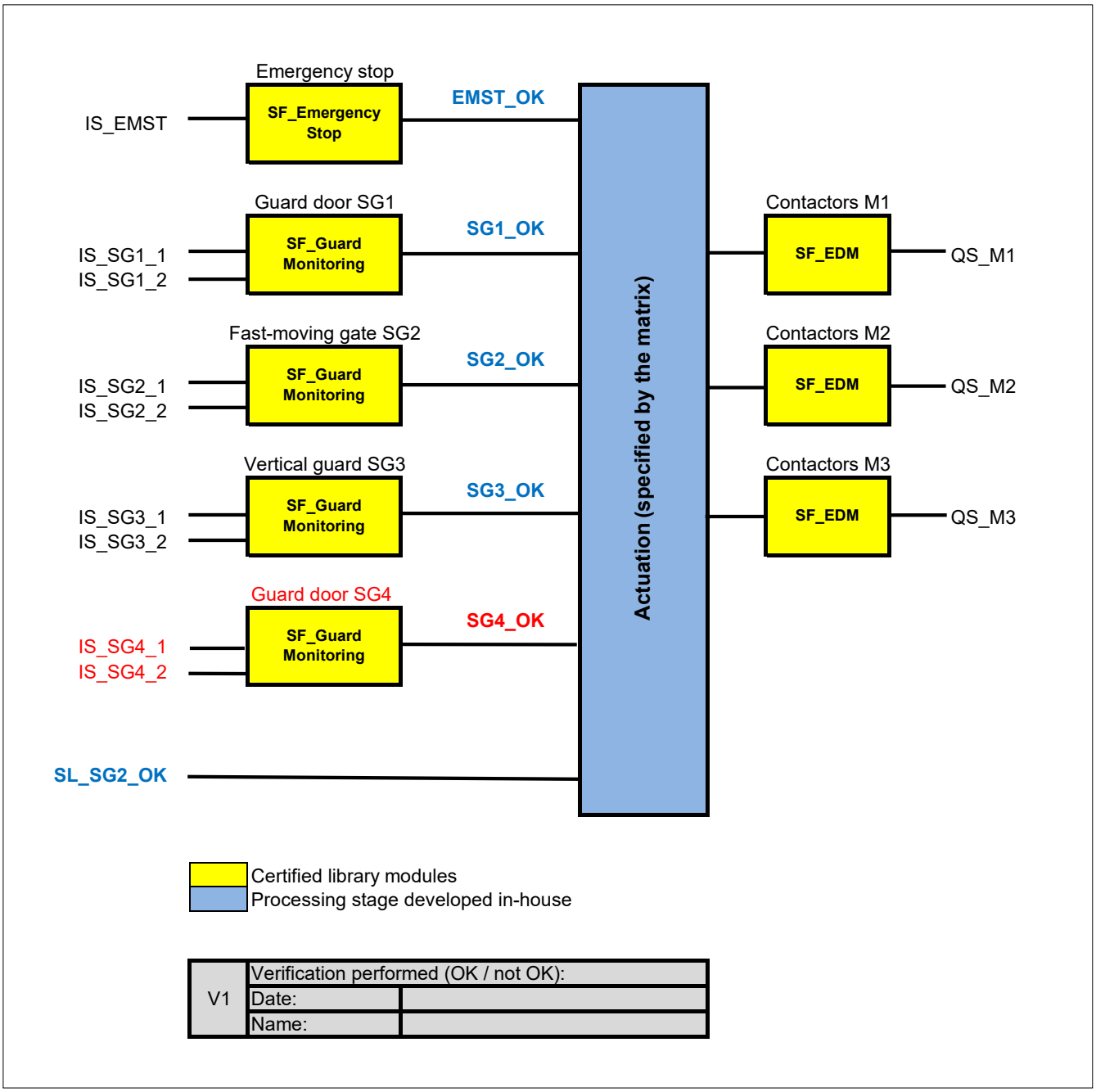


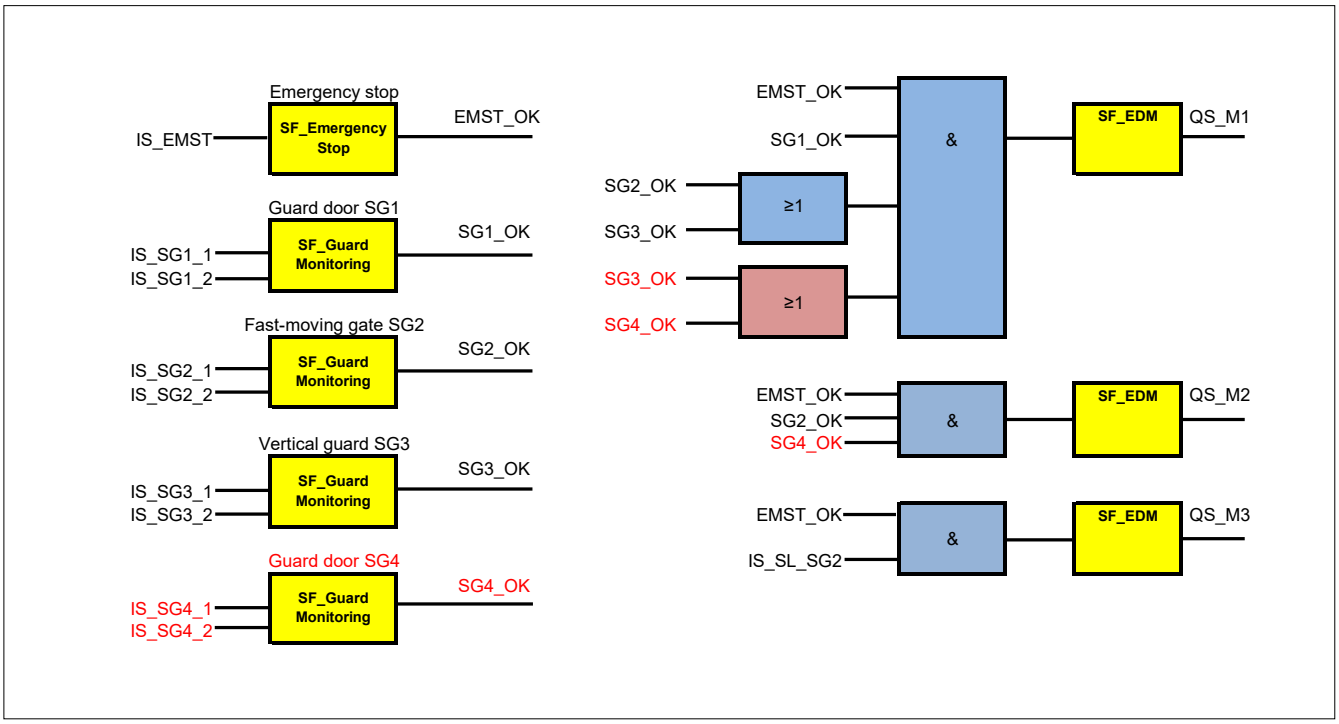
Figure 40:
C&E matrix (document B4) of the robot cell with additional guard door

			Cause												Effect				D1		
			Switching states of the inputs												Outputs						
Mode of operation	Preceding state in the test	State	IS_IMST (I8.4)	IS_SG1_1 (I8.2)	IS_SG1_2 (I9.6)	IS_SG2_1 (I8.1)	IS_SG2_2 (I9.5)	IS_SG3_1 (I8.0)	IS_SG3_2 (I9.4)	IS_SG4_1 (I8.2)	IS_SG4_2 (I9.7)	IS_SL_SG2 (I8.5)	Description		QS_M1 (Q24.0)	QS_M2 (Q24.1)	QS_M3 (Q24.2)	Acknowledge-ment I_ACK (I4.0)	Reviewed (OK/not OK):	Name	Date
													C1: Software consistent with the matrix documentation								
		1	1	1	1	1	1	1	1	1	1	1	**ALL_OK**		ON	ON	ON				
	1	2	0	1	1	1	1	1	1	1	1	1	SF1: Emergency stop actuated		EMST_OK OFF	EMST_OK OFF	EMST_OK OFF	ON			
	1	3	1	0	0	1	1	1	1	1	1	1	SF2: SG1 open		SG1_OK OFF	NOP	NOP	ON			
	1	4	1	1	1	0	0	1	1	1	1	1	SF3: SG2 open		NOP	SG2_OK OFF	NOP	ON			
	1	5	1	1	1	0	0	0	0	1	1	1	SF4: SG2 and SG3 open		SG2_OK v SG3_OK OFF	NOP	NOP	ON			
	1	6	1	1	1	0	0	1	1	1	1	0	SF5: Edge protection sensor actuated		NOP	NOP	IS_SL_SG2 OFF	ON			
	1	**7**	1	1	1	1	1	1	1	**0**	**0**	1	**SF6: SG4 open**		**NOP**	SG4_OK OFF	**NOP**	**ON**			
	1	**8**	1	1	1	1	1	**0**	**0**	**0**	**0**	1	**SF7: SG4 and SG3 open**		SG4_OK v SG3_OK OFF	NOP	**NOP**	**ON**			
V1	Verification performed (OK / not OK):																Software signature:				
	Date:																				
	Name:																				

Figure 41:
Compact matrix presentation of the robot cell with additional guard door

Actuators		Switch offs		C1			D1		
			SFs involved	Software consistent with the matrix documentation			Function validated		
Output	**Description**	**Mode of operation, all**		**OK/not OK**	**Name**	**Date**	**OK/not OK**	**Name**	**Date**
QS_M1	Motor M1	EMST_OK& SG1_OK & (SG2_OK v SG3_OK) & (SG4_OK v SG3_OK)	1, 2, 4, 7						
QS_M2	Motor M2	EMST_OK& SG1_OK & SG2_OK & SG4_OK	1, 3, 6						
QS_M3	Motor M3	EMST_OK & SL_SG2_OK	1, 5						
V1	Verification performed (OK / not OK):			Software signature:					
	Date:								
	Name:								

Figure 42:
Program sketch (document B5) of the robot cell with additional guard door



6.16 Simplification of recurring safety functions

Safety functions such as emergency stop are frequently found on machines. The larger a machine, the more emergency stop switching devices are fitted to it. These switching devices can be handled by a safety function in the software. However, it is important for each individual emergency stop switching device to be tested. The emergency stop switching devices are all summarized in a separate table, serving as an overview for the tests.

Two basic circuit solutions are possible. One is for all emergency stop switching devices to be connected in series on the hardware side and to be assigned to a single binary input per channel in the PLC. The other is for the emergency stop switching devices to be wired to separate inputs on the PLC and the series connection to be effected on the software side by means of an AND element in the PLC.

The discrepancy between the two channels can be monitored both directly on the PLC input card and by means of a library module (e.g. SF_Equivalent according to PLCopen [8]) in the program. It is not necessary for each channel to be tested explicitly to ensure monitoring of the two emergency stop channels for a discrepancy; it is sufficient for the parameterization of the input cards to be checked for correctness, and the correct wiring to be checked by means of the I/O-check.

Figure 43 shows the emergency stop functional test, taken from the example of the rotary table (Section 7.4). In this example, 18 emergency stop buttons are installed on the system and are connected in series for demand of the emergency-stop safety function.

The item codes of all emergency stop buttons are stated, together with validation columns in which the functional test of each emergency stop button is confirmed. When each emergency stop switching device is actuated, a check must be performed of whether the levels on the two inputs of the PLC change. An I/O-check is thus also performed for each emergency stop button to verify that the wiring is correct.

If the emergency-stop switching devices are connected to inputs individually rather than being connected in series, a column showing the inputs can be added to the table in Figure 44.

Whether the emergency stop signal is grouped in the hardware (by connection of the emergency stop switching devices in series) or in the software (by connection of the inputs in series in the PLC) has no bearing upon the function.

On large installations, the use of separate inputs is convenient for troubleshooting, as faults can then be located more quickly.

These tables ensure that all emergency stop switching devices have been tested. The matrix is used to test the safety function; this results in only one emergency stop switching device needing to be actuated for the test.

Figure 43:
Example emergency stop functional test

Emergency stop button I0.0	D1 Function tested		
	OK/not OK	Name	Date
-1S1			
-1S2			
-1S3			
-1S4			
-1S5			
-1S6			
-1S7			
-1S8			
-1S9			
-1S10			
-1S11			
-1S12			
-1S13			
-1S14			
-1S15			
-1S16			
-1S17			
-1S18			

Figure 44:
Example emergency stop functional test with input data

Emergency stop button		D1 Function tested		
IL	Inputs	OK/not OK	Name	Date
-1S1	I0.0/I1.4			
-1S2	I0.1/I1.5			
-1S3	I0.2/I1.6			
-1S4	I0.3/I1.7			
-1S5	I0.4/I2.0			
-1S6	I0.5/I2.1			
-1S7	I0.6/I2.2			
-1S8	I0.7/I2.3			
-1S9	I1.0/I2.4			
-1S10	I1.1/I2.5			
-1S11	I1.2/I2.6			
-1S12	I1.3/I2.7			
-1S13	I10.0/I11.4			
-1S14	I10.1/I11.5			
-1S15	I10.2/I11.6			
-1S16	I10.3/I11.7			
-1S17	I10.4/I12.0			
-1S18	I10.5/I12.1			

6.17 Observance of measures for fault control

A distinction can be drawn between two types of failures in machine control systems: those caused by systematic faults and those by random faults. Systematic faults (errors) should be avoided by a structured and transparent working method. For the control of random faults (caused by component defects, ageing or wear), library modules contain measures for detecting such faults and then de-activating the control signal (output signal) and placing the machine in a safe state.

In the selected modular architecture comprised of inputs stage, processing stage (ACT module) and outputs stage (see Figure 45 with the example from Section 6.2), library modules are used in the inputs and outputs stages. No fault control measures are implemented in the ACT module.

Selected examples of these fault control measures employing function blocks from the PLCopen library [8] are shown below. Specifically, these function blocks are SF_EDM for contactor monitoring, SF_GuardMonitoring for guard door monitoring and SF_EmergencyStop for emergency stop monitoring.

Safety modules in the libraries of other PLC manufacturers work similarly and also feature integral fault detection.

Figure 46 shows the SF_EDM module and its integration into the software. The SF_EDM module monitors the state of the actuated contactor through the readback inputs S_EDM1 and S_EDM2. If the contactor is operating correctly, the states of these inputs are always the inverse of that of the S_EDM_Out output. If this is not the case, a readback error is detected and saved after the MonitoringTime time parameter has expired. If the readback error is active, output S_EDM_Out can no longer be switched to TRUE, i.e. the contactor for example cannot be energized.

Figure 47 shows the call to the SF_GuardMonitoring module. SF_GuardMonitoring monitors a guard door . As soon as one of the two guard door contacts (inputs S_GuardSwitch1,2) assumes the FALSE state, the output S_GuardMonitoring is also set to FALSE. S_StartReset controls the acknowledgement behaviour. Automatic acknowledgement may be activated only if it is not possible for the system to cause an automatic start-up of the machine.

In the final example, Figure 48 shows the call to the SF_EmergencyStop function block, which monitors the emergency stop. The SF_EmergencyStop module enables emergency stops with stop category 0 to be implemented. The output S_EStopOut is set to FALSE as soon as a FALSE signal is present on the S_EstopIn input. The outputs are not reactivated until an acknowledgement. Automatic acknowledgement may be set with S_AutoReset only if automatic start-up of the machine is prevented by some other means.

Fault control measures of this kind are found in many safety library modules of control system manufacturers. The use of library modules is also specified for this reason in the programming rules. As a rule, automatic re-starting is also prohibited, in order to prevent a machine from performing unanticipated and possibly hazardous actions.

Figure 45:
Example modular architecture

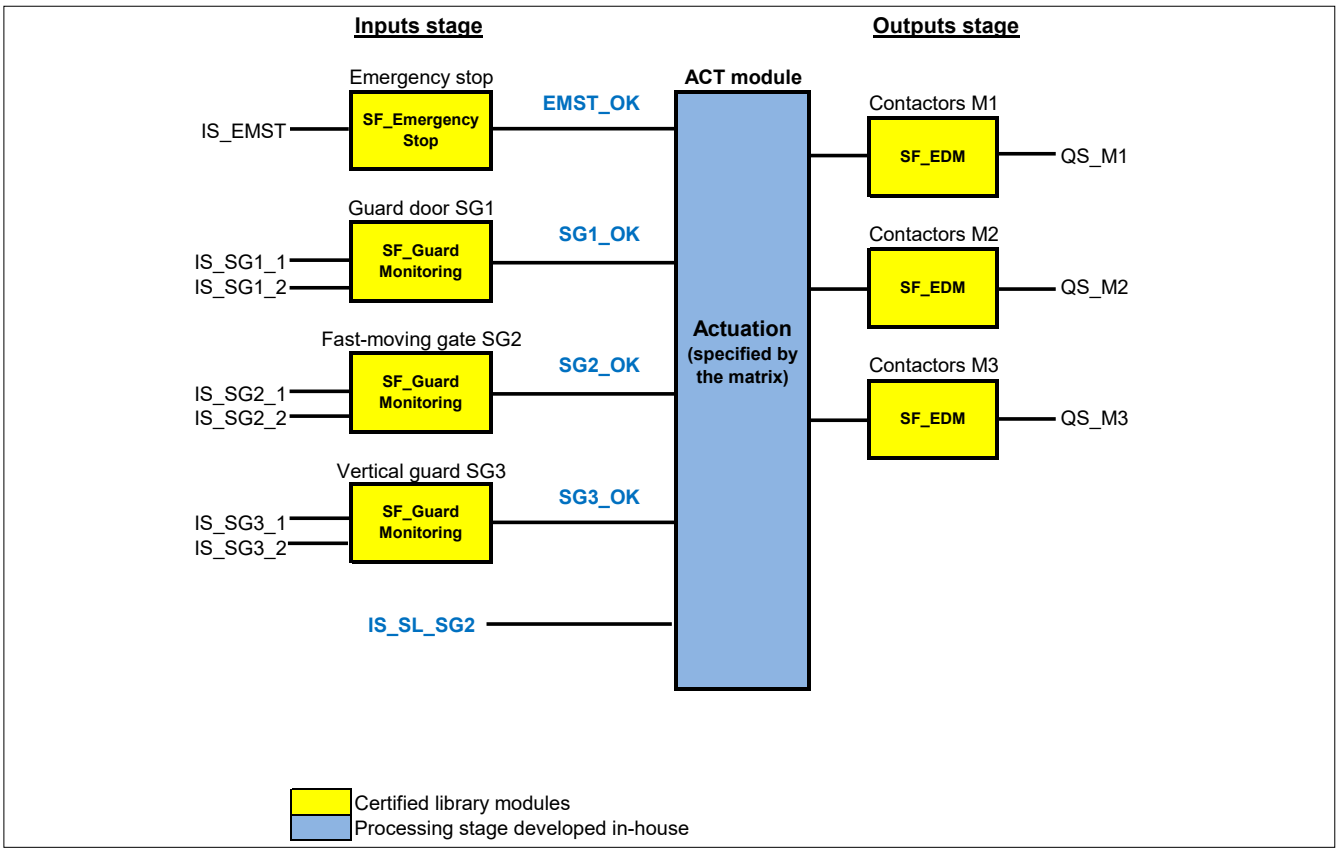


Figure 46:
Calling of the library module SF_EDM

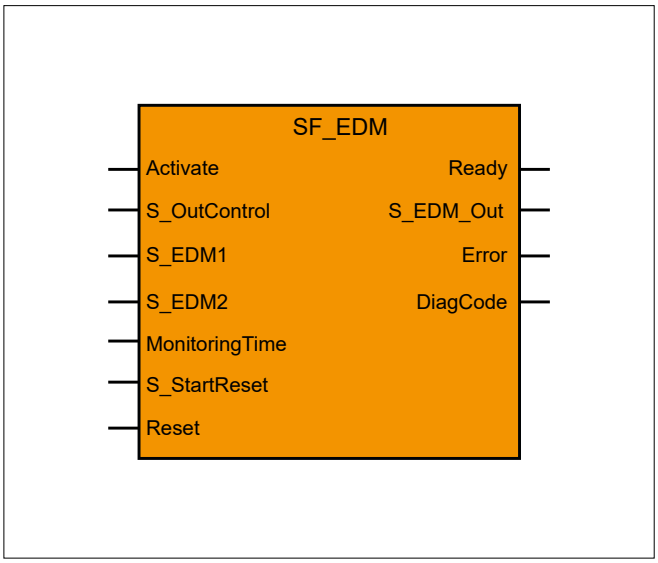


Figure 47:
Calling of the library module SF_GuardMonitoring

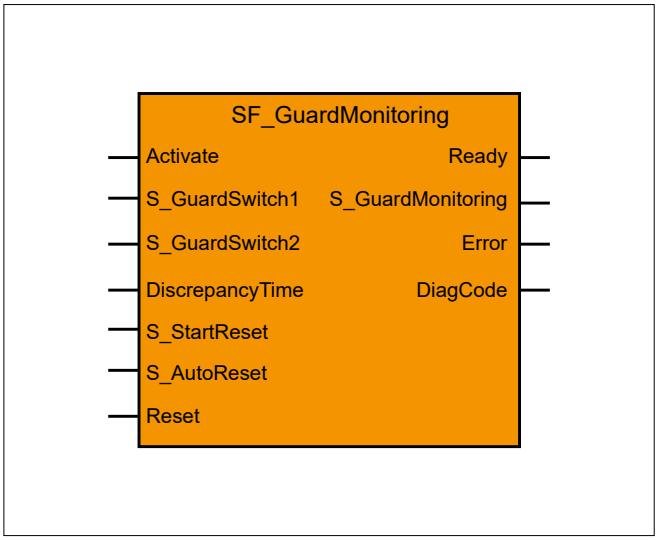
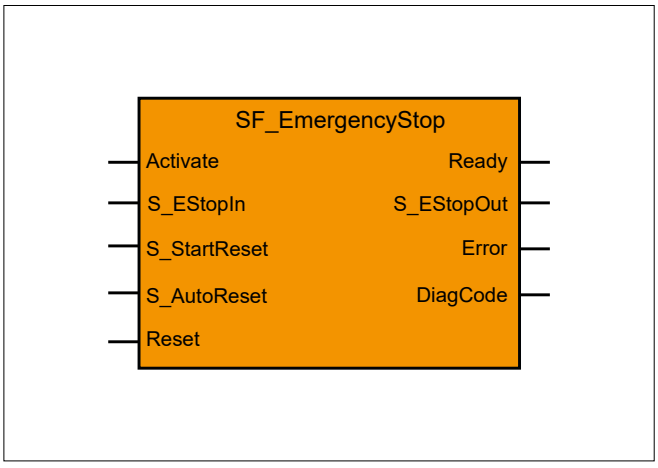


Figure 48:
Calling of library module SF_EmergencyStop



7 Overview of the software examples covered

This report began by discussing the design of safety-related application software (SRASW) in general terms. It describes the IFA matrix method step by step with reference to an example. Some of these steps, however, require practice. Furthermore, owing to the diversity of possible safety functions and their implementation, the steps do not lend themselves to being described in generic terms. This chapter therefore presents the documentation of a large number of software examples that implement typical safety functions. All examples are documented according to the matrix method and are available for download.

The circuit diagrams are block diagrams and are limited to presentation of the safety function(s) and the relevant components required for their implementation. In the interests of clarity, certain additional circuitry that is normally required has been omitted, for example that for the assurance of electric shock protection, for control of overvoltage/undervoltage and overpressure or low pressure, for the detection of insulation faults, short-circuits and earth faults, for example on lines routed externally, or for assurance of the required resistance to electromagnetic interference. Such details include protective circuitry in the electrical system, such as fuses and diodes, for example in the form of freewheeling diodes or decoupling diodes. In accordance with the fault lists in DIN EN ISO 13849-2, issues such as the influence of conductor short circuits must of course also be considered in relation to the safety function concerned and the conditions of use. All components used must therefore be selected with consideration for their suitability according to their specification. Overdimensioning is one of the well-tried safety principles.

The example controls with circuit diagrams and explanatory texts are therefore limited to essential aspects and should not be regarded as suggestions for implementation. All documents for the example circuits were created in the SOFTEMA tool (see Chapter 14) in the prototype version available at the time this report was created. They therefore differ from the example documents in the research report of DGUV project FF-FP0319 (see Section 2.2).

The following sections briefly describe the main functions of the examples. Further information can be found in the corresponding Microsoft Excel documents. The control programs themselves were created in DGUV project FF-FP0319 by the project's authors. Siemens SIMATIC STEP7 V5.5 and Distributed Safety V5.4 were used for this purpose. An exception was the last example, which was created with Pilz PNOZmulti Configurator 9.0.1.

The password for the program listings of the Siemens examples is: pw_fcpu. The level 1 password for the example with PNOZmulti is: 1.

The Excel documents can be found in a single archive file in the download area of this IFA report. They can be opened and viewed in SOFTEMA (Chapter 14). The program data is available only on the website of project FF-FP0319 [5].

Table 21 shows the software examples discussed and the associated Excel files and program listings.

The Excel files are divided into a number of worksheets. An overview of these can be found in Table 22.

Table 21:
Overview of the software examples covered

Section	Software example	Excel document (.xlsx) (on report page)	Program listing (.zip) (on project page)
7.1	Robot production cell	Robot production cell	Robot production cell
7.2	Robot production cell with setup mode	Robot cell setup mode	Robotcell_setup
7.3	Robot production cell with additional guard door	Robot cell with modification	Robotcell_modification
7.4	Rotary table system	Rotary table system	Rotary table system
7.5	Machine tool	Machine tool	Machine tool
7.6	Safely limited speed (SLS) with standard FI	SLS with standard FI	SLS_StandardFI
7.7	Safely limited speed (SLS) with safety FI	SLS with safety FI	SLS_SafeFI
7.8	Muting	Muting	Muting
7.9	Two-hand control	Two-hand control	Two-hand control
7.10	PNOZmulti	PNOZmulti	PNOZmulti

Table 22:
Overview of the worksheets in the Excel documents

Worksheets used
V-model
A1 Safety functions
System sketch
A2.2 Circuit diagram
A2.3 System structure
A2.4: I/O list
A3 Measures
A4 Requirements
B1 Safety program architecture
B2 Standard program architecture
B3 Modular architecture
B4 Matrix C&E
B4 Matrix compact
B5 Program sketch
C1 Codereview
D1 Validation
Changes
Persons
Project
Documents
Protocol

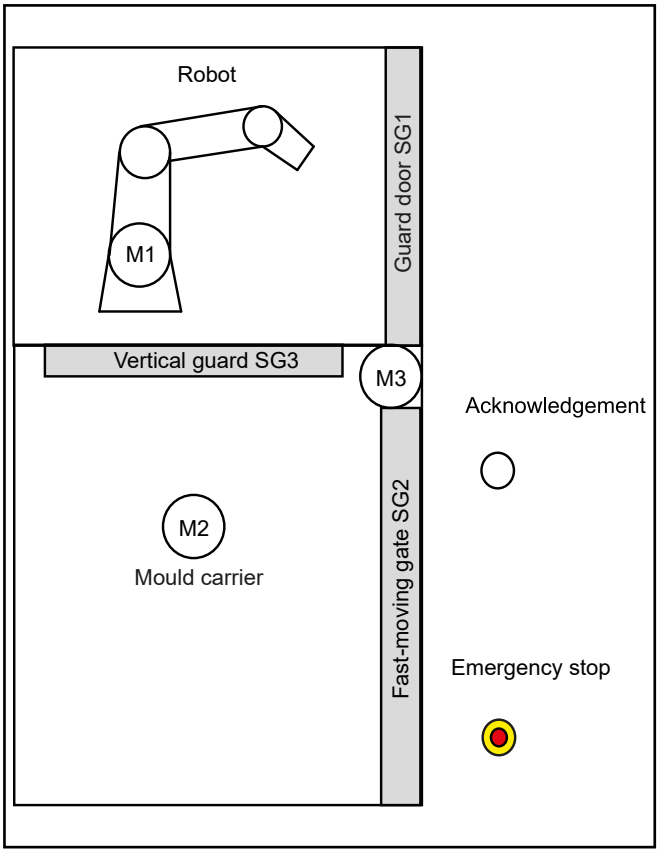
The first worksheet, “V-model”, shows an overview of the two V-models for software and module development, together with the associated documents. The subsequent worksheets bear the names of the V-model documents. Section 5.5 also describes the respective content. The “Changes” worksheet is used to track modifications made to the application program. The procedure for this is described in Section 6.15. The “Persons” worksheet can be used to document the personnel involved in development, validation and testing. The “Project” worksheet contains all project-related information for the application program. Project-related documents and files can be managed in the “Documents” worksheet. Changes relevant to safety (e.g. changes to the PL_r of the safety functions) are documented on the “Protocol” worksheet.

7.1 Robot production cell

In the example of a robot production cell, a simple production cell as shown in Figure 49 is discussed. Automatic mode is the only operating mode. The example is also described in Section 6.2.

In the installation, the mould, which is mounted on the mould carrier M2, is filled with foam by a robot M1. The robot then returns to its staging area. The vertical guard SG3 is an automatic guard that should open only when the robot fills the mould. Once the foam has cured, a machine operator opens the SG2 fast-moving gate and retrieves the finished workpiece from the mould. The machine operator then closes the fast-moving gate again. An edge protection sensor SL_SG2 (not shown) is fitted to the fast-moving gate SG2 to protect operating personnel against crush injuries. When the edge protection sensor is tripped, the motor M3 of the fast-moving gate is switched off immediately. The mould carrier must be stationary while the fast-moving gate is open. Should the vertical guard not be closed when the fast-moving gate is opened, the robot must also be halted, for safety reasons. The guard door SG1 provides access to the robot for the performance of maintenance work. The robot must be at a standstill while the guard door is open. If the emergency stop EMST is actuated, the robot, the mould carrier and the high-speed guard are all brought to a standstill.

Figure 49:
System sketch of the robot production cell

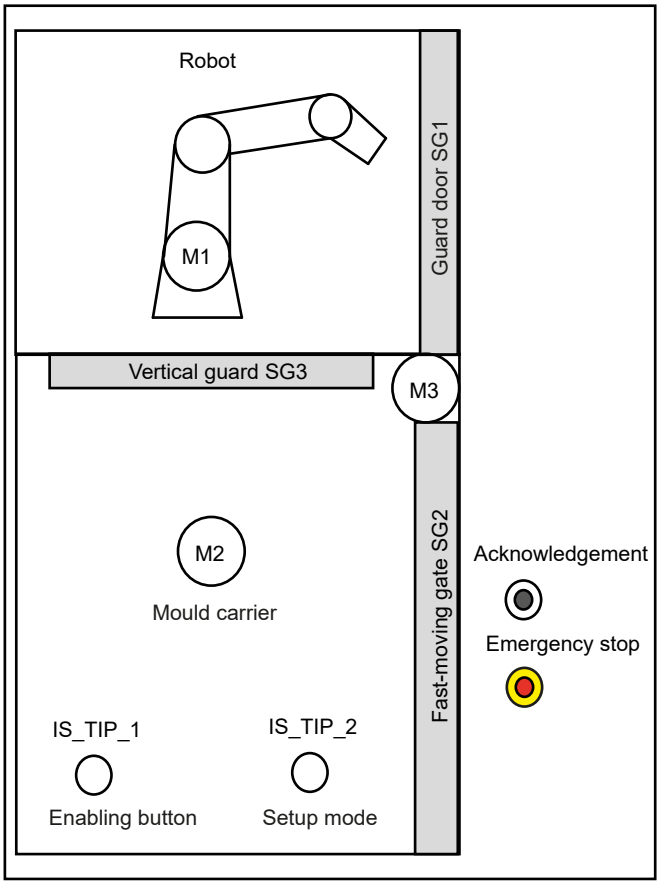


7.2 Robot production cell with setup mode

Figure 50 shows the System sketch for the example of a robot production cell with setup mode. This represents an extension of Example 7.1. This example features automatic and setup modes. The example is also described in Section 6.11.

The automatic mode function is that described in Section 7.1. When the fast-moving gate SG2 is open and the vertical guard SG3 is closed, the mould carrier M2 can be moved in setup mode at a safely limited speed (SLS) by actuation of one of the enabling buttons 3S1, 3S2.

Figure 50:
System sketch of the robot production cell with setup mode

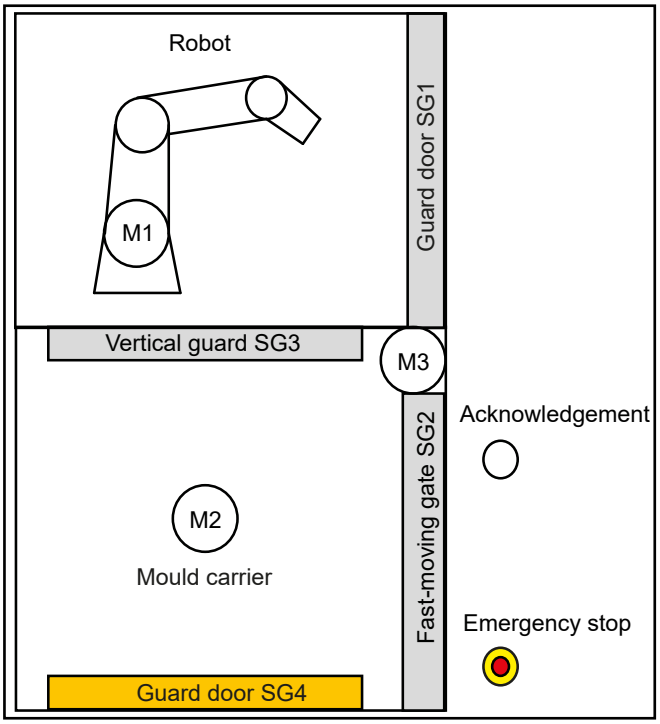


7.3 Robot production cell with additional guard door

The System sketch of the robot production cell with additional guard door, shown in Figure 51, is an extension of Example 7.1 with the addition of the guard door SG4. Automatic mode is the only operating mode. The example is also covered in Section 6.15.

The function is the same as that implemented as in Example 7.1. In addition, the mould carrier M2 is halted when the guard door SG4 is opened. The robot M1 must be halted if the vertical guard SG3 is open at the same time as the guard door SG4.

Figure 51:
System sketch of the robot production cell with additional guard door

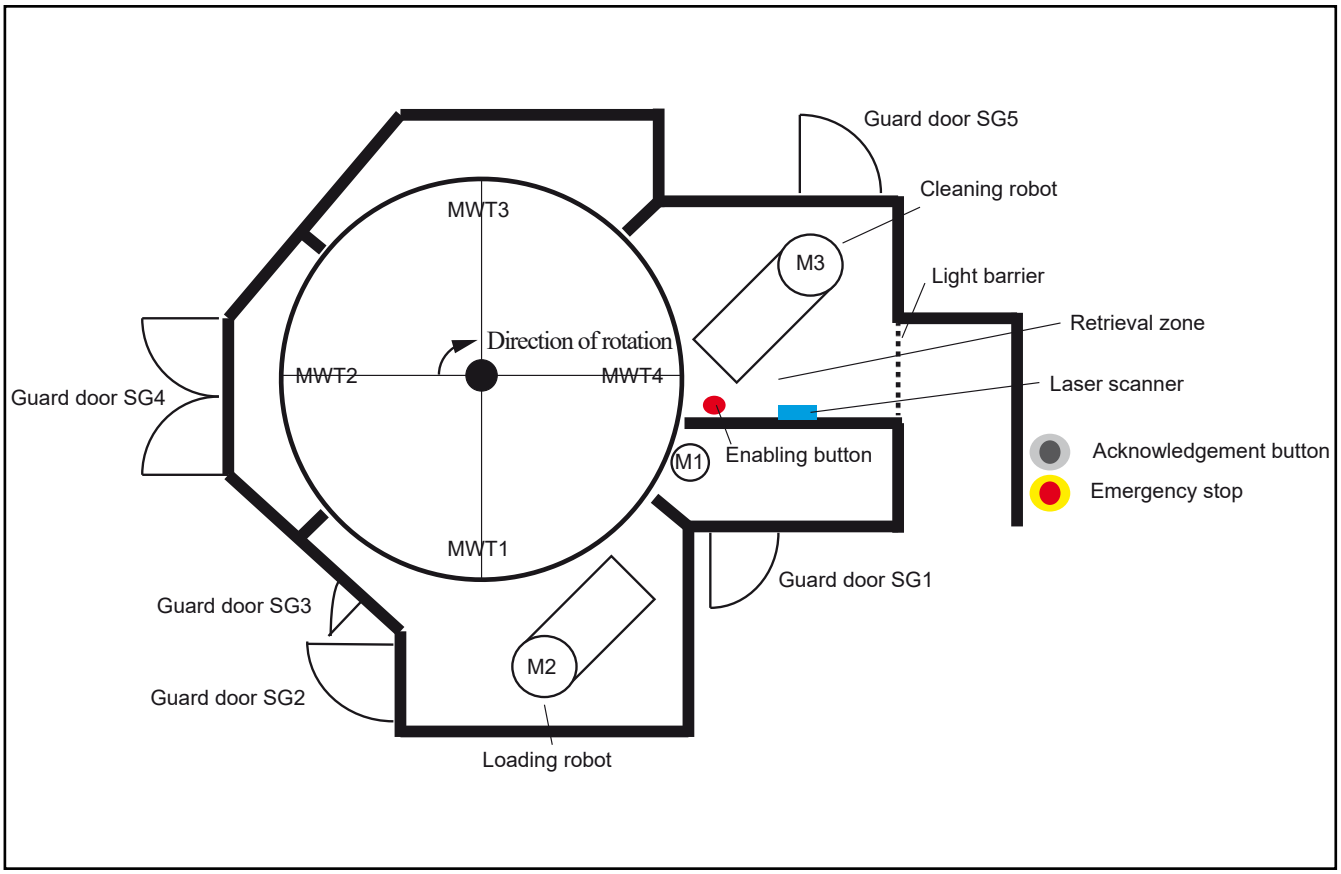


7.4 Rotary table system

Figure 52 shows the System sketch of the example featuring the rotary table system. Automatic and setup operating modes are implemented in this example.

This system produces workpieces in cyclic operation. The loading robot M2 loads a mould onto a workpiece carrier. The rotary table then rotates one cycle step further and the next workpiece carrier is filled. Each workpiece thus cures over two cycle positions. In the fourth position, a machine operator retrieves the workpiece. A robot M3 then cleans the workpiece carrier. The retrieval zone is monitored by a light barrier LB. In addition, a laser scanner LC detects the presence of persons in the retrieval zone. The workpiece carriers must all be fitted with workpiece moulds or protective mesh for bypassing the safety contacts. Should a mould or the relevant protective mesh be missing, all four workpiece carrier motors MWT1 to MWT4 and the rotary table motor M1 are switched off, and the loading robot and cleaning robot are halted. Should one of the guard doors SG1, SG2, SG4, SG5 be opened or the emergency stop EMST actuated, all workpiece carrier motors are switched off and the rotary table, the loading robot and the cleaning robot are halted. Likewise, all motors are switched off if the light barrier is interrupted and a workpiece carrier is not present in the retrieval zone. When the guard door SG3 is opened, the loading robot is brought to a halt. The loading robot is also brought to a halt when axis 1 of the loading robot impacts a limiting device. If axis 1 of the cleaning robot impacts a limiting device, it is brought to a halt. If the light barrier is interrupted whilst a workpiece carrier is present in the retrieval zone, the motor of the workpiece carrier concerned is switched off. If axes 1 and 2 of the cleaning robot are not in their initial position when the light barrier is interrupted or the laser scanner is tripped, the cleaning robot is brought to a halt. If the laser scanner is tripped whilst the cleaning robot is in its initial position, the rotary table can be moved at a safely limited speed (SLS) by means of the enabling button.

Figure 52:
System sketch of the rotary table



7.5 Machine tool

The “Machine tool” example (Figure 53) describes a multi-axis milling machine. This machine possesses automatic and setup operating modes. The example is based on an example from [19].

The machine has a moving workpiece table. The table can be moved in the x-axis with the motor X1 and rotated by the motor C. The milling tool can be moved to all spatial coordinates by the axis motors X2, Y and Z. The tool is rotated by the spindle motor S. The automatic tool changer is moved by the rotary axis W and the pneumatic axis X3. The axes X1, C, X2, Y, Z, S and W are moved by drives with integral safety functions.

In automatic mode, the guard doors must be closed and locked. All guard doors are equipped with electromechanical guard locking. If the emergency stop EMST is activated, all drives are halted with SS1 and the pneumatic axis X3 is disconnected from its power supply. SS1 stands for safe stop 1, i.e. the drive is ramped down and then disconnected from the power supply with STO (safe torque off).

For guard doors SG1 or SG2 to be opened, all drives are shut down with SS2 and the power supply to X3 is disconnected. SS2 stands for safe stop 2, i.e. the drive is ramped down and a safe stationary state then monitored with SOS (safe operating stop). This response also occurs when the mode is switched to setup mode.

For the guard door SG3 to be opened, the drives of the tool (X2, Y, Z, S) and the tool changer W are run down by SS2 and the axis X3 is halted. The axes of the workpiece table remain active, as they cannot be reached from the guard door SG3.

In setup mode, opening of the guard doors can be demanded by means of pushbuttons. To ensure that the pneumatic axis X3 is stationary, guard locking of the relevant guard door is released only after a five-second delay. In addition, the remaining axes must signal safe standstill by SOS. Standstill of axes X1 and C is not queried for the guard door SG3.

If the guard door SG1 is open and the other guard doors are closed, the axes of the workpiece table (X1, C) and the movement axes of the tool (X2, Y, Z) can be moved in setup mode at safely limited speed (SLS) with the “tap forwards“ or “tap backwards” buttons. The axes to be moved are selected by means of buttons. With guard door SG2 open and guard doors SG1 and SG3 closed, the rotary axis of the tool changer W can be moved in setup mode at safely limited speed (SLS) with the “magazine forwards“ or “magazine backwards“ buttons. These movements are required for fitting the tool to the tool changer.

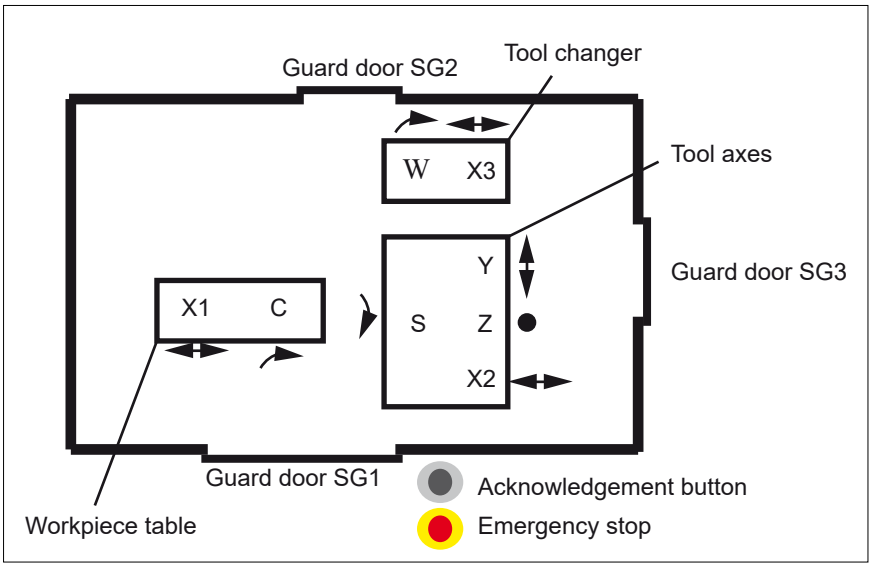


Figure 53:
System sketch of a machine tool

7.6 Safely limited speed (SLS) with standard FI

The example of “SLS with standard FI” shows the structure of a drive control system with safety PLC (Figure 54). Since the frequency inverter has no integral safety functions, the speed must be read in by two rotary encoders G1 and G2 and the speeds compared. This two-channel arrangement enables the safety function to be implemented.

This example features automatic and setup operating modes. In automatic mode, the motor can turn when the guard door is closed. In setup mode, the motor may be run at safely limited speed (SLS) with the guard door open by means of the push button S1. If emergency stop is actuated, the motor is always switched off.

This example is taken from BGIA Report 2/2008 [2] (Example 21).

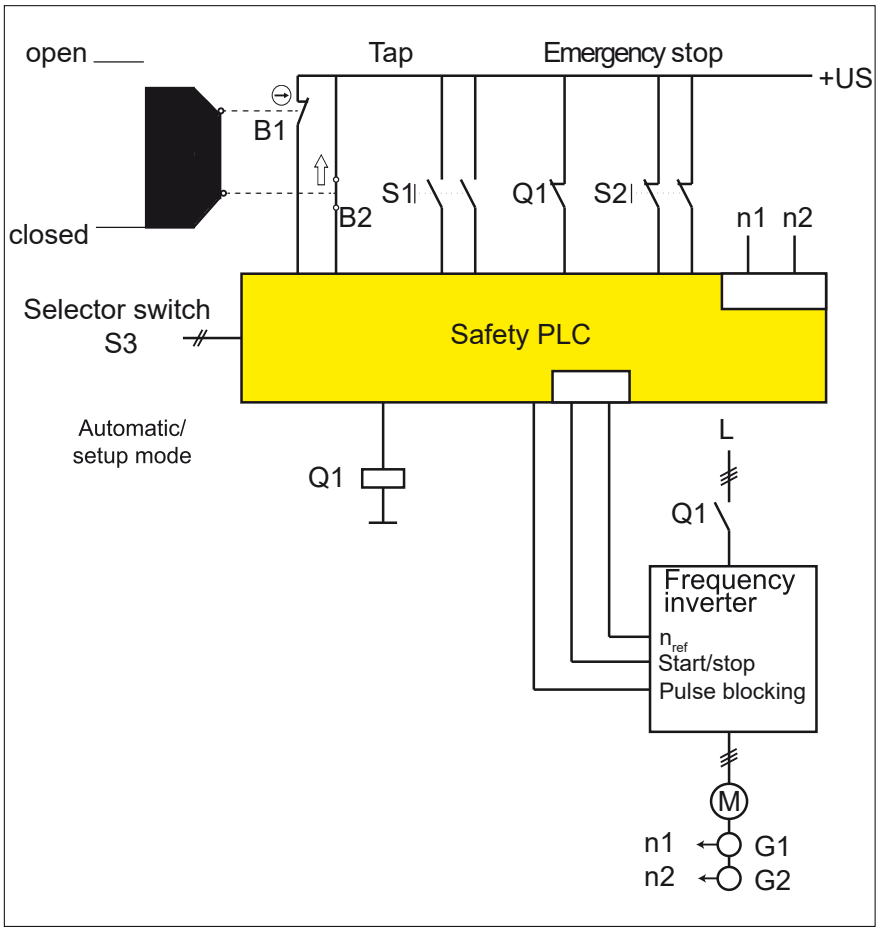


Figure 54:
Hardware structure of the “SLS with standard FI” example

7.7 Safely limited speed (SLS) with safety FI

Figure 55 shows the hardware structure of the “SLS with safety FI” example. This example compares the overhead of using drive controls with integral safety functions and those without, as in the example in Section 7.6. In this example, the safety functions of the drive control are actuated by the safety PLC.

The safety functions are the same as in the example in Section 7.6: emergency stop, guard door monitoring and SLS in setup mode.

When emergency stop is actuated, the motor is stopped with the safety function SS1. If the guard door is opened, the motor is stopped with SS2 and its stationary state monitored automatically with SOS. From this stationary state, the safely limited speed (SLS) can be activated by means of the push button.

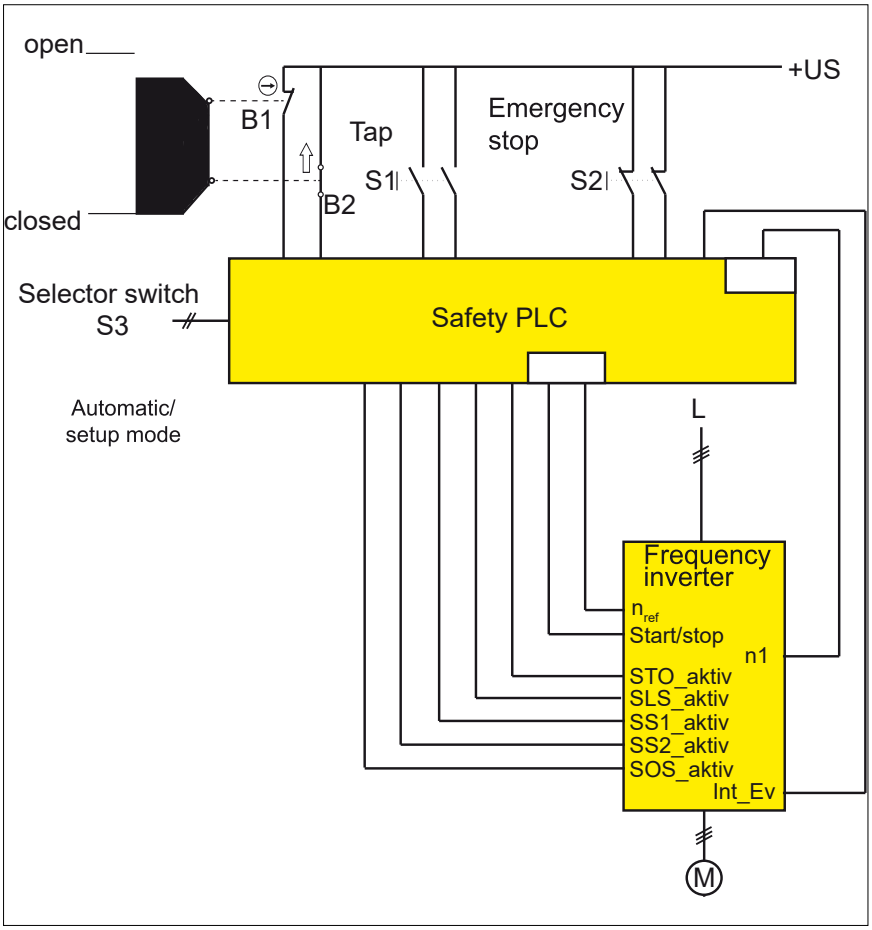


Figure 55:
Hardware structure of the
“Safely limited speed with safety FI” example

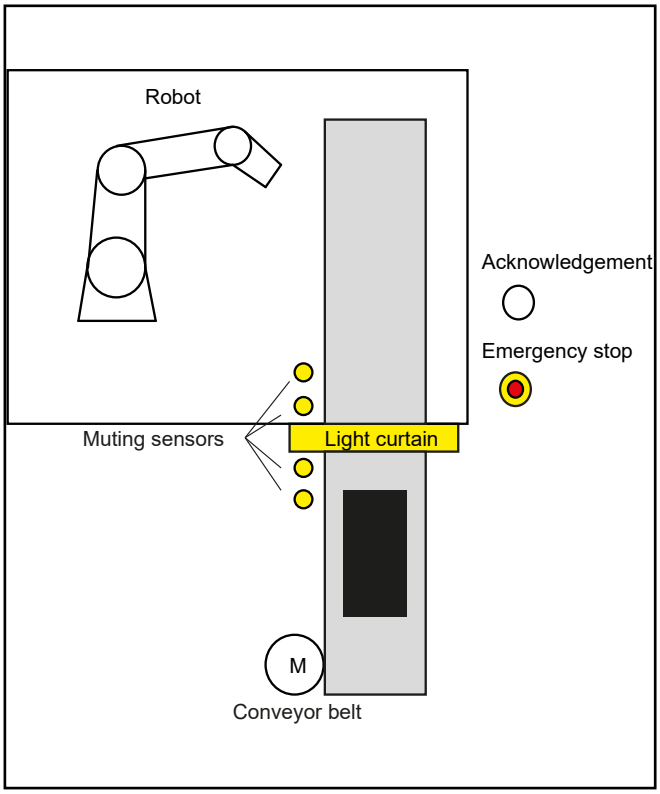
7.8 Muting

The “Muting” example describes a robot production cell into which the material is fed through a light curtain (Figure 56). Automatic mode is the installation’s only operating mode.

The workpieces are transported to the robot’s production area by a conveyor belt. In the process, they pass through a light curtain. The workpiece triggers the muting sensors in the correct sequence and can therefore pass through the light curtain without stopping the system. Following processing, the workpiece moves back out of the working area.

If the light curtain is interrupted without the muting sensors being triggered correctly, enabling of the conveyor belt and the robot is switched off. Actuation of the emergency stop button has the same effect.

Figure 56:
System sketch of a production cell with material transport through a light curtain

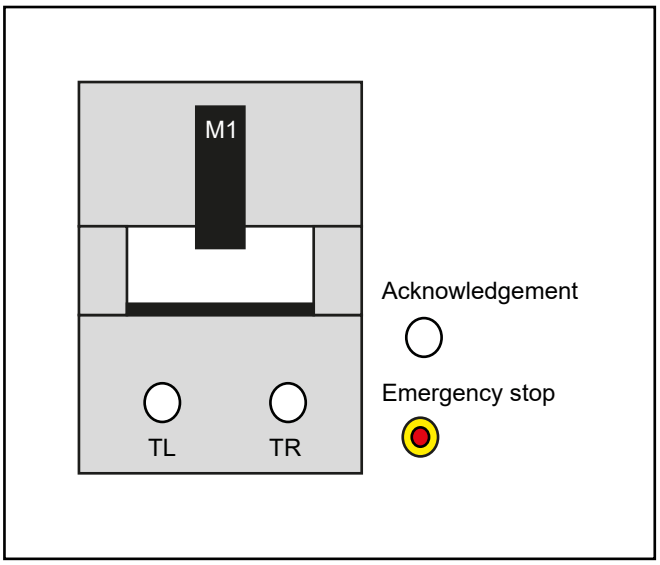


7.9 Two-hand control

The “Two-hand control” example describes a press (Figure 57). Setup is this machine’s only operating mode.

The machine operator places a workpiece in the press zone and must then press both push buttons of the two-hand control at the same time to enable the motor M1. The motor is stopped immediately when the emergency stop button is actuated.

Figure 57:
System sketch of a press



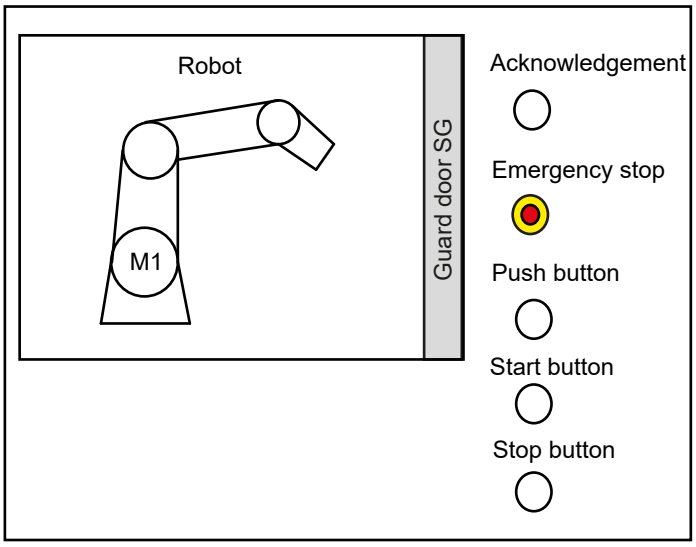
7.10 Configurable switching device

In this example, a configurable switching device monitors a robot production cell (Figure 58). The arrangement has automatic and setup operating modes.

In automatic mode, the robot moves when the guard door SG is closed. In setup mode, the robot can be moved at safely limited speed (SLS) with the guard door open with enabling by means of an push button.

This example is intended to illustrate that the IFA matrix method for software specification and documentation of SRASW is also suitable with the use of configurable switching devices.

Figure 58:
System sketch of a robot production cell, monitored by means of a configurable switching device



8 Role of embedded software for application programming

Development and execution of SRASW is possible only with software embedded in a programmable electronic control system (embedded software, SRESW: firmware, runtime system, etc.). Embedded software therefore makes a significant contribution to the reliability of the safety functions. DIN EN ISO 13849-1 does not differentiate in its requirements between programmable standard and safety components. The requirements for the realization of SRESW and SRASW (Section 4.6 of the standard) must therefore be met for standard electronic programmable components (such as a standard PLC), as well as for safety components.

When a certified safety-related control system is used, only Section 8.1 in this chapter must be observed.

8.1 Role of the SRESW of a safety-related control system

The SRESW of a safety component within the meaning of the Machinery Directive is linked inseparably to the control electronics and is therefore usually developed, tested and certified together with the latter, usually in accordance with DIN EN 61508 and DIN EN ISO 13849-1. The reliability of this SRESW is certified by the manufacturer by an SIL or the performance level. Safety functions with PL_r e may be implemented by programmable controllers with PL e. The same applies by analogy to lower PLs.

8.2 Evaluation of the SRESW of a standard controller

Safety-related controls for industrial applications are often implemented by means of standard components. Restrictions apply in this case, however, from which tested and certified safety components are exempt. The SRESW of a standard component must also be evaluated in all cases. Either the SRESW has been developed itself in accordance with DIN EN ISO 13849-1, Section 4.6.2, or the manufacturer of the standard component is able to certify this. Neither is common; it is therefore not usually possible to prove that the requirements for SRESW are met for a programmable standard component. How is this addressed in application programming?

Whereas the 2008 version of DIN EN ISO 13849-1 did not set out any specific requirements in this respect, Amendment 1 of DIN EN ISO 13849-1 [1], Section 4.6.2 contains the following new requirement:

For components for which SRESW requirements are not fulfilled, e.g. PLCs without safety rating by the manufacturer, these components may be used under the following alternative conditions:

- the SRP/CS is limited to PL a or b and uses category B, 2 or 3;

- the SRP/CS is limited to PL c or PL d and may use multiple components for two channels in category 2 or 3. The components of these two channels use diverse technologies."

Caution: the descriptions in BGIA Report 2/2008 [2], Section 6.3.10 thus become obsolete.

These new requirements apply to embedded software only. However, when components without a safety assessment are used by the component manufacturer, numerous other requirements in accordance with DIN EN ISO 13849-1 must be met in addition to those for embedded software, e.g:

- With regard to the avoidance and control of systematic faults or suitability for the foreseeable environmental conditions, e.g. climate, vibration, electromagnetic compatibility.
- Each component used in an SRP/CS must satisfy at least the requirements of category B (since all categories contain the basic requirements of category B). Category B requires compliance with the applicable standards, e.g. EN 61131-2 for PLCs, EN 61800-2 for frequency inverters, EN 60947-5-2 for proximity switches.
- Systematic failures must be taken into account for each SRP/CS in accordance with Annex G.

The new SRESW requirements stated above apply not only to programmable controllers without a manufacturer's safety assessment, but to all components used in an SRP/CS that contain embedded software. These include, for example, frequency inverters and intelligent sensors (e.g. encoders, proximity switches).

Table 23 shows the possible combinations of PL and category with standard components and whether, and if so how, the requirements applicable to SRESW are to be met.

Table 23:
Requirements for the SRESW of standard components (to DIN EN ISO 13849-1:2015)

No	PL	Category	Conditions	Requirements for the SRESW of the standard components
1	a or b	B, 2, 3	Compliance with relevant product standards Quality-assured development as a basic safety principle	No SRESW requirements apply to industrial standard components
2	a, b, c	1	Electronic components cannot generally be used for implementation, since they are not considered well-tried components in the sense of DIN EN ISO 13849-1, Section 6.2.4	Not applicable
3	c or d	2 or 3	As No 1 Two channels with diverse technology; the required fault detection (diagnostic coverage, DC) is implemented by SRASW	No SRESW requirements apply to industrial standard components
4	c or d	2 or 3	Two channels without diverse technology; the required fault detection (diagnostic coverage, DC) is implemented by SRASW	Full SRESW requirements in accordance with DIN EN ISO 13849-1, Section 4.6.2 apply, including to industrial standard components. A safety analysis by the component manufacturer is required
5	e	3 or 4	DIN EN ISO 13849-1, Section 4.6.2 states that PL e is not possible for standard components	Not applicable

A note on “technological diversity”: this means that owing to the diversity of two channels (the difference in the technologies employed), the probability of a dangerous failure of the SRP/CS being caused by an error in the SRESW is strongly reduced. Systematic failures and common cause failures are relevant in this context.

The requirement for “technological diversity” can normally be regarded as satisfied in the following examples:

- One channel (functional channel or test channel) employs components containing embedded software. The second channel employs solely components without embedded software, i.e. mechanical, electronic, electromechanical, pneumatic or hydraulic components.

- The two channels employ diverse embedded software, such as different operating systems running on identical or different hardware.
  Note: when identical hardware is used, particular attention must be paid to the systematic capability of the components for the required performance level.

- The two channels employ different hardware (e.g. microprocessors with different processor cores). It is assumed in this case that the associated embedded software was programmed in different development environments.

The requirement for “technological diversity” can normally be regarded as not being satisfied in the following examples:

- The two channels employ components of the same kind from different manufacturers, without more detailed information on the diversity of the embedded software. In this scenario, it cannot normally be ruled out that the two manufacturers use the same embedded software components, and possibly even identical hardware (brand labelling).

- The two channels employ different component types from the same manufacturer, without more detailed information on the embedded software.

9 Use of standard controllers for SRASW

A question often asked: Can safety-related control systems be implemented with standard components such as a standard PLC? The IFA published a position paper [20] in 2011 on this subject. The paper stated that "*safety-related control systems can in principle be implemented with the use of standard components. However, safety components offer the advantage that their manufacturer relieves the machine designer of the task of safety-related assessment and analysis of these components when they are used. Besides the use of a suitable architecture (category), implementation of a required fault detection and consideration of failure rates/probabilities, attainment of functional safety requires consideration to be given to components' systematic capability. In general, the use of complex elements or subsystems of the same design (homogeneous redundancy) should be ruled out, as questions regarding systematic capability and the required diagnostic coverage often can not be adequately answered.*"

Standard components can therefore be used in principle, but with increased overhead and greater responsibility on the part of the party implementing the control system. Homogeneous redundancy, for example by the use of two identical standard PLCs, is unsuitable, owing to the almost complete impossibility of assessing the probability of systematic hardware and SRESW failures of these components (since they have not been tested or certified). Details of the requirements concerning the SRESW of standard components can be found in Section 8.2.

This chapter describes the conditions under which the SRASW is to be developed when standard components are used. A requirement in this case is that use of the programmed standard components (i.e. their hardware and SRESW) is in fact permissible.

9.1 Determining the necessary measures for fault avoidance

As with safety components, the selection of measures for fault avoidance for the SRASW used in a standard component is based upon the required performance level of the implemented safety function. However, as the critical impacts of a software error are reduced in two-channel architectures by diversity in programming and technology, and the probability of a dangerous failure of the safety function is thereby also reduced, the IFA recommends that the requirements/their effectiveness can be reduced by one PL_r in some of the cases described below. This can be inferred from Section 7.4.3, Synthesis of elements to achieve the required systematic capability, of DIN EN 61508-2:2010.

9.2 Single-channel architectures

Single-channel architectures are the typical use case for a standard component such as a standard PLC: a single-channel architecture of category B for a maximum possible PL_r of b. No diagnostic coverage level need be attained for the component. The SRASW must be implemented with use of the basic measures for fault avoidance (Section 3.2). The IFA matrix method can be used for this purpose (Section 9.4).

Category 1 is also single-channel, but requires well-tried components in accordance with DIN EN ISO 13849-1, Section 6.2.4. This excludes complex electronic components. PL_r c can therefore be attained only with two-channel architectures.

9.3 Two-channel architectures

If a programmable standard component is used for each channel in two-channel architectures, the SRASW of the two components can be either the same, or diverse. The distinction between these two cases is described below. In this context, category 2 is also included among the two-channel architectures (one functional channel, one test channel).

9.3.1 Characteristics of diverse SRASW

Diverse SRASW means that two (or more) programs based on the same specification are intended to perform the same tasks but have been developed differently. This is intended to reduce the probability of a dangerous failure caused by the simultaneous occurrence of a systematic fault in the SRASW. Characteristics of diverse SRASW may for example be:

- Different programmers or programming teams
- Different designs (software structure, algorithms, etc.) produced by the same people
- Different programming languages: text-based language (ST) vs. graphical language (FBD)
- Different development environments: free programming of a PLC vs. graphical configuration of a control relay

9.3.2 Two channels with identical, homogeneous SRASW

If a programmed standard component in one channel of the part of the control system is used redundantly with another programmed standard component in the other channel and the two items of SRASW are not programmed differently (for characteristics of diversity, refer to Section 9.3.1), the normative requirements for the required PL (Section 3.2) apply in full to both items of SRASW, owing to the probability of a simultaneous

dangerous failure caused by systematic faults. Independently of this requirement, the normative requirements for the SRESW must also be met (Section 8.2).

9.3.3 Two channels with diverse SRASW

If a programmed standard component in one channel of the part of the control system is used redundantly with another programmed standard component in the other channel in category 3 or 4, and the two items of SRASW are programmed differently (see Section 9.3.1), the normative requirements for both items of SRASW can be lowered by one PL (e.g. requirements for PL_r c instead of PL_r d, see Section 3.2), owing to the lower probability of a simultaneous dangerous failure caused by systematic faults in the two items of SRASW. In the case of category 2, the requirements can be lowered only for the SRASW of the test channel. Independently of this requirement, the normative requirements for the SRESW must also be met (Section 8.2).

9.3.4 Only one channel with SRASW

If a programmed standard component is employed in one channel of the part of the control system in diverse redundancy with a technology other than programmable electronics (e.g. fluid power technology) in the other channel in category 3 or 4, the normative requirements can be reduced by one PL (e.g. the requirements for PL_r c apply in place of PL_r d, see Section 3.2), owing to the lower probability of a dangerous failure caused by systematic faults in this item of SRASW. In the case of category 2, the requirements can be lowered only for the SRASW of the test channel. Independently of this requirement, the normative requirements for the SRESW must also be met (Section 8.2).

9.4 Application of the IFA matrix method to standard components

The IFA matrix method described here can also be applied to application programming of standard components, provided the software structure corresponds to the three stages as shown in Figure 10, i.e. inputs stage ⇢ processing stage ⇢ outputs stage, and the measures for fault detection are implemented in the inputs and outputs stages. This applies all the more when two standard components are used in two channels with diverse application programming. A good example of this is provided by the validation example in DIN EN ISO 13849-2 [10], Annex E. Example projects illustrating this are also available (see Section 12.3).

9.5 Use of standard components for measures for fault control

Standard components are also frequently used to implement test equipment. What reliability is required in this case for these standard components: single-channel or two-channel? A general requirement for test equipment is that it should not fail significantly earlier than the components that it monitors. At the same time, it is inefficient for much greater investment to be made in the reliability of the test equipment than in the parts of the control system performing the safety function proper.

DIN EN ISO 13849-1 therefore imposes only limited requirements upon the reliability of the test equipment. For categories 3 and 4, reliance is upon single-fault tolerance, since several dangerous failures in total – including failure of the test equipment – must occur before the safety function ceases to be performed. It is considered extremely unlikely that such a case would occur unnoticed. Test facilities can therefore be implemented in single-channel form – with use of standard components such as a standard PLC – without any particular requirements concerning their reliability. Examples can be found in BGIA Report 2/2008 [2]: examples 22 (component K4), 28 (K3) and 33 (K1).

For category 2, a supplementary requirement exists – at least for the simplified procedure for determining the PL (DIN EN ISO 13849-1, Section 4.5.4) – that was imposed for calculation of the PFH_D: the rate of dangerous failure of the test equipment (e.g. of the standard PLC) must not exceed twice the rate of dangerous failure of the components that it monitors. In cases of doubt, this comparison can be carried out channel by channel. The $MTTF_D$ value of the entire test channel should then not be lower than half the $MTTF_D$ value of the functional channel. Here too, corresponding examples can be found in BGIA Report 2/2008: examples 9 (component K3), 11 (K1) and 12 (K1). With regard to SRASW, the normative requirements must be observed and the case studies in Section 9.3 considered: in the case of diverse SRASW (or SRASW in the test channel only), the requirements can be reduced by one PL_r.

Categories B and 1 do not require testing.

10 Typical test and monitoring measures in SRASW

In certified safety control systems and the manufacturer's function modules supplied with them, self-testing and online monitoring measures for fault control have already been implemented appropriately by the control system manufacturer. In contrast, this chapter provides information on standard components and function blocks developed by users for which fault-control measures cannot be implemented until the application programming stage.

The test and monitoring measures for random and systematic control system failures constitute an important part of the SRASW. Effective tests permit, for example, poor reliability of the components (including peripheral components) to be compensated for to some degree, thereby enabling the performance level of the safety function to be raised. The quality of fault detection is stated in DIN EN ISO 13849-1 by the diagnostic coverage (DC). Annex E of the standard contains a comprehensive list of typical test measures and the DC values that can be attained by them.

In contrast to the measures for fault avoidance (Chapter 5), these test and monitoring measures are also referred to as fault-control measures. Self-tests are often already implemented in the SRESW (embedded software) to manage failures of the programmable controller itself. Tests of the connected peripheral devices such as sensors or actuators can also be programmed in the SRASW. These tests are typically already implemented in the manufacturer's function blocks in the inputs and outputs stages (Section 6.17). In addition, project/machine-specific function blocks with tests and monitoring functionality must also be programmed in-house in the application. This chapter is intended to provide relevant information.

10.1 Typical techniques for testing and monitoring

Typical techniques exist for testing the connected peripheral devices on which certain behaviour is anticipated (e.g. due to a signal change), against which a check is performed with a different signal:

- Cross-comparison of two logic components: two logic components of a part of a control system (e.g. two standard PLCs) exchange their input signals and processing results and compare them.
- Logic and time-based program runtime monitoring: an item of test equipment checks whether all important program modules of the monitored control are executed in the expected time and sequence.
- Plausibility check of sensor components: a sensor signal (binary or analog) is compared with a second sensor signal from the same measuring point. An analog sensor signal is compared with an anticipated signal range.
- Direct monitoring: an actuator component (contactor/valve) is switched by an output signal from the logic stage. The switching operation is registered directly on the actuator component by a diagnostic component, such as an electrical contact, and reported back to the logic. After the discrepancy time has been allowed to elapse, the switching signal is compared with the check-back signal to monitor whether switching was performed correctly.
- Indirect monitoring: an actuator component (frequency inverter/contactor/valve) is switched by an output signal from the logic stage. The switching operation itself cannot be measured on the actuator component. Only the effect of switching is registered, by a diagnostic component on downstream parts of a control system, interconnecting means or actuators (e.g. pressure switches/rotary encoders/position switches/displacement transducers), and is reported back to the logic stage. After the discrepancy time has been allowed to elapse, the switching signal is compared with the check-back signal to monitor whether switching was successful.

Owing to time delays in the signals, a tolerance time (the discrepancy time) must always be allowed to elapse before the result of the comparison is valid. Analog signals require a value tolerance to be allowed for technical reasons.

10.2 Boundary conditions for test and monitoring measures

Detection of a dangerous failure is only the first part of a successful test: a safe state must also be brought about that presents no further danger. This includes an effective shut-off path, which for example in the case of single-channel tested systems (category 2) entails a requirement for a second shut-off element. This is necessary in order for the safe state to be brought about or maintained when the test has detected failure of the normal shut-off element. In two-channel systems (categories 3 and 4), the safe state is brought about by the second shut-off path when one of the channels fails.

Initiation of a test, its performance and the necessary shut-off process should ideally be performed automatically by the SRASW. Only in exceptional cases would it appear advisable to rely here upon manual intervention, for example by the machine operating personnel. Calculation of the diagnostic coverage for two-channel systems nonetheless takes account of fault detection when a demand is made for the safety function, i.e. consideration is not limited to tests triggered automatically in the SRASW. Electromechanical components in particular, such as relays or contactors, are generally capable of detecting the fault only when a demand is made for the safety function. For

the level of diagnostic coverage of fault detection in the event of a demand being made for the safety function, the frequency with which this occurs must be considered.

10.3 Test frequency

A further aspect is the question of the required test frequency. A test which is not executed sufficiently frequently may under certain circumstances be overtaken by the incidence of a hazardous event, and therefore gives a false sense of safety. As a rule of thumb, the test frequency is always in competition with the frequency of other events; for this reason, a universal adequate frequency cannot be stated. In two-channel category 3 and 4 systems, the test frequency is in competition with the frequency of incidence of a second dangerous failure, since a risk of the safety function not being executed exists only if the second channel fails before a test has detected the failure of the first channel. As per the definition, category 4 systems even tolerate an accumulation of undetected faults. In two-channel systems, a test frequency of once per shift has proved suitable in practice.

The situation is different for category 2 systems with single channel testing. In this case, the test must be passed before a demand is next made for the safety function, i.e. before a potential hazard occurs. In this case, the test frequency is therefore in competition with the frequency of the demand for the safety function. A factor of 100 is considered sufficient, i.e. a test frequency that is at least 100 times the mean demand rate for the safety function. Where a factor of 100 is not possible, the test rate can be reduced to a factor of 25. This reduction results in an increase in the probability of failure PFH_D of approx. 10% over that for a factor of 100 [21].

Should, in systems with single-channel testing, the tests be executed so quickly that the safe state is attained before a hazard arises, no requirements are imposed upon the frequency of testing (DIN EN ISO 13849-1:2016, Section 4.5.4, requirements for category 2).

10.4 Further information

Suggestions for testing measures can be found for example in the literature [8; 22 to 24].

11 Combinations of multiple parts of a control system with software

So far, this report has referred only to a programmable part of a control system (SRP/CS) with an application program, serving as part of a safety function. In some cases, several parts of a control system (e.g. PLCs and programmable drive controls), each with its own application program, may have to be connected in series as subsystems each executing part of a safety function. During evaluation of the control hardware, i.e. the probability of a dangerous, random component failure, the probabilities of failure (PFH_D) of the individual parts of the control system are added together. As a result, each additional part of the control system results in a decrease in hardware reliability.

What must then be considered with regard to SRASW and systematic failures when several programmed parts of a control system are combined for implementation of a safety function? Does this influence the requirements or the development process?

Section 6.3 of DIN EN ISO 13849-1 [1] addresses the combination of multiple parts of a control system, with the aim of determining an overall PL for the combination and thus for the attained safety function. At the same time, a total PFH_D is determined by adding the individual PFH_D values of the parts of the control system together. However, this sub-Chapter contains no explicit requirements or specifications for combination of application programs of these parts of a control system. The standard also states that: "*The PL of the combined SRP/CS is limited by:*
- the lowest PL of any individual SRP/CSi involved in performing the safety function (because the PL is determined also by non-quantifiable aspects) [such as software quality]".

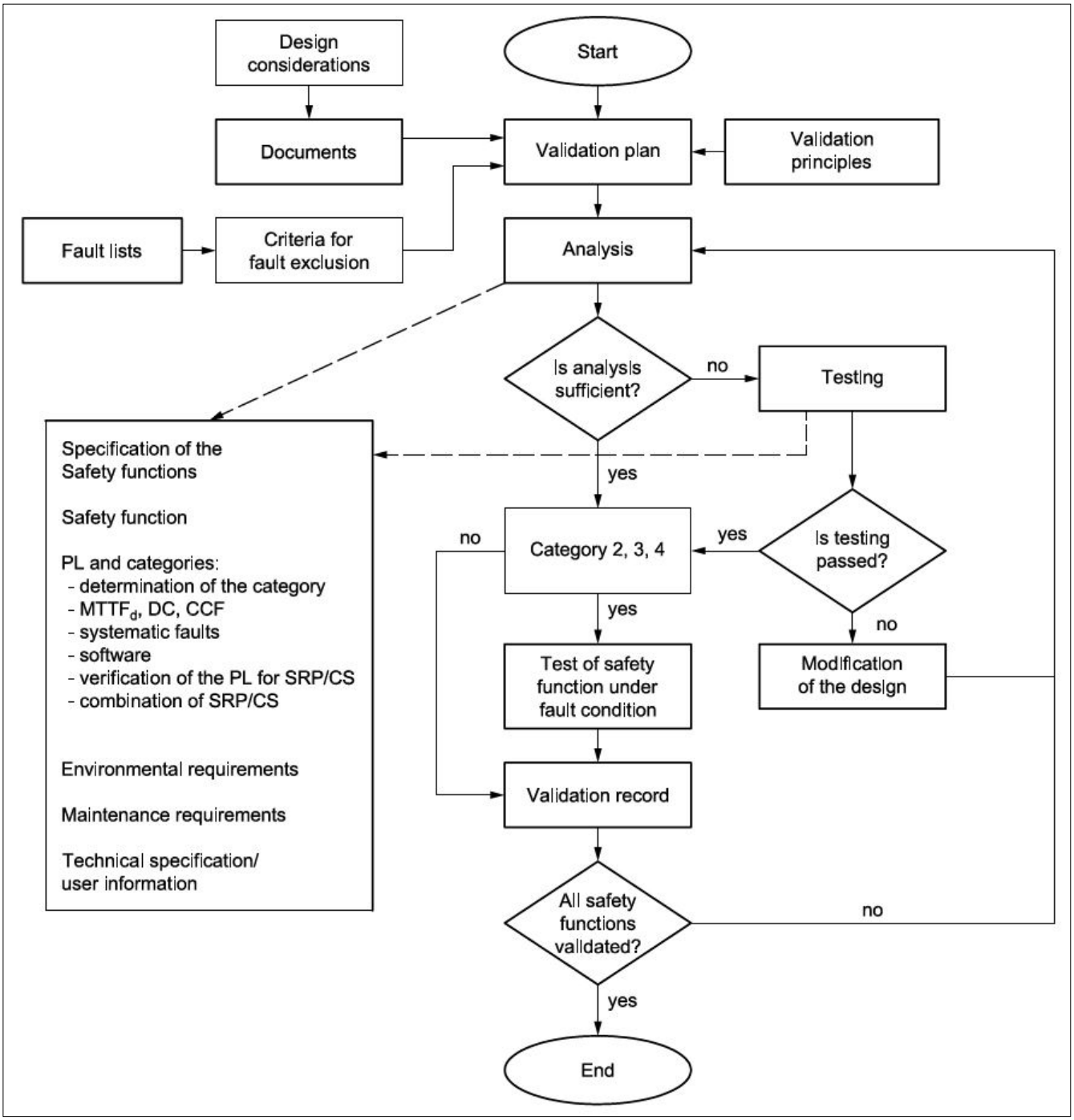
Consequently, the attainable PL for the combination/safety function as a whole is determined by the "poorest" application software, even when the total PFH_D is very low. This is also illustrated by Figure 3 in Section 3.2.

Conversely, this also means that specification of a PL_r for a safety function results in the same requirements to Section 4.6 of the standard applying to all application programs of the parts of the control system. Checking is of course required of whether connecting the parts of the control system and their application programs in series could increase the probability of systematic failures being caused by the programs owing to possible adverse interactions between them.

12 Validation of SRASW

Validation refers to quality assurance measures for the avoidance of errors during the design and implementation of safety-related parts of control systems (SRP/CSs) which perform safety functions. DIN EN ISO 13849-2:2013 [10] deals with this subject at particular length. Validation can be performed by analysis alone or by a combination of analysis and testing. Figure 59 provides an overview of the validation process.

Figure 59:
Overview of the validation process (to DIN EN ISO 13849-2:2013)



Validation is proof of suitability for the real-case intended purpose. It is performed during or at the end of the development process. It involves checking that specified safety requirements concerning the safety-related parts of the machine's control system are met. In relation to SRASW it involves, for example, checking of whether the safety functions have been attained by the SRASW a) as specified and b) in the quality required by the PL_r. For the categories 2, 3 and 4 (with fault detection), validation of the safety function must also include tests performed under fault conditions (expanded functional test, Section 5.10). Parts of the SRASW that detect and control faults are thus also tested.

12.1 General requirements for validation

In DIN EN ISO 13849-2:2013, general requirements are first formulated for all aspects of validation, some of these in relation to SRASW. These general requirements are described in brief below. Validation should be performed by persons who are not involved in designing the parts of a control system (see also Section 5.15).

When the IFA matrix method is used in conjunction with the SOFTEMA tool (Chapter 14), these general requirements are met. Further information, for example concerning ambient conditions or test equipment, can be linked to SOFTEMA in the form of documents.

12.1.1 Validation by analysis and tests

Validation of an item of SRASW comprises analysis (see DIN EN ISO 13849-2, Chapter 5) and performance of functional tests (see DIN EN ISO 13849-2, Chapter 6) under foreseeable conditions in accordance with the validation plan.

Chapter 5 of DIN EN ISO 13849-2 describes the general procedure and analysis techniques. It states that *validation of the SRP/CS must be carried out by an analysis* which must include i. a. the non-quantifiable, qualitative aspects which affect system behaviour (if applicable, software aspects).

During analysis it is to be determined by inspection of documents, for example by review or walk-through and where applicable by the use of analysis tools, such as tools for static and dynamic software analysis or FMEA tools, whether the specified requirements have been met.

The analysis should be begun as early as possible and conducted simultaneously with the development process, to enable problems to be corrected at an early stage. It may be necessary for performance of some parts of the analysis to be delayed until development has reached an advanced stage. Specific analysis steps relating to software are presented in Section 12.2 of the present report.

The general testing procedure and techniques employed are then described in Chapter 6 of DIN EN ISO 13849-2. Concerning the need for testing, it says that: *When validation by analysis is not conclusive, testing shall be carried out to complete the validation. Testing is always complementary to analysis and is often necessary.*

With regard to the number of test items, it states that *unless otherwise specified, the tests shall be made on a single production sample of the safety-related part being tested.* A single production sample is therefore sufficient for testing with respect to SRASW.

This is followed by a summary of Chapter 6 describing the requirements for the validation tests:

- Tests must be planned, executed in a logical sequence and Protocoled.
- The test plan must include the test specifications, the required outcome of the tests and the chronology of the tests.
- The test protocols produced must contain the names of the persons carrying out the tests, the environmental conditions, the test procedure and equipment used, the test date and the results of the tests.
- The test protocols must be compared with the test plan to ensure that the specified functional and performance targets are attained.
- The test must be performed on a test sample as close as possible to its final operating configuration, i.e. with all peripheral devices and covers attached.
- Testing may be applied manually or automatically (e.g. by computer).
- Where appropriate, the safety functions must be validated by means of tests in which various combinations of input signals are entered into the SRP/CSs. The resulting reaction at the outputs must be compared with the specified output signals.
- It is recommended that the combination of these input signals be input systematically into the control system and machine. An example of this logic is: switch on the power, start up, work sequence, change of direction, restart. If necessary, an expanded range of input data must be entered to take account of abnormal or unusual situations and for observance of the response of the SRP/CSs to them. Such combinations of input data must take account of foreseeable operating errors. The SOFTEMA tool supports documentation of these test cases.

12.1.2 Validation plan

Validation must be planned and documented in accordance with DIN EN ISO 13849-2. The documentation must present the following:

- The identity of the documents for the specifications
- The operating and ambient conditions during the tests
- The analyses and tests to be applied

- Reference to applicable test/inspection standards
- The persons or parties responsible for each step in the validation process

12.1.3 Information for validation

For validation of SRASW, the following documents are typically required in accordance with DIN EN ISO 13849-2:

- Specification of the required characteristics of each safety function and its required category and performance level
- Block diagram(s) with function description of the blocks
- Circuit diagrams including their links/connections
- Function description of the circuit diagrams
- Flow chart(s) for switching components and signals relevant to safety
- If relevant: user information, e.g. instructions for installation and operation/user manual

The software documentation must contain the following:

- A clear and unambiguous specification stating the safety performance to be attained by the software
- Proof that the software design attains the required performance level (see Section 12.2)
- Details of tests (in particular, test reports) performed to demonstrate attainment of the required safety performance

12.1.4 Protocol of validation

Validation by analysis and tests must be Protocoled. The Protocol must show the validation procedure for all safety-related requirements. Cross-references to previous, clearly marked Protocols are possible. Safety-related parts that have not passed validation must also be documented. It must be ensured that all parts pass validation again following a modification.

12.2 Special requirements for the validation of SRASW

Besides the general requirements, DIN EN ISO 13849-2, Section 9.5 also formulates special requirements for the validation of SRASW. These will be presented here, including with regard to how they can be implemented in the IFA matrix method with the SOFTEMA tool (Chapter 14). First of all, the scope of validation is shown in Table 24.

12.2.1 Documentation analysis

The first step (according to DIN EN ISO 13849-2) is to check *that there is a documentation for the specification and design of the safety-related software. This documentation shall be reviewed for completeness and absence of erroneous interpretations, omissions or inconsistencies.*

This is therefore the analysis step based on the documentation, and can be performed easily if an Excel file for the IFA matrix method is available. The functional characteristics of "completeness", "correctness", etc. can be analysed all the more quickly by the formal verification measures implemented in SOFTEMA.

Table 24:
Scope of validation of SRASW according to DIN EN ISO 13849-2:2013

Requirement according to DIN EN ISO 13849-2	Implementation steps in the IFA matrix method (documents)	Sections of the report
Specified function behaviour and performance criteria when executed on the target hardware	1) Verification of the C&E matrix (B4) against the specification of safety functions (A1)	6.3, 6.7
	2) Verification of the code against the C&E matrix (B4)	6.7
	3) Functional test of SRASW on target hardware against the C&E matrix (B4)	6.7
	4) Protocol of code review (C1) and validation (D1)	6.8
Adequate software measures for the specified PL_r of the safety function(s)	1) Listing of safety functions with PL_r and determining of the maximum PL_r (A1)	6.3
	2) Selection of software measures with reference to the PL_r in the catalogue of measures(A3)	6.5
	3) Verification of implementation of the measures during programming (also in A3)	6.5
	4) Protocol of code review (C1)	6.8
Measures and methods applied for the avoidance of systematic software faults during software development	1) Selection and application of measures (A3) and normative requirements (A4)	6.5
	2) Implementation and Protocoling of code review (C1) and validation (D1)	6.8

In deviation from this, the standard notes that:

NOTE In the case of small programs, an analysis of the program by means of reviews or walk-through of the control flow, procedures, etc. using the software documentation (control flow chart, source code of modules or blocks, I/O and variable allocation lists, cross-reference lists) can be sufficient.

For small programs, the software documentation provided by the programming environment may therefore suffice in place of documentation of a software design (Figure 6), provided the control flow is made comprehensible by suitable diagrams. Unfortunately, the "small" property is not specified.

12.2.2 Software test

If the analysis is not sufficient for validation, black-box tests[1] of the SRASW are carried out on the target hardware. The tests and the Protocols made of them should encompass the requirements in Table 25 according to the PL_r and the category. Does every item of SRASW that has already been validated need to be re-tested? The standard states that *individual software functions [program modules; function blocks; functions] which have already been validated do not need to be validated again. Where a number of such function safety blocks are combined for a specific project, however, the resulting total safety function shall be validated.*

1 A black box test is a test of the software's dynamic behaviour under real-case functional conditions. It detects deviations from the software specification. No knowledge of the internal structure of the software is required for performance of the test. Source: DIN EN 61508-7:2011 [3]

Rephrased with respect to the IFA matrix method: even if all function blocks of the pre-processing and actuator control stages are certified/validated, it is always necessary to validate the logic of the ACT module and the entire safety function of the program.

Should the safety-related software subsequently be modified, it must be revalidated on an appropriate scale. This is also implemented suitably by means of the matrix method and SOFTEMA.

12.3 Validation example from DIN EN ISO 13849-2, Annex E

Annex E of DIN EN ISO 13849-2 [10] provides an example of validation of an SRP/CS. However, it does not describe software validation. For this example in the standard, the present report therefore presents a possible approach with reference to the IFA matrix method.

The notable aspect of this example is that two standard PLCs are used. Accordingly, specification and documentation are performed twice, in a separate Excel file for each PLC (PLC A and PLC B). The specification of the safety functions (document A1) is identical.

The two Excel documents for this example can be found in a separate archive file in the download area of this IFA report. The Excel files can also be opened and viewed by means of the SOFTEMA tool (Chapter 14).

Table 25:
Requirements for tests and protocols

PL/category	Test measure to DIN EN ISO 13849-2	Implementation in the IFA matrix method (documents)
All PL_r	Black box test of functional behaviour and performance, e.g. dynamic behaviour	1) Test plan of the C&E matrix (B4) with rows for safety functions; with validation column and name/date 2) Protocol of validation (D1)
Recommended for PL_r d or e	Additional expanded test cases based on limit value analyses	1) Test plan of the C&E matrix (B4) with additional test cases; with validation columns and name/date 2) Protocol of validation (D1)
All PL_r	I/O tests to ensure that the safety-related input and output signals are used correctly	1) I/O list (A2.4) with validation columns and name/date 2) Protocol of validation (D1)
PL_r and categories with fault detection	Test cases simulating faults that are determined analytically beforehand, together with the anticipated response, to assess the suitability of software-based fault control measures	1) Test plan of the C&E matrix (B4) with additional test cases; with validation columns and name/date 2) Protocol of validation (D1)

13 Technical documentation and user information

Chapter 10 and 11 of DIN EN ISO 13849-1 indicate what information is to be documented, and what parts of this information are to be passed on for use of the control system (generally to the machine operator).

13.1 Technical documentation

Before the manufacturer issues the EC declaration of conformity for a machine, he must draw up its technical documentation. With regard to SRASW, certain keywords are relevant in Chapter 10 of the standard, and are referred to in brackets below. Firstly, the specification of the implemented safety functions is required, together with the various design documents and the well-commented program (*"characteristics of each safety function", "performance level", "design rationale", "software documentation"*). In addition, the certified or internally validated library functions used must be listed together with their identifying information (version number, author, date, etc.). Application of programming guidelines and language subsets must also be documented (*"measures against systematic failure"*). Should these already be covered by the programming tool, reference to these properties is sufficient. Finally, the test activities must be documented. The integration test and validation of the safety functions are often performed together. These tests must obviously be planned, and documented together with the test results. When the IFA matrix method is used, all this information required by Chapter 10 of the standard for the technical documentation is already available. As a note, the standard states that *in general, this documentation is foreseen as being for the manufacturer's internal purposes and will not be distributed to the machine user.* Should the operator wish or need to perform software modifications themselves and therefore be reliant on this documentation, alternative arrangements for this must be made in the contract.

Modern software documentation also includes configuration management. For safety-related software in particular, it is obvious and therefore a requirement that its development should be comprehensible to all parties involved and for reviews at a later stage:

- Who specified, programmed, commissioned, verified and validated it, and when?
- How was development performed, e.g. with what tools and their settings, re-used functions and their identifying information, programming guidelines?
- Which program versions are loaded in what controls?

This and other necessary information, including all relevant development documents, must be retained and suitably archived for later use, for example for the event of modification after five years in operation. At the same time, all tools used should be archived in the version concerned, to enable documents to be read even many years later.

13.2 User information

What is to be understood by information for use according to Chapter 11 of DIN EN ISO 13849-1? No specific requirements are formulated for SRASW. A general requirement is stated that *in particular, information which is important for the safe use of the SRP/CS shall be given to the user.*

14 SOFTEMA software tool for developing and testing SRASW

In order for the IFA matrix method to be used efficiently and with assured quality, the IFA developed the SOFTEMA software tool (refer to the project information page of IFA Project 5137: https://www.dguv.de/ifa/forschung/projektverzeichnis/ifa5137.jsp. Like the IFA's SISTEMA [15] tool, SOFTEMA is available for download free of charge. This chapter provides an overview of the tool's core features and functions. Further information and guidance for users are available separately on the SOFTEMA download site (see Section 14.4).

14.1 What is SOFTEMA capable of?

The examples available for download in the form of Microsoft Excel files (Chapter 7 and Section 12.3) can be viewed in SOFTEMA. Users can also use SOFTEMA to create and edit their own projects. SOFTEMA opens a project-specific file for specification and documentation of an application program. Multiple instances of the software can be opened, enabling multiple projects and programs to be viewed and worked on simultaneously. This enables project data to be copied and pasted between different instances of SOFTEMA (or instances of Excel) via the clipboard.

SOFTEMA project files use the "Microsoft Excel workbook (*.xlsx)" file type. Note that the legacy "*.xls" Excel file type cannot be opened in SOFTEMA. One reason is that this file type supports only 256 table columns. The project files can be edited either with SOFTEMA or directly in Microsoft Excel. All tables can be edited freely in Excel. In SOFTEMA, the content is write-protected by the user management function. The specialized SOFTEMA functions described below are available only in SOFTEMA. The project files do not contain macros. All SOFTEMA functions are integrated and protected in the software. In Excel, however, additional worksheets can be added and used for development and documentation, for example for documentation of the control hardware. SOFTEMA ignores these tables, and cannot load or display them (in the first release). SOFTEMA supports further functions in addition to use of the matrix method:

- Automatic updating of tables following modification of input data
- Formal verification of tables (for missing, conflicting or double entries)
- Management of project members
- Role-based user permissions
- Support during verification, validation and testing
- Support with modifications
- Dedicated editors for the different forms of cell content
- Management of documents and changes
- Search/replace functions
- Specific print functions and reports
- Automatic Protocoling of changes to particularly safety-critical cell content

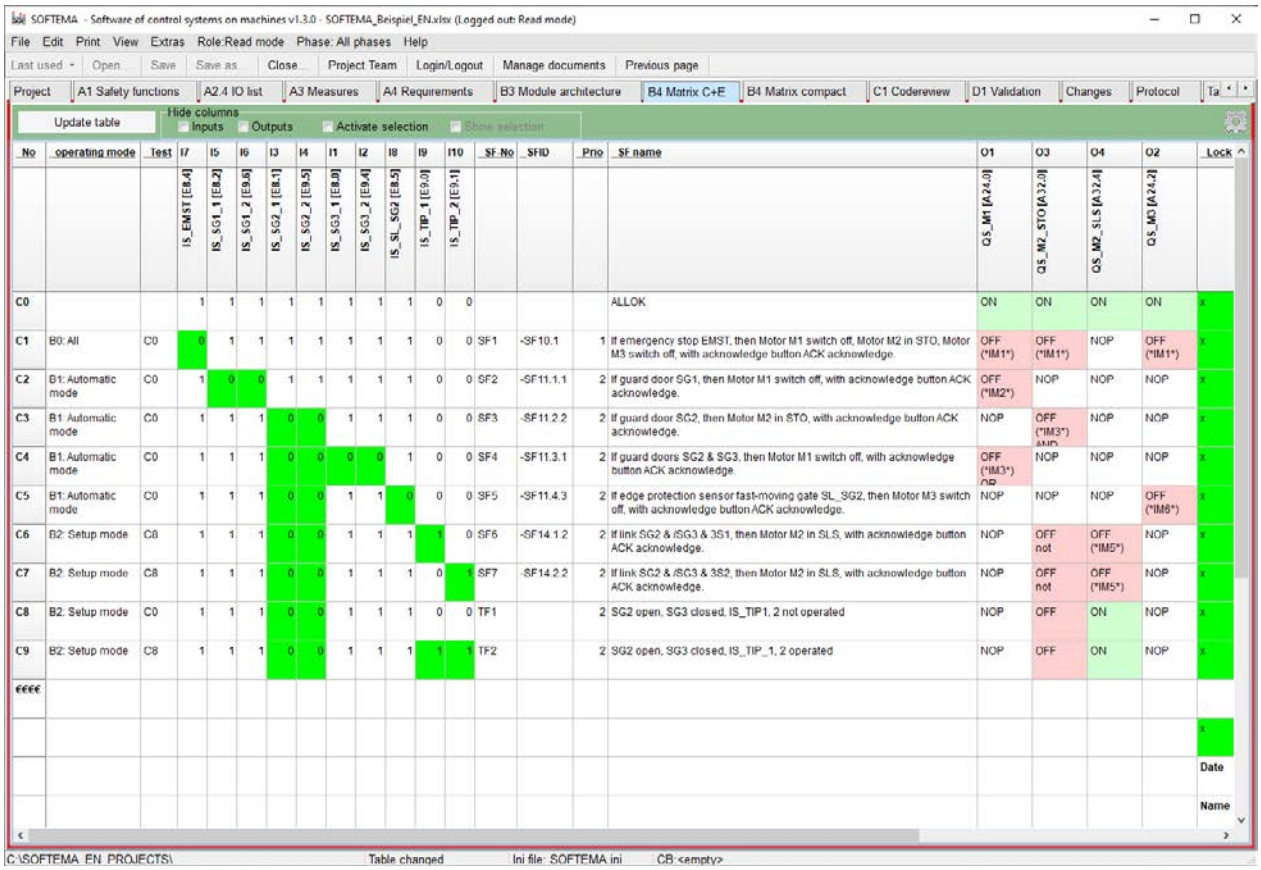
14.2 How is SOFTEMA used?

SOFTEMA manages the tables required for the IFA matrix method and beyond that the information required for project management, such as project description, user management, change logs, document management, etc. Figure 60 shows, for example, the C&E matrix of a project in SOFTEMA.

For a new project, the user opens an empty but preformatted project template. Once the project description ("Project" table) has been completed, safety functions and their properties such as the PL_r, operating mode, priority, etc. are entered in Table A1, "Safety functions". The input and output signals and their respective variable names and hardware/network addresses are entered in Table A2.4, "I/O list". External content can also be copied and pasted into all tables via the clipboard. The catalogue of measures for fault avoidance and the programming rules can be selected and adjusted in Table A3, "Measures". Tables A3, "Measures", and A4, "Requirements", should already be populated with data in advance in the project template. The safety functions, the peripheral hardware and the I/O list result in a list of the required function blocks for the inputs and outputs stages. These should be managed in Table B3, "Modular architecture". In contrast to the presentations of Document B3 in this report, SOFTEMA manages the function modules in a list. Following these preparations, Table B4, "Matrix C+E", can be completed. The buttons for automatic updating for I/O signals and safety functions are used for this purpose. Software specification proper is then performed by:

- Assignment of input signals to the discrete safety functions
- Entry of logical operations linking the signals for the switching operations to the output signals.

Figure 60:
C&E matrix in SOFTEMA



This is required for programming of the actuation logic. A dedicated editor assists in creation of this logic. The compact presentation in Table B4, “Matrix compact”, provides support for comprehensive projects. This table is created solely with the update function, which automatically converts table B4, “Matrix C+E”. At this point at the latest, all available functions for formal verification of the tables referred to must have been used, in order for omissions, duplications and contradictions to be detected and corrected.

Following verification of all input documents and the specification described above, the program can be written. The code is also verified. This process is documented in detail in a number of tables and is also summarized in Table C1, “Code review”. The program is then validated. This, too, is documented in detail in a number of tables and summarized in Table D1, “Validation”. The questions can be adapted and added to if needed in Tables C1 and D1. Persons subsequently reviewing the project can likewise document and annotate their activity.

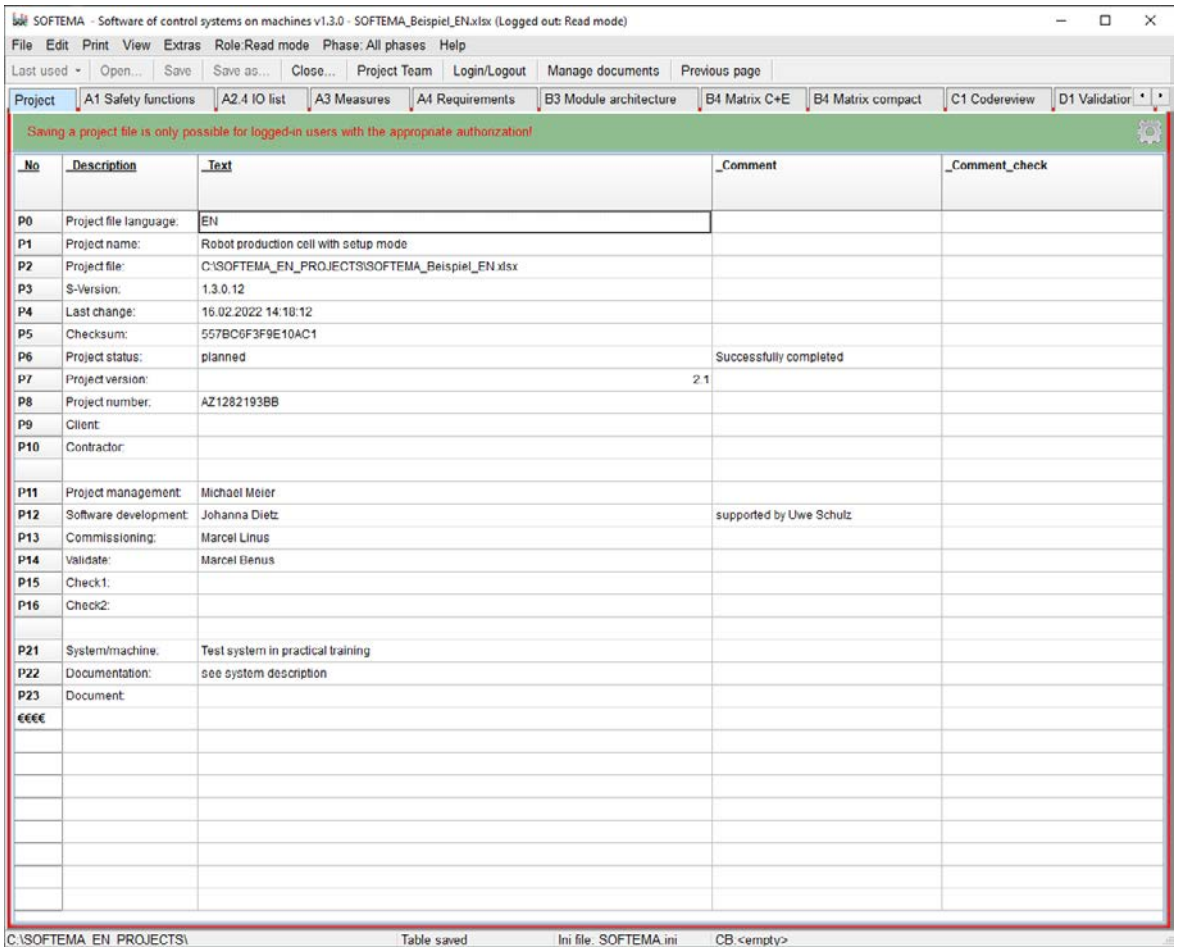
When the safety functions or the I/O signals are modified, the changes in tables A1 and A2.4 are in turn automatically updated in the specification tables, and edited by the user. All modifications are initially highlighted in colour (yellow). The highlighting is deleted manually when repeat programming, verification and validation of these modifications has been completed.

14.3 SOFTEMA user interface

SOFTEMA runs on the Microsoft Windows 7, Windows 8 or Windows 10 operating systems. It uses conventional menu technology with a fixed toolbar for the most important commands. Further information is displayed in the title bar (top) and status bar (bottom).

The SOFTEMA application interface (Figure 61) is characterized by intuitive usability, as it closely follows the concepts and use of MS Excel. Most of the application interface is taken up by the workspace – a table – in the centre (see Figure 61). SOFTEMA manages all entries in tables that can be selected on the tabs at the top of the workspace. Each table corresponds to a worksheet in the Excel file. Above the table in each tab is an area containing table functions that can be called up by means of buttons, selection lists or radio buttons.

Figure 61:
SOFTEMA application interface, presentation of the project description



14.4 Where can SOFTEMA be obtained?

The SOFTEMA tool has been available on the IFA website since 2017. It can be used free of charge following registration. Current information on the state of development and the link to the download are available at: https://www.dguv.de/ifa/praxishilfen/praxishilfen-maschinenschutz/software-softema/index.jsp. Please note the liability disclaimer and licence information.

14.5 How is SOFTEMA installed and executed?

SOFTEMA is supplied with an installer. For the time being, the program is available only with a German user interface. Although the project files are in Microsoft Excel file format (extension *.xlsx), Microsoft Excel need not be installed. It is, however, very useful for occasionally editing the project files directly.

15 Literature

[1] DIN EN ISO 13849-1: Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design (12/2008), and Amendment 1 of DIN EN ISO 13849-1 (2016) Beuth, Berlin 2008/2016

[2] *Hauke, M.; Schaefer, M.; Apfeld, R.; Bömer, T.; Huelke, M.* et al.: Functional safety of machine controls – Application of EN ISO 13849 – (BGIA-Report 2/2008e). Published by.: Deutsche Gesetzliche Unfallversicherung (DGUV), Berlin 2008 (currently undergoing revision)

[3] DIN EN 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems – Parts 1 to 7 (all 02/2011). Beuth, Berlin 2011

[4] DIN EN 954-1: Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design (3/1997). Beuth, Berlin 1997 (no longer valid)

[5] Norm compliant development and documentation of safety related application software in manufacturing system engineering. Project No FF-FP0319. Published by: Deutsche Gesetzliche Unfallversicherung (DGUV), Berlin 2014. https://www.dguv.de/ifa/forschung/projektverzeichnis/ff-fp_0319-2.jsp

[6] DIN EN 62061: Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems (09/2013). Beuth, Berlin 2013

[7] VDI/VDE-Richtlinie: Applikationsprogrammierung von Sicherheitsfunktionen für Maschinenbau und Fertigungstechnik (draft). GMA AK1.50 Methoden der Steuerungstechnik

[8] PLCopen – Technical Committee 5 – Safety Software Technical Specification, Part 1: Concepts and Function Blocks Version 1.0 – Official Release, 2006

[9] *Becker, N.; Eggeling, M.; Huelke, M.*: SPS-Software für fehlersichere Steuerungen – Normgerecht entwickeln und dokumentieren. atp edition – Automatisierungstechnische Praxis 57 (2015) No 4, pp. 34-47

[10] DIN EN ISO 13849-2: Safety of machinery – Safety-related parts of control systems – Part 2: Validation (2/2013). Beuth, Berlin 2013

[11] DIN ISO/TR 23849: Guidance on the application of ISO 13849-1 and IEC 62061 in the design of safety-related control systems for machinery (12/2014). Beuth, Berlin 2014

[12] DIN EN 61131-3: Programmable controllers – Part 3: Programming languages (IEC 61131-3:2013) (06/2014). Beuth, Berlin 2014

[13] DIN EN ISO 12100: Safety of machinery – General principles for design – Risk assessment and risk reduction (3/2011). Beuth, Berlin 2011

[14] Definition of the safety functions: what is important? (SISTEMA Cookbook 6). Published by: Deutsche Gesetzliche Unfallversicherung (DGUV), Berlin 2015. https://www.dguv.de/ifa/praxishilfen/practical-solutions-machine-safety/software-sistema/sistema-kochbuecher/index.jsp

[15] Software-Assistent SISTEMA. Published by: Deutsche Gesetzliche Unfallversicherung (DGUV), Berlin 2015. https://www.dguv.de/ifa/praxishilfen/practical-solutions-machine-safety/software-sistema/sistema-kochbuecher/index.jsp

[16] *Boehm, B.*: Guidelines for verifying and validating software requirements and design specifications. In: *Samet, P. A.* (Eds.): Euro IFIP: Proceedings of the European Conference on Applied Information Technology of the International Federation for Information Processing, London, 25-28 September 1979, pp. 711-719. North-Holland, Amsterdam 1979

[17] *Becker, N.; Eggeling, M.*: DGUV project final report Nr. FF-FP0319. Published by: Deutsche Gesetzliche Unfallversicherung (DGUV), Berlin 2013. https://www.dguv.de/ifa/forschung/projektverzeichnis/ff-fp_0319-2.jsp

[18] DIN EN 61508-3: Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements (2/2011). Beuth, Berlin 2011

[19] *Barg, J.; Eisenhut-Fuchsberger, F.; Orth, A.; Ost, J.; Springhorn, C.*: 10 Schritte zum performance level: Handbuch zur Umsetzung der funktionalen Sicherheit nach ISO 13849. Published by: Bosch Rexroth, Würzburg 2011

[20] *Bömer, T.; Schaefer, M.*: Unterschiede bei der Verwendung von fertigen Sicherheitsbauteilen und Standardbauteilen für die Realisierung von Sicherheitsfunktionen an Maschinen. Published by: Institut für Arbeitsschutz der Deutschen Gesetzlichen Unfallversicherung (IFA), Sankt Augustin 2011

[21] When the designated architectures don't match (The SISTEMA Cookbook 4). Published by: Deutsche Gesetzliche Unfallversicherung (DGUV), Berlin 2012. https://www.dguv.de/ifa/praxishilfen/practical-solutions-machine-safety/software-sistema/sistema-kochbuecher/index.jsp

[22] *Mai, M.; Reuß, G.*: Self-tests for microprocessors incorporating safety functions or: “Quo vadis, fault? (BGIA-Report 2/2007e). Published by: Hauptverband der gewerblichen Berufsgenossenschaften (HVBG), Sankt Augustin 2006. https://www.dguv.de/ifa/publikationen/reports-download/bgia-reports-2005-bis-2006/bgia-report-7-2006/index-2.jsp

[23] *Ostermann, B.*: Entwickeln und Bewerten Fehler erkennender Programmbausteine in speicherprogrammierbaren Steuerungen (SPS) zur Erhöhung deren Sicherheit. Degree thesis. Fachhochschule Bonn-Rhein-Sieg, Sankt Augustin 2006. www.maschinenbautage.eu/index.php?id=289

[24] *Gall, H.; Kemp, K.*: Wirksamkeit von zeitlichen und logischen Programmlaufüberwachungen beim Betrieb von Rechnersystemen. – Grundlagen der Rechnersicherheit – Leitlinien und Programm für den Einsatz von Programmlaufüberwachungen. Schriftenreihe der Bundesanstalt für Arbeitsschutz und Arbeitsmedizin: Forschungsbericht, Fb 772. Hrsg: Bundesanstalt für Arbeitsschutz und Arbeitsmedizin (BAuA), Dortmund 1997. Bremerhaven, Wirtschaftsverlag NW Verlag für neue Wissenschaft 1997

16 List of abbreviations

Abbreviation	Description
ACT	actuation (processing stage)
BGIA	Berufsgenossenschaftliches Institut für Arbeitsschutz (now: IFA)
CAE	computer aided engineering
C&E matrix	cause and effect matrix; synonym: cause and effect table
CPU	central processing unit
EL	Equipment labelling
DC/DC_{avg}	average diagnostic coverage
EMC	electromagnetic compatibility
FBD	function block diagram [18] (PLC language)
FMEA	failure mode and effect analysis
FI	frequency inverter
FVL	full variability language
IFA	Institute for Occupational Health and Safety of the German Social Accident Insurance (Institut für Arbeitsschutz der Deutschen Gesetzlichen Unfallversicherung)
I/O	input/output (e.g. of a PLC)
KAN	Commission for Occupational Health and Safety and Standardization (Kommission Arbeitschutz und Normung)
LD	ladder diagram (PLC language)
LVL	limited variability language
$MTTF_D$	mean time to dangerous failure
NOP	no operation (command in the C&E matrix that has no effect)
PFH_D	probability of a dangerous failure per hour
PL	performance level
PL_r	required performance level
PLC	programmable logic controller
SF	safety function
SFI	safety function identifier
SIL	safety integrity level
SISTEMA	IFA software assistant: Safety Integrity Software Tool for the Evaluation of Machine Applications
SOFTEMA	IFA software assistant: safe software on machines
PLC(s)	programmable logic controller(s)
SRASW	safety-related application software
SRESW	safety-related embedded software
SRP/CS	safety related part of a control system
SPLC(s)	safety programmable logic controller(s)
TÜV	Technischer Überwachungsverein
VDMA	Verband Deutscher Maschinen- und Anlagenbauer