TAMPERE UNIVERSITY OF TECHNOLOGY

VELU VARJORANTA

SOFTWARE SAFETY ISSUES IN MACHINE CONTROL SYSTEM DESIGN PROCESSES

Master of Science Thesis

Examiner: Prof. Jose Martinez Lastra
Examiner and topic approved at the Council Meeting of the Faculty of Automation, Mechanical and materials Engineering on October 5th 2011

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
VARJORANTA, VELU: Software safety issues in machine control system design processes
Master of Science Thesis, 75 pages, 2 Appendix pages
March 2012
Major: Factory automation
Examiner: Prof. Jose Martinez Lastra
Keywords: IEC 61508, EN 62061, safety, PLC, control systems, SIL, PL, certification

The growth of reliability of PLCs has widened their affordance in different industrial machinery. In the same time as the amount of automation in various machinery increases, the substance of safety equipment reliability has grown in the absence of human control. These factors have led to an increase in the demand for safety-related PLCs. The complexity and size of the software-based control applications grows and so does the importance to be able to produce safe software code. The requirements for machine safety posed by the national and international laws are intricate and many industrial standards have been formed to facilitate designing machinery that meet these requirements. This thesis concentrates on these standards and the recommendation and requirements they pose.

This thesis handles international standards EN 954, EN 50128, EN 62061, IEC 61131, IEC 61508, ISO 12100, ISO 13849 and ISO 14121. Their scopes and differences are discussed in Chapter 2. Terms and methods for defining various areas and levels of safety integrity are explained in the third chapter. Chapter 4 handles with the effects of operating systems as a base for safety-critical applications and introduces two safety-certified operating systems based on virtualization layers for machine automation control systems. Communication buses used in many automated machineries and their safety modifications are discussed briefly in Chapter 5.

Chapter 6 digs deeper into the effects of the requirements on safety standards on designing software-based machine automation control systems through process models, architectural methods and coding rules. In Chapter 7 two safety-certified integrated development environments for developing safety-related software are introduced. Process models and safety requirements handled in the thesis are applied in a theoretical example project in Chapter eight.

Through charting various safety-related requirements it becomes clear that compared to non-safety-related projects, safety-related projects require many times more documentation on made design choices, analysis and implementations. The requirements for this large amount of documentation are based on the need to be able to provide evidence on sufficient rigour and comprehensiveness of the made safety analyses and implementation quality. To be able to control all this information this thesis recommends forming information databases for companies designing such machinery. These databases would facilitate finding and updating all the relevant information from safety standards to project-specific documents in one place.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
VARJORANTA, VELU: Turvallisuus ohjelmistopohjaisten koneautomaatiojärjestelmien suunnitteluprosesseissa
Diplomityö, 75 sivua, 2 liitesivua
Maaliskuu 2012
Pääaine: Factory Automation
Tarkastaja: professori Jose Martinez Lastra
Avainsanat: IEC 61508, EN 62061, turvallisuus, PLC, ohjausjärjestelmät, SIL, PL, sertifiointi

PLC-laitteiden luotettavuuden kasvu on laajentanut niiden käyttömahdollisuuksia erilaisissa teollisuuden laitteissa samalla kun automaation määrä tehdasympäristöissä jatkaa kasvuaan. Ihmisohjauksen puuttuessa on automaattisten turvalaitteistojen luotettavuuden merkitys suurentunut, mikä on johtanut turvakriittisten PLC-laitteiden kysynnän kasvuun. Laitteiden ohjausjärjestelmät monimutkaistuvat ja tarve turvallisen ohjelmistokoodin tuottamiselle kasvattaa merkitystään. Lain vaatimukset koneiden turvallisuudelle ovat monisyiset ja niiden saavuttamisen helpottamiseksi on tuotettu teollisia standardeja, joiden vaatimuksiin ja suosituksiin tämä diplomityö keskittyy.

Tämä diplomityö käsittelee standardeja EN 954, EN 50128, EN 62061, IEC 61131, IEC 61508, ISO 12100, ISO 13849 ja ISO 14121. Niiden aihealueet ja erot keskenään käsitellään kappaleessa kaksi. Standardien käyttämiä eri turvallisuuden alueiden ja turvallisuustasojen määrittämiseksi käyttämiä termejä ja menetelmiä selitetään kappaleessa kolme. Kappale neljä käsittelee käyttöympäristöjen vaikutusta turvakriittisten ohjelmistojen alustana ja esittelee samalla kaksi virtuaaliympäristöihin perustuvaa turvasertifioitua käyttöjärjestelmää koneautomaatiojärjestelmiin. Koneautomaatiojärjestelmissä käytettyjen väyläratkaisuiden erilaisia turvaversioita käsitellään lyhyesti kappaleessa viisi.

Kappale kuusi käsittelee tarkemmin standardien vaatimusten vaikutusta ohjelmistopohjaisten koneautomaatiojärjestelmien toteuttamiseen prosessimallien, arkkitehtuuristen menetelmien ja ohjelmointiohjeiden muodossa. Kappaleessa seitsemän esitellään kaksi turvaluokiteltua ohjelmointiympäristöä turvakriittisten ohjelmistojen tuottamiseksi koneenohjausjärjestelmiin ja arvioidaan kehityksen suuntaa lähitulevaisuudessa. Työssä käsiteltyjä prosessimalleja ja turvavaatimuksia sovelletaan pintapuolisesti teoreettisessa käytännön esimerkissä kappaleessa kahdeksan.

Turvavaatimusten kartoittamisesta käy ilmi että verrattuna projekteihin, joissa turvallisuuteen ei kiinnitetä yhtä suurta huomiota, vaatii turvakriittisen ohjausjärjestelmän suunnittelu moninkertaisesti erilaista dokumentaatiota tehdyistä analyyseistä, suunnitelmista ja toteutuksista. Vaatimuksen perustana on varmistaa riittävät todisteet kattavasta riskianalyysien ja turvatoimintojen toteuttamisen tarkkuudesta ja laajuudesta erilaisten vaaratilanteiden välttämiseksi. Tämän suuren tietomäärän hallitsemiseksi tämä diplomityö suosittelee muodostettavaksi yrityksille omia selkeitä tietokantoja tarvittaville ja itse tuotettaville dokumenteille, jotta kaikki projektiin osallistuvien olisi helppo ylläpitää ja päästä käsiksi kaikkeen tarvittavaan aineistoon standardeista ja toimintaohjeista projektikohtaisiin dokumentteihin asti.

PREFACE

The topic for this thesis was provided by CrossControl Oy. It was done as a part of FIMA FAMOUS work package 3 which concentrates on researching safety concepts and software design and safety concepts. The examiner for this thesis was professor Jose Martinez Lastra from Tampere University of Technology. Supervisors were M.Sc. Marko Elo from CrossControl Oy and researcher Jani Jokinen from Tampere University of Technology.

I would like to thank Marko for providing me this topic and helping me to delimit it. This topic has given me new ways of perceiving machine safety and user operated machinery. The FAMOUS project has increased my knowledge on different industry branches and their needs in the field of system safety. The project has also provided me with a lot of safety-related material, for which I would like to thank all the project attendees, especially Timo Malm from VTT. Thanks go also to Jani Jokinen for supervising the writing process and helping me with all the bureaucratic issues. I also want to thank my girlfriend for coping with me working weekdays and writing this thesis all the weekends. In addition, I want to express my gratitude for my parents for the support they have provided me during my studies in the university.

Tampere, March 11th, 2012

Velu Varjoranta

CONTENTS

TERMS AND DEFINITIONS

API	Application programming interface
CAN	Controller Area Network
CCF	Common Cause Failure
CPU	Central processing unit
CRC	Cyclic redundancy check
DC	Diagnostic Coverage
E/E/PE	Electrical, electronic or programmable electronic
EN	European standard adopted by CEN, CENELEC or ETSI
EUC	Equipment under control
Failure	Termination of the ability of a SRECS, a subsystem, or a subsystem element to perform a required function
FMEA	Failure mode and effect analysis
FTA	Failure tree analysis
FVL	Full variable language
Hazard	Potential source of physical injury or damage to health
HAZOP	Hazard and Operability Studies
I/O	Input/Output. Inputs are signal received by a system and outputs present transmitted signals.
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
LVL	Limited variable language
MTTF	Mean time to hazardous failure
PFH	Probability of Failure per Hour
PHA	Preliminary Hazard Analysis
PL	Performance level
PLC	Programmable logic controller
OS	Operating system.
Redundancy	Existence of more than one means for performing a function
RTE	Run time environment
RTOS	Real time operating system
SDE	Software development environment
SIL	Safety integrity level
SRECS	Safety-related electrical control system
SRS	Safety Requirements Specification
SWFMEA	Software Failure Mode Analysis
SWSIL	Software Safety Integrity Level
TUT	Tampere University of Technology
VTT	Technical Research Centre of Finland

1 INTRODUCTION

A few decades ago all machine automation systems were relay-based. This kind of control systems included large amounts of relays which were relatively large, heavy and easily broken. Modifications required lots of design effort and the sheer size of the relay groups made mobile machinery heavy and large. Programmable logic controllers (PLC) changed the automation industry being light and easily reprogrammable units which were connected to the unit's sensors and actuators. However the reliability of the first generations of PLCs hasn't been found reliable enough for implementing safety applications. Therefore many of the safety-related functions are still preferably implemented with special safety certified hardware units external to the PLC control units. During the recent years safety certified PLC units have become available in the markets. The PLCs hardware is duplicated for enhanced reliability and the software applications have different requirements compared to applications used in non-safe PLCs. This thesis approaches the topic of safety-related projects implementing safety certifiable software through the guides and requirements set by industrial safety-related standards to meet the requirement of the law.

Some basic terms should be clarified before proceeding further to the topic. The first term is "safety". What actually is safety? International safety framework standard IEC 61508 describes safety as "freedom from unacceptable risk" [34]. The system may contain several risks, but still be considered safe. Risk is explained by another safety-related standard ISO 12100-1 as "combination of the probability of occurrence of harm and the severity of that harm" [29]. The term is further explained in Chapter 3. The risks are mitigated primarily through mechanical design, secondarily through safety equipment (light curtains, walls and signs) and safety functions executed by safety PLCs. These safety functions are the methods for the control system to achieve or maintain a safe state in the system in case of a system failure or dangerous user action. These safety functions can be for example actions after pressing an emergency stop button or triggering a light beam of a safety curtain, preventing an unexpected engine start, anti-sway controls on harbour cranes, reducing working speed of robots while a user is detected in the work area or anti-collision system in autonomous moving machines.

Safety should not be confused with security. Security is concerned when protecting the system from intentional attacks towards its functionality or information confidentiality. Safety is concerned when avoiding inflicting harm to humans directly or indirectly through the environment.

1.1 Background and problem

This thesis works as a part of nation-wide research project FIMA FAMOUS. FIMA – Forum for Intelligent Machines is a network for mobile work machine manufacturers, specialist companies, system integrators and research institutes. The FAMOUS project (Future Semi-Autonomous Machines for Safe and Efficient Worksites) concentrates on safety issues on mobile work machines. The thesis acts as a generic introduction to the issues related in designing safety-related machine automation systems with safe PLCs safety certification as a goal. The topic is large and may be hard for a system designer to quickly get grasp of all the issues related.

1.2 Research on topic

This topic is widely researched in various projects around the world. Following is a few researches considered in this thesis.

AMOS
Agile Development Approach for Safety and Reliable Systems – AMOS – is a project launched by pan-European ITEA2 programme. Aim of the project is to reduce cost and effort in building safety-related and reliable systems through integration of agile methods with safety methods.

FIMA FAMOUS
As mentioned in Chapter 1.1, FIMA FAMOUS concentrates on different branches of mobile machinery from the aspects of autonomous, semi-autonomous and remote-operated machines. Discussed topics have been for example situation awareness and fleet command.

KOTOTU
Koneiden ohjausjärjestelmien toiminnallinen turvallisuus, KOTOTU, was a project by VTT in which internet-based process tools were developed for identifying and handling safety hazards in an early phase of the project. Also a calculation tool for determining system performance levels was introduced.

OHJELMATURVA
VTT (Technical research Centre of Finland) finished its software safety research project “Ohjelmaturva” [23] (Safety-critical software in machinery) during the working process of this thesis. The program concentrated mostly on the same software issues handled in this thesis.

PSAFECER
ARTEMIS is a European research initiative for embedded systems, where one of the research projects, pSafeCer [43], concentrates on certification of software components as an aim decrease the cost and effort required to develop safety-related software.

TIKOSU
TIKOSU - Tietokantakeskeinen koneenohjausjärjestelmän suunnittelu (Database Centric Development of Machine Control Systems) is a FIMA project with a goal evaluating the efficiency of integrating various development process artefacts in an Integrated Industrial Documentation and Analysis database (IIDAbase).

1.3 Goal of research

The aim of this research is to gain more understanding on the requirements that various safety standards pose on developing safety-related applications on PLC equipment. What requirements are set for the equipment connected to the safety PLC, such as communication systems, reliable sensors or redundant architecture. The thesis addresses the most important issues and process models as well takes a look at two virtual environment solutions from Green Hills Software and SYSGO AG. Two different approaches of safe integrated development environments (IDE) from 3S and KW-software are also introduced to gain information on how different IDEs aid the implementation process of safe logic applications. Also an interview with three large Finnish OEM manufacturers were held to gain information on the interest on safety issues on the markets and how safety issues are taken into account in the industry today.

1.4 Delimitations

The aim is not to go too deep in the details in the standards but handle the most important issues in a way that gives the reader a picture on the whole process of developing safe machinery. Researching the IDEs have to be kept on light level because one of these environments, CoDeSys Safety, isn't released by the time this thesis is being written. The case project is also handled only on a theoretical level, for there is no plans on implementing a safety certified version of the machinery.

2 SAFETY-RELATED STANDARDS AND DIRECTIVES

Industrial standards are commonly used to unify certain structures such as electronic connectors or signalling methods. Safety-related issues are standardized to be able to control machine safety levels nation-wide and to ensure easy implementation of safe equipment for manufacturers. Safety-related standards on the European Union area are based on directives to ensure uniform safety norms and legislation in member countries. For example in Finland machine manufacturers are obliged to follow at least Konelaki 1016/2004, Koneasetus VNa 400/2008 [17] and Työturvallisuuslaki 738/2002. First two of the previous are based on EU's Machinery directive 2006/42/EC [18], which is the base of most of the machine- and safety-related Finnish acts and decisions. Machine directive can also be accompanied with other special directives affecting the product (for example. low voltage directive 2006/95/EC, ATEX directive 94/9/EC and pressure equipment directive 97/23/EC). Directives describe only the common and higher level requirements while the standards are set to fulfil the gaps and define the requirements in a more precise manner. [35]

Standards are considered *harmonized* if they meet the requirements of the directive completely and are confirmed by European Union authorities. All machines – such as weapons or chainsaws – can't however be completely safe. The standards define the state-of-art of the machine (the best technology that can be implemented at the moment) so the standard for these dangerous machines can still be harmonized and the machine can be considered legitimate. Unlike the obligatory directives, the standards (even the harmonized standards) are actually only guidelines and tools to meet the requirements of the directives and thus laws. If the manufacturer wants to use another approach than the one described in the standard, it has to be shown that the deviant solution meets the requirements of the directive. [35]

2.1 EN 954

EN 954: *Safety of machinery – Safety-related parts of control systems* is a safety standard mainly aimed for machine builders. This standard was taken into use in 1996, and it considers hydraulic, electromechanical and pneumatic control systems. The introduced methods are considered quite clear and simple to implement, but for example its risk graph isn't demanding enough for modern day safety equipment. [28, 36]

Part one (EN 954-1) doesn't take purely electronic, programmable control systems or programming into consideration and thus is considered obsolete after 31.12.2011. It was combined with EN ISO 13849-1 by the end of the year 2011 leaving only the latter operational.

EN 954 consists of the following parts:
Part 1: General design principles
Part 2: Validation

2.2 EN 50128

EN 50128: *Railway applications - Communications, signalling and processing systems. Software for railway control and protection system* is an application standard of IEC 61508 for software in railway safety equipment. The term *application standard* means that the standard introduces approaches and methods to meet the requirements set by its framework standard. Standards EN 50126 and EN 50129 specify the required level of safety and this standard specify the methods for meeting these requirements. EN 50128 specifies five levels for software safety called software safety integrity levels: SWSIL0 – 4. SWSIL0 represents non-safe software and software of SWSIL4 has the highest safety integrity. The requirements for reaching a specific integrity level are mostly in accordance with IEC 61508 and as the standard focuses mainly on software it is a good tool for assessing software safety. [35]

2.3 EN 62061

EN 62061: *Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems* is a harmonized application standard for IEC 61508. As the name of the standard states, EN 62061 handles functional safety of electromechanical, electric and programmable control systems. This standard, which is made for automation and machine designers, uses methods for assessing failure rates based on calculating the probabilities of possible failures on safety functions thus making it ideal for vast and complex systems. It is meant to complement the requirements stated in IEC 61508 with requirements needed in machine building and ease the interpretations of its framework standard by providing design models and practices made solely for machine control systems. [32, 36, 40]

2.4 IEC 61131

IEC 61131: *Programmable controllers* handles programmable logic controllers, their hardware requirements, wiring, programming languages, communication channels and user guidelines. From this standard, part 3 has had the most significant effect on software machine control industry. The third part, named "*Programming languages*", defines five different programming languages used in PLC programming: instruction list,

ladder diagram, sequential function chart, function block diagram and structured text. It also lists data types compatible with logic programming and different variable types.

Part 6: *Functional safety* is an update to the standard and is to be released in early 2012. It addresses functional safety in PLCs and their associated peripherals in the scope of hardware and software. Process models, safety assessment procedures, document production for development processes and end-user and development and verification planning for both hardware and software are discussed according to relevant safety-related framework standards. IEC 61131-6 standard fulfils the product specific requirements of IEC 61508-1, -2 and -3, and it also refers to the requirements of IEC 61511, EN 62061 and ISO 13849. The standard is mainly intended for functional safety PLC (FS-PLC) manufacturers.

IEC 61131 consists of the following parts:
Part 1: General information
Part 2: Equipment requirements and tests
Part 3: Programming languages
Part 4: User guidelines
Part 5: Communications
Part 6: Functional safety
Part 7: Fuzzy control programming
Part 8: Guidelines for the application and implementation of languages for programmable controllers

2.5 IEC 61508

IEC 61508: *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems* is an un-harmonized framework safety standard for all kinds of industry in the European Union. Although the standard isn't harmonized with the machinery directive, some harmonized standards, such as EN 62061 and ISO 13849 refer to its part 3 (IEC 61508-3: *Software requirements*) in an obliging way. EN 62061 has been derived from IEC 61508, and ISO 13849 is a "competing" standard from International Organization for Standardization ISO [16]. IEC 61508 has been designed to take future development into account in some extent. However, the software industry develops hastily and the current, 2nd edition of the standard was released in spring 2010. Many of the updates considered the third part of the standard. [24, 34]

As the standard considers all kinds of industry on a higher level, the industry-specific lower level standards have been derived from the IEC 61508 (Figure 2.1). Both the parent and the industry-specific standards have to be followed during the development process as the industry-specific standards may present more precise, adaptive instructions to the main standards.

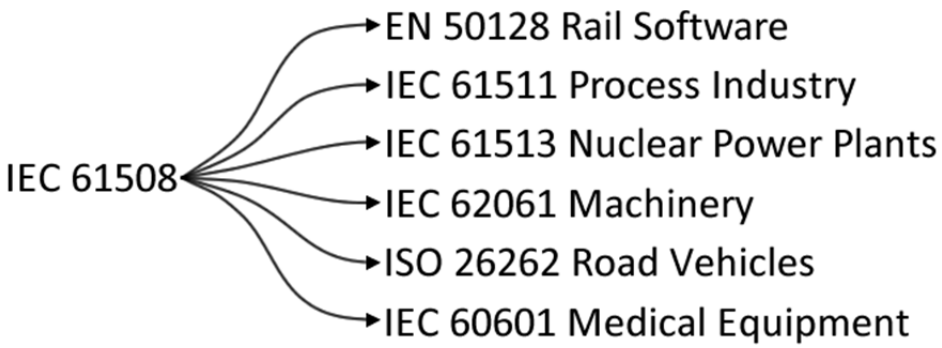


Figure 2.1. *Industry-specific safety-related standards based on IEC 61508*

The main issues considered in IEC 61508 are Safety Integrity Levels (SIL) and methods for designing safe machinery. SIL levels are used in determining the level of safety on safety-related equipment. Safety Integrity Levels are considered in more detail in Chapter 3.2.1. [34]

IEC 61508 consists of the following parts:
Part 0: Functional safety and IEC 61508
Part 1: General requirements
Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems
Part 3: Software requirements
Part 4: Definitions and abbreviations
Part 5: Examples of methods for the determination of safety integrity levels
Part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3
Part 7: Overview of techniques and measures

2.6 ISO 12100

ISO 12100: *Safety of machinery - General principles for design - Risk assessment and risk reduction* has been the main risk assessment and mitigation planning standard in machine automation industry since it was combined with ISO 14121-1 in October 2010. It includes the same risk assessment methods as described in Section 2.8 and presents general risk reduction and hazard elimination methods. The methods are based on knowledge and experience on design, use, common risks in machinery and information on previous accidents. [29, 40]

2.7 ISO 13849

ISO 13849: *Safety of machinery - Safety-related parts of control systems* is a harmonized standard for mechanical, hydraulic, pneumatic, electric and electronic control systems for safety equipment designers (product manufacturers) and for machine builders in some extent. [21, 22, 40]

ISO 13849 is a higher level standard which can be applied to design process of all kinds of machinery [24]. It isn't however applicable for designing systems requiring higher performance levels (PL) than PLe (comparable to SIL3, which is described in Chapter 3.2.1). As its methods are based on approximation of parameters and typically pre-calculated architecture models the standard cannot be applied to designing complicate and/or electronic systems [22, 32].

Part 1 (ISO 13849-1) describes safety requirements for control systems and guidance on design and integration principles of safety-related parts of control systems. Part 2 (ISO 13849-2) concentrates on validation issues including annexes in which control system safety principles and component and system failure modes are introduced. However validation of programmable control systems isn't considered comprehensive enough and it is advised to use other standards in these cases. [16, 24, 30, 40]

ISO 13849 consists of the following parts:
Part 1: General principles for design
Part 2: Validation

2.8 ISO 14121

EN ISO 14121: *Safety of machinery - Risk assessment* is a standard that has been used as a tool for assessing the severity of the dangerous failures and probabilities of dangerous events. It presents the main principles and concepts of risk assessment intended to be used to design the risk mitigation methods of ISO 12100-1. Beginning from October 2010, part one (ISO 14121-1) has been completely replaced by ISO 12100:2010. The second part (ISO 14121-2) includes practical guidance and examples of risk assessment and will remain operative under its old name and index. [16]

After a machine has passed a certain level in mechanical design phase, its risk assessment can be started. Primary risk mitigation should be implemented in the physical design phase by making dangerous parts and areas unreachable to users. This standard gives instructions on how to assess residual risks unhandled in the physical design. It concentrates on the dangerous situations caused by the control system. Appropriate risk reduction measures are formed based on the assessment made according to ISO 12100.

ISO 14121 consists of the following parts:
Part 1: Principles
Part 2: Practical guidance and examples of methods

3 DETERMINING THE LEVEL OF SAFETY

All the standards introduced in Chapter 2 handle different methods of defining safety of machinery or methods to increase it. But what actually is safety? How can it defined in this aspect? IEC 61508-4 defines safety as "freedom from unacceptable risk" [34]. Safety is a relative concept. The definition states that a safe system might not be completely risk-free, but the risks are divided into less and more serious risks. Decisions are made based on the division, according to which the mitigated risks are selected.

Neil Storey defines risk in his book *Safety-Critical Computer Systems* [39] as follows: "*Risk is a combination of the frequency of probability of a specified hazardous event, and its consequence*". The magnitude of the risk is thus influenced by the consequences of a possible following accident and how probable this event is. It is useless to construct an expensive system to prevent an event which consequences might be serious but which most probably would not happen in thousands of years. A dangerous event may be an event occurred by normal use of machinery in which an accident may happen through easy access to a dangerous area for the lack of, or malfunction of safety equipment. Methods for avoiding dangerous events caused by malfunctioning hardware or design flaws in control systems could be considered the main topic in the safety standards considered in Chapter 2. Malfunctions can be divided into three types [2]:

- System failure, which is often caused by hardware or control system malfunction.
- Random fault, when components fail due to environmental issues. Occurring of random faults is hard to assess.
- Systematic faults are usually hardware or software design flaws.

In Table 3.1 is described a simplified model to assess the magnitude of risk according to *Koneturvallisuus: Ohjausjärjestelmät ja turvalaitteet* [36]. In the leftmost column the effect of consequences caused by an accident is assessed with a scale from 1 to 100. Scale value one means minimum effect on the user, such as mere contact. Scale value one hundred means a loss of a limb or death. The second column describes the probability of occurrence for this event with a scale from 0.1 to 1. In the third column the consequence and probability columns are multiplied to gain the magnitude of the risk.

Table 3.1. *Risk assessment example [36].*

Consequence	Probability	Risk
100	1	100
100	0,5	50
100	0,1	10
50	1	50
50	0,5	25
50	0,1	5
10	1	10
10	0,5	5
10	0,1	1
1	1	1
1	0,5	0,5
1	0,1	0,1

The risk magnitudes acquired by this method can now be categorized to different risk groups. Table 3.2 is a table introduced in *Koneturvallisuus, Ohjausjärjestelmät ja turvalaitteet* [36]. Categories for different risk groups have been formed and each of the group has been set their risk mitigation requirements stated with performance levels and safety integrity levels.

Table 3.2. *Guidelines for selecting proper PL and SIL according to risk magnitude [36].*

MODERATE RISK	**SIGNIFICANT RISK**	**INTOLERABLE RISK**
Performance level c	Performance level d	Performance level e
SIL 1	SIL 2	SIL 3
TOLERABLE RISK	**MODERATE RISK**	**SIGNIFICANT RISK**
Performance level b	Performance level c	Performance level d
SIL 1	SIL 1	SIL 2
MINOR RISK	**TOLERABLE RISK**	**MODERATE RISK**
Performance level a	Performance level b	Performance level c
SIL 0	SIL 1	SIL 1

Risk assessment is an important phase in project design process. With it the project designers are obliged to consider all critical safety-related issues early in the development process. With a comprehensive assessment can a clear overall image of different risks be formed and it is easier to make an evaluation on which risks should be taken into account with risk mitigation functions. In some cases it may be wiser to keep the system simple and easy to maintain in order to enhance safety. The above-mentioned method is only a simplified method based on information given by ISO 12100 and ISO

14121-2. However it acts as a clear and simple example for understanding the instructions and options provided by the standards. In the risk assessment process also the industry-specific C-standards (Chapter 2.5) and the environmental requirements caused by the machine's operating environments should be taken into account, for they may provide further requirements for the assessment.

The reached level of safety integrity through safety functions can be stated in two different methods: ISO 13849 uses performance levels (PL) and IEC 61508 handles with safety integrity levels (SIL). The selection criteria between the two main standards are affected by several issues: the clientele, nationality or the industrial branch may already be familiar with either one of the standards. Also the type and complexity of the machine affects the choice. IEC 61508 and its application standards apply well to designing complex software-based control systems for its comprehensive software section in IEC 61508-3. ISO 13849 is used in designing less complicated systems or machines requiring a less programmable or electronic control system. A VTT publication *Turvallisuuteen liittyvät ohjausjärjestelmät konesovelluksissa* [21] suggest that the use of ISO 13849 limits to systems that in which:

- Low level of risk reduction is needed (PLa or PLb).
- Hardware failures are well known and they are easy to assess.
- The effect of PLC on safety is low, and needed risk reduction is moderate (PL=a, b, c, or d).
- Diverse hardware and software are used and the needed risk reduction is moderate (PL=a, b, c, or d).
- Certified software and hardware is used.

In the coming years ISO 13849 and IEC 61508 will be combined into a single safety main standard to avoid inconsistencies and excess repetition. [36]

3.1 Safety according to ISO 13849

ISO 13849 defines safety on two different methods: performance levels define the safety functions' ability to assure safety in case of a dangerous failure. Categories define the architectural methods to cope with hardware failures. Both the methods are based on the same failure detection methods and failure frequencies.

3.1.1 Mean time to dangerous failure

Mean time to dangerous failure ($MTTF_d$) defines the average time in years between failures that disable the safety function considered. This kind of failure cannot be detected by diagnostics, nor does is stop the machinery or lead it into a safe state. Safe state means a state in which the machine doesn't pose a threat to the user. This may mean a stopped saw blade, no traction on the tires or zero voltage on the system. This kind of failure is a state that couldn't be anticipated or foreseen. $MTTF_d$ can usually be

found in the machine's manual or the manufacturer's component information documents. While surveying the system's failure times, it is recommended to use minimal amount of different sources of information to ensure congruent calculation methods between the values of $MTTF_d$ [36]. If no information is given for a component, ten years can be used as a good guess [36]. Mean time to dangerous failure is only an indicative estimate, so it cannot be given an absolute and precise value.

ISO 13849-1 - *Safety-related parts of control systems – Part 1: General principles for design* introduces formulas to determine the average time between failures for single- and two-channel (see Chapter 6.1) systems. These formulas are based on the $MTTF_d$-values of individual components in the channel. Formula 1 [30] presents the calculation method for a single channel in which $MTTF_d$ represents the average failure time for the whole channel and $MTTF_{di}$ and $MTTF_{dj}$ the average failure times for each component in the channel. The first sum term in the formula represents each component considered separately and in the second sum term all similar components are grouped together. [30]

$$\frac{1}{MTTF_d} = \sum_{i=1}^{\tilde{N}} \frac{1}{MTTF_{di}} = \sum_{j=1}^{\tilde{N}} \frac{nj}{MTTF_{dj}} \quad (1)$$

$$MTTF_d = \frac{2}{3}\left[MTTF_{dC1} + MTTF_{dC2} - \frac{1}{\frac{1}{MTTF_{dC1}} + \frac{1}{MTTF_{dC2}}}\right] \quad (2)$$

If the channels in a redundant system are identical, can formula 1 be used to calculate $MTTF_d$ for both channels. If the channels differ, can the smallest $MTTF_d$-value be selected according to the worst case scenario or alternatively formula 2 [30] can be used. Formula 2 balances the common $MTTF_d$ formed with two channels $MTTF_{dC1}$ and $MTTF_{dC2}$. For example if $MTTF_{dC1}$ is 3 years and $MTTF_{dC2}$ is 100 years, the formula computes the complete mean time between dangerous failures to 66 years. This is considerably larger than the expected value of $MTTF_{dC1}$ which is also selected in case if worst case scenario methods were used.

3.1.2 Common cause failure

Common cause failure (CCF) is another factor used in determining performance level and thus the level of system safety. CCF stands for failure of multiple components, devices of safety functions due to failure of an interconnected single component, device or safety function [36]. Factors usually leading to common cause failures are:

- shared processors, memories and other components,
- functional dependence,
- physical proximity with other hardware or
- failure of communication buses.

Standards ISO 13849 and EN 62061 include methods for determining CCF. In these methods performing certain safety functions gives a certain score. The acquired points are summed up to a total score, which define the amount of common cause failures (β) as percentage. [36, 37]

3.1.3 Diagnostic coverage

Preventing all the failures in a system requires a great amount of work, and may still seem impossible. Dangerous events may however be avoided by recognizing the failures in time and performing certain actions to prevent accidents. Percentage of possible failures covered is called diagnostic coverage (DC). Different performance levels require a certain percentage of failures to be covered by diagnostics system. Also failure of the diagnostic system itself has to be taken into account. International standards ISO 13849 and EN 62061 give guidance on designing a functional diagnostic system. [36, 37]

3.1.4 Categories

Categories describe the architectural design requirements and the system's resistance to failures in five levels: B, 1, 2, 3 and 4. These categories are based on the safety categories of EN 954-1, the predecessor of ISO 13849. Their requirements for different categories are also almost identical. As EN 954-1 used categories to define safety per se, ISO 13849 uses them only as a part of safety definition. [28, 30, 36]

The lowest level of safety is needed in categories B and 1. They concentrate mainly on component choices and basic safety principles. Categories 2 to 4 have higher requirements with component failure rates and they also require use of failure detection and methods discussed earlier. Table 3.3 represents a comparison table between different categories following a detailed list of the categories.

Table 3.3. *Safety categories according to ISO 13849 [36].*

Category	Basic requirement	MTTFd (a)	DC (%)	Amount of CCF (%)
B	Adaptability to operating and environmental conditions.	3 – 29	< 60	Not relevant.
1	Well tested safety principles and components.	30 – 100	< 60	Not relevant.
2	Self-diagnostics.	3 – 100	60 – 98	CCF must meet the requirements of appendix F.
3	Tolerating a single failure.	3 – 100	60 – 98	CCF must meet the requirements of appendix F.
4	Tolerating a single failure. No undetected failures.	30 – 100	99 – 100	CCF must meet the requirements of appendix F.

Category B

The components used in the system have to endure the expected operating and environmental conditions so that failures are very unlikely to happen. ISO 13849-2 sets up a list of basic safety principles that have to be practiced. These principles are actions which should primarily be used to mitigate risks:

- Using proper materials based on temperature, load, corrosion, environment et cetera.
- Limiting force, torque or equivalent.
- Securing component position.
- Preventing unexpected start-ups.
- Simplicity, such as small component count or separating safety equipment from other equipment.

System $MTTF_d$ must be between 3 to 29 years. One fault can easily lead to a system failure. [30, 36]

Category 1

In addition to practicing basic safety principles, the use of well-tried safety principles and components is required. Failures are unlikely to happen and will guide the system to a safe state. $MTTF_d$ must be between 30 to 100 years. The system or subsystem can still fail in case of a single failure. But because of well-tried safety principles, a failure is less likely than in category B. [36]

Category 2

In addition to practicing basic safety principles and use of well-tried safety principles and components, self-diagnostics of safety functions is run at least on start-up and before a failure. It is recommended to run diagnostics from time to time during program run. The standard dictates that the testing frequency should be 100 times greater than the frequency of the need of safety functions themselves. Diagnostics should cover 60 to 98 per cent of the system. The amount of common cause failures must fulfil the requirements of appendix F in the standard and $MTTF_d$ must be between 3 to 100 years. [36]

Category 3

In addition to all of the requirements in category 2, category 3 machines should be able to execute safety measures even if there was a single failure in the system. . This can be achieved by a multi-channel (redundant) system, which means redundancy on sensors, actuators and safety equipment. In addition single failures can be detected with a diagnostics system. [36]

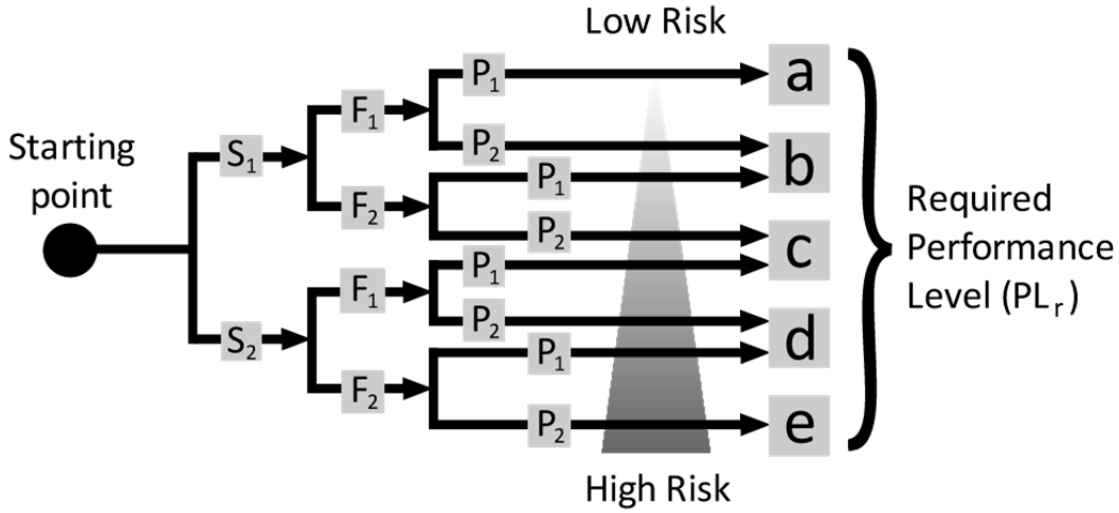
Category 4

Compared to category 3 machines, category 4 machines should detect all failures in the system. This is accomplished by redundancy and continuous automatic observation of the system. Diagnostics should cover 99 to 100 per cent of the system. The amount of common cause failures must fulfil the requirements of appendix F in the standard and $MTTF_d$ must be between 30 to 100 years. [36]

3.1.5 Performance levels

The performance level is a discrete level that specifies the ability of the safety-related parts of the control system to perform a safety function. They can be seen as description of ability to recover from failures in expected situations. Levels are between PLa to PLe, PLa being the least safe and level e being the most safe and the most demanding.

The required performance level (PL_r) is result of risk assessment, which means the amount of risk mitigation that the safety functions should carry out. The greater the effect of risk mitigation must be, the higher the performance level should be. Annex A in the first part of the standard (ISO 13849-1) introduces a method for defining the required performance level (Figure 3.1). Each risk acts as a starting point for this method. The first step in assessing the PL_r for a risk addresses consequences of the risks induced by failures. The second step addresses the frequency and exposure time of such failures and the last step addresses if the failure can be identified or avoided before an accident could happen. As a result of this method a required performance level is acquired.



Severity of injury
S_1 = Mild injury
S_2 = Serious injury, including death

Frequency and/or exposure to a hazard
F_1 = Seldom to less often and/or the exposure time is short
F_2 = Frequent to continuous and/or the exposure time is long

Possibility of avoiding the hazard
P_1 = Possible under specific conditions
P_2 = Scarcely possible

Figure 3.1. *Determining the required performance level (PL_r) of a safety function [25, 30].*

The performance level of components or safety functions can be defined using all the techniques described in Chapters 3.1.1 - 3.1.4 with a method shown in Figure 3.2. The categories and the level of diagnostic coverage are on the horizontal axis and reachable performance level is on the vertical axis. $MTTF_d$ is portrayed with different shades of grey bars. This figure introduced by ISO 13849-1 doesn't cover all the combinations of categories, DCs and $MTTF_d$s, and it is only an indicative figure which represents how different combinations and properties affect the PL. In the annex K of ISO 13849 is a table on which the Figure 3.2 is formed upon. The table in the annex includes more precise values for determining PL based on average probabilities of dangerous failures per hour.

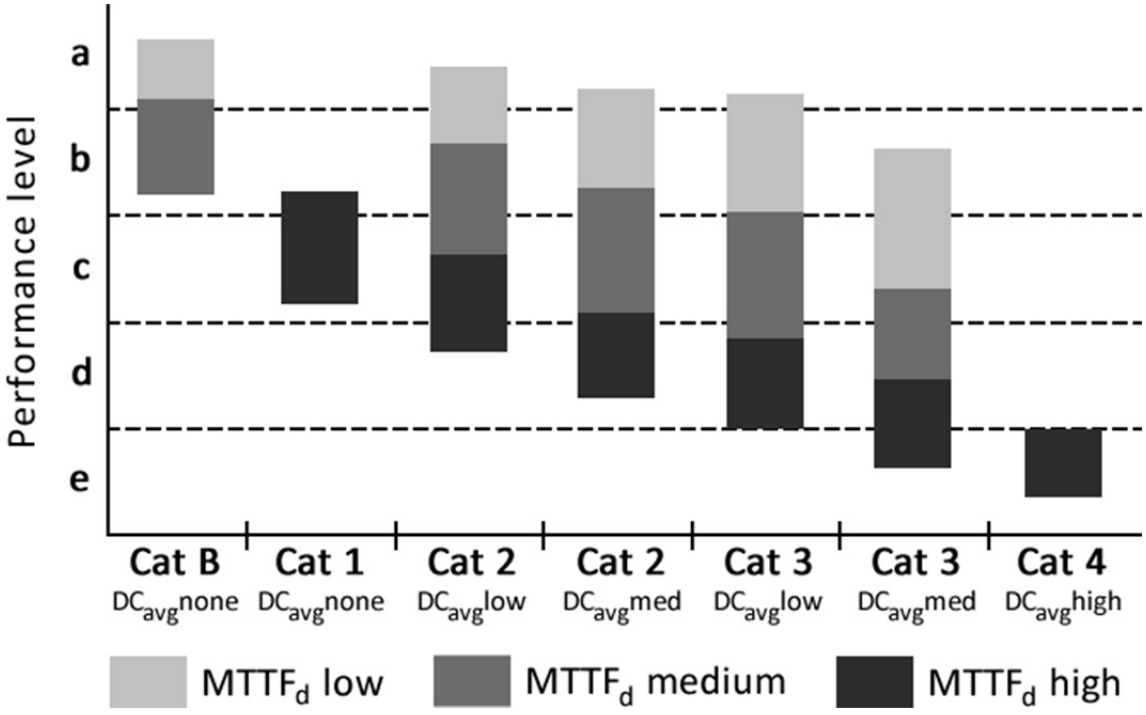


Figure 3.2. *Determination of performance level graphically [30].*

3.2 Safety according to IEC 61508

The safety framework standard IEC 61508 handles safety issues from a functional safety point of view. Functional safety is determined by the standard as "*part of the overall safety relating to the equipment under control and the control system that depends on the correct functioning of the electronic/electric/programmable electronic safety-related systems and other risk reduction measures*" [34]. Practically this means that the system and equipment should work as intended according to its inputs even in case of a failure. In more precise this concerns the control system of the system.

3.2.1 Safety Integrity Levels

IEC 61508 uses safety integrity levels (SIL) as a measure of safety. There are four levels of integrity: SIL1 to SIL4. SIL1 is the least safe and demanding, and SIL4 is the most demanding on the safety requirements thus making it the safest level. Expenditures rise drastically towards the safer levels due to redundant and special hardware, more comprehensive software testing and increased requirements on design processes. According to interviews made for local mobile machinery manufacturers the expenses and requirements for a SIL4-level system are so high that it is used less in the industry and is considered too demanding for machine automation. The application standard for machine automation addresses only levels SIL1 to SIL3.

The calculation methods for determining the safety integrity level for automation systems are quite the same as for performance levels in ISO 13849. The failures frequencies for components and subsystems are calculated, and the required amount of risk reduction for the function is evaluated. The unit for a failure frequency is called the probability of dangerous failure per hour (PFH_d). As for the $MTTF_d$, the component and system manufacturers may announce the values of PFH_d in their product documents. Evaluating the failure frequency requires quite complex calculus which should take the intended usage time (T_M) into account to ensure also long term reliability. [34, 36]

Table 3.4. *Comparison between SIL and PL [36].*

SIL level	Performance level	Probability of dangerous failure PFH_D	Years of use before failure (approx.)
-	a	10^{-5} … 10^{-4}	1 … 10
1	b	$3 \cdot 10^{-6}$.... 10^{-5}	10 … 40
1	c	10^{-6} … $3 \cdot 10^{-5}$	40 … 100
2	d	10^{-7} … 10^{-6}	100 … 1000
3	e	10^{-8} … 10^{-7}	1 000 … 1 0000
4	-	10^{-9} … 10^{-8}	10 000 … 100 000

As it is for performance levels, the process or a complete system cannot be assigned a safety integrity level. SIL is a measure for safety functions. It can be used to determine the level of reliability in case of a system failure. If a product is equipped with a component labelled as SIL# it might not mean that the complete system and its safety equipment meet the requirements of SIL#. Therefore it is suggested to use terms like SIL claim limit and SIL capability introduced in EN 62061 to distinguish between meanings of different types of SIL. [8]

SIL claim limit

Main hardware parts such as input (sensors), control system (PLC or relays) and output (actuators) on safety functions are usually interconnected in serial. A claim limit for a safety function means that the SIL definition is limited to the level of the component or subsystem of the lowest SIL [8]. A chain is as strong as its weakest link. Figure 3.3 depicts a chain of hardware which consists of SIL2 components and it reaches SIL level two. However if a single component for example sensor A2 or actuator B would be SIL1 and the other components remained the same, would the architecture reach only SIL1 according to the lowest level rating. [32]

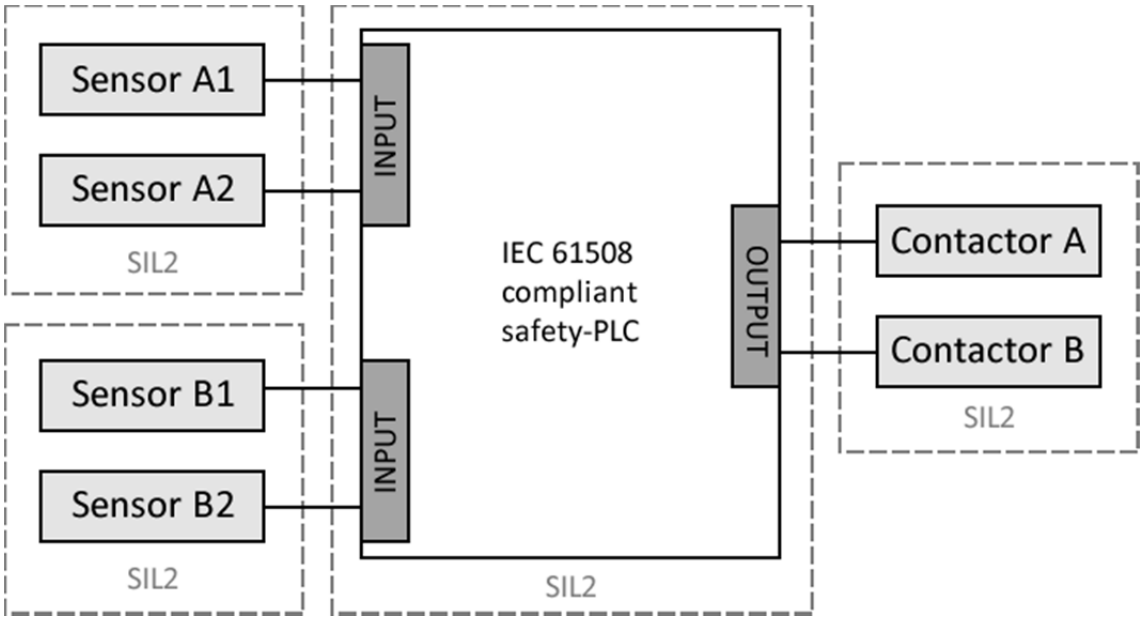


Figure 3.3. *Hardware architecture for a safety function with two inputs and one output implemented with redundant SIL2 -compliant system [32].*

SIL capability

Maximum SIL for a system which can be reached in relation to architectural constraints and systematic safety integrity [23]. A component rated with a certain SIL capability is capable of forming a safety function reaching at most the same safety integrity level as the components SIL capability. For example a component certified to SIL2 can be used in forming a safety function mitigating risk worth of SIL1 or SIL2. The second edition of IEC 61508 states however that two redundant lower level components can form a higher integrity level component. For example two SIL2 devices connected in parallel form a SIL3 component. [32]

To reach a certain safety integrity level, the design process must take into consideration all the aspects and influences of system design, process design, mechanical design, component selection, safety principles, architecture, operations and control system design. Overall safety consists of many issues and requires collaboration between several departments. [8]

3.3 Risk analysis studies

A risk analysis is required in an early phase of safety-related hardware and software projects. Analyses are run to identify as many hazardous events in the system as possible and to form each event a risk mitigating safety function. Analysis methods can be based on documenting and mitigating already known risks or to reveal new risks through analysing the whole system one part at a time. The risk assessments can be divided roughly into four steps according to ISO 14121 [31, 40]:

- Define the machine's limits.
- Identify the hazards, dangerous situations and events.
- Assess the severity of dangerous event (mere bruises - need for sick leave - loss of limb or death).
- Assess the probability of an accident due to a dangerous event.

Table 3.5. *Software assessment techniques according to EN 50128 [33].*

Technique/measure	SWSIL0	SWSIL1	SWSIL2	SWSIL3	SWSIL4
Cause consequence diagrams	R	R	R	R	R
Event tree analysis	-	R	R	R	R
Fault tree analysis	R	R	R	HR	HR
Software error effect analysis	-	R	R	HR	HR
Common cause failure analysis	-	R	R	HR	HR

Table 3.5 presents requirements set by EN 50128 to its five software safety integrity levels (SWSIL0-4). 'R' stands for "recommended" and 'HR' for "highly recommended". IEC 61508-3 has a similar list of recommendations, but it doesn't prefer any method to other within its different safety integrity levels. In the following subchapters three different risk analysis methods based on different basic analysis principles are introduced. These methods represent typical risk analysis method types, but are only a few among many other methods.

3.3.1 HAZOP

Hazard and Operability Studies (HAZOP) is an analysis method for identifying hazardous events based on the expertise of analysis attendees. The method, described in international standard IEC 61882, tries to identify all the relevant and possible hazardous events and their causes, assess the severity of those events and plan the risk mitigation

or elimination methods. A group of five to seven people attend meetings in which the system is being analysed. The group should consist of people from different backgrounds: safety experts, software and machinery engineers, end-users and other personnel who may have something to contribute to the subject. It is suggested in United Kingdom Ministry of Defence standard 00-58 [44] that the attending personnel at least partly changes from meeting to meeting to give a wider perspective to the assessment. Up to 20 different actors can attend the meeting during the analysis process, but the number of attendees for a single meeting shouldn't surpass ten people to maintain efficiency. HAZOP can thus be described as a brainstorming technique.

The system under inspection is analysed according to bottom-up procedure, which means that the process is started from single blocks or components. Each component or factor is being analysed for all possible failures or deviances from its normal functionality. Types of these deviances are determined with specific guide words. Table 3.6 list all the HAZOP guide words and their descriptions. All of the deviances are collected into a single tabular analysis in which the causes, their consequences, safeguards that reduce deviation likelihood or severity and the required measures for preventing these deviances are listed. A single deviance can have multiple causes and measure suggestions, so the analysis can get quite large and laborious when executed in great detail.

Table 3.6. *HAZOP guide words [44].*

No / not	This is the complete negation of the design intention. No part of the intention is achieved and nothing else happens.
More	This is a quantitative increase.
Less	This is a quantitative decrease.
As well as	All the design intention is achieved together with additions.
Part of	Only some of the design intention is achieved.
Reverse	The logical opposite of the intention is achieved.
Other than	Complete substitution, where no part of the original intention is achieved but something quite different happens.
Early	Something happens earlier than expected relative to clock time.
Late	Something happens later than expected relative to clock time.
Before	Something happens before it is expected, relating to order or sequence.
After	Something happens after it is expected, relating to order or sequence.

It is worth carrying out a HAZOP analysis in the early phases of the project to avoid unnecessary operations and repairs. United Kingdom Ministry of Defence standard 00-58 suggests that analysis is carried out on different phases and multiple times during the design process [44]. Analysis can also be performed on multiple levels. High-to-low level analysing can be useful while testing large systems. The system is first analysed on a higher level, when the most detailed levels are subsystems. Each subsystem is analysed further with more detailed analyses if considered necessary. Limited by the risks of hazardous events it is worth balancing between laboriousness and detail. It is not worth analysing irrelevant and lower severity events in too much detail. Also the use of other analyses, such as use of top-down principle method FTA described in Chapter 3.3.3 is suggested to be used with HAZOP to achieve a wide perspective on the system. [44]

3.3.2 FMEA

Failure Mode and Effect Analysis (FMEA) is an analysis method described in three standards: IEC 60812, MIL-STD-1629A and SAE J-1739. This method assesses the effects of component based failures on system safety and functionality on a bottom-up based technique. The primary system type for analysis is non-programmable systems [20], but software-based systems have been developed their own version of the method named Software Failure Mode and Effect analysis (SWFMEA).

The analysis process starts by selecting a component which failure types and effects of a failure on the system are assessed. The component is come up with two to three different ways it can fail. Each of these failures is added two to three consequences or effects on the system. Also methods for avoiding the failures, their probabilities, severities and other relevant information are listed. All of this information is collected into a tabular report, which usually consists of sixteen columns. Eleven of these address the FMEA process and five addresses action results. Table 3.7 list the fields of the report.

Table 3.7. *FMEA worksheet column headings [6].*

FMEA process	**Action results**
Item and Function	Action Taken
Potential Failure Mode	Severity
Potential Effect(s) of Failure	Occurrence
Severity	Detection
Potential Cause(s) of Failure	Risk Priority Number
Occurrence	
Current Controls	
Detection	
Risk Priority Number	
Recommended Action	
Responsibility and Target Completion Date	

With this systematic method it is easier to detect random failures and more failures altogether than with top-down methods. On the other hand, the analysis can get quite laborious due to all the relevant components in the system have to be analysed. Large systems should consider modules or subsystems as the lowest level components to mitigate the analysis workload needed. Also balancing between the more critical and non-critical components (according to the scope selected) will enhance the efficiency of the analysis and save time. [15, 20]

FMEA has also its weaknesses: it doesn't detect human errors, sequential errors or design failures. Because the analysis is based on analysing a single component, it cannot either detect common cause failures. Therefore it is worth using the method with other analysis methods, making it possible to analyse the system with multiple points of view. [15, 20]

FMEA-based SWFMEA addresses analysing possible software failures that can lead into a system failure. The main phases in the analysis process are to define the criticality of the failure and define the measures to avoid it. For example the failure can be detected with diagnostics or tests or the failure cannot occur due to program structure. Hardware based failure analysis is easier to implement than a software based analysis. Component failure rates, for example for resistors of relays are well known, the parts fail due to aging and wear and failure data is available based on experience, manufacturer information and reliability tests are publicly available. Software modules on the other hand do not literally fail, they only display incorrect behaviour and depend on the dynamic behaviour of the application. The hardest part in SWFMEA is to define different failure modes. Literature describes different definitions, but no well-established modes are settled on. Ristord and Esmenjaud suggest following failure modes [27]:

- the operating system stops
- the program stops with a clear message
- the program stops without clear message
- the program runs, producing obviously wrong results
- the program runs, producing apparently correct but in fact wrong results.

Goddard states that two types of SWFMEA can be defined [6]. System software FMEA concentrates on the system on a higher level. It is executed on an early phase to analyse the efficiency of the architecture. Detailed software FMEA could be compared to component level based FMEA. It is quite laborious and the results can be attained in a late state of the project. This type of analysis is often cost effective only for systems with limited hardware integrity. Goddard states that function block level analysis would be a proper level for inspection, but report *Failure mode and effects analysis of software-based automation systems* from Finnish radiation and nuclear safety authority STUK [6] finds this unfeasible for it creates extensive and complicated analysis and

function block failure modes are not known. Ristord and Esmenjaud however suggest that only application (function) level should be analysed [27].

According to the STUK report [6] SWFMEA method is hard to implement, but is useful in the early phases of the project, because it reveals weaknesses and saves from further development or repair needs. The analyses require knowledge of the software from the analysers so the analysis group might also be harder to select in case only a few people take part in implementing the code. As with normal FMEA, also the use of other analysis methods is suggested to reveal other types of failures and consequences.

3.3.3 FTA

Failure tree analysis (FTA) is a hazard analysis method described in IEC 61025 – *Fault Tree Analysis.* It is based on analysing failures and causes that lead to known hazardous events. The method differs from the pre-mentioned methods with its execution order. The analysis begins from the known hazardous event or undesired event and proceeds according to top-down principles towards the causes or failed components which caused the event. This method is less laborious than bottom-up methods for it analyses only undesired events that are already known. Therefore it is useful for complex systems and systems comprising of great amount of components [20]. Top-down methods can also be easily implemented with V-model based process models.

As HAZOP and FMEA base their analysis on failure of single components, failure tree analysis works on a higher level taking also human factors, common cause failures, non-functional errors and sequences that may lead to an undesired event into account. It however analyses only known failures and it may not detect all previously known failures as bottom-up method would. This method is thus suggested to be used with other methods, such as FMEA or preliminary hazard analysis (PHA), in order to increase the amount of found hazardous events or failures.

As a result from failure tree analysis tree-like diagrams are created. The analysis process starts from selecting an undesired or hazardous event such as user injuring his fingers between moving paper mill rolls that should be at a standstill. Only one event can be appointed to a diagram. Inspection resolution and the scope in which the system is analysed are chosen. These selections affect the step sizes and the type of events that are recorded into the diagram. The selected event is divided into sub-events that are the causes that lead to the main event. These events are connected to the main event with logic gates such as AND, OR or XOR. Each of these events is divided into sub-events in a similar manner. This is repeated until the lowest level events cannot be divided anymore within the scope and resolution of the analysis. This formed diagram can be used to assess the effect of single events or failures on the undesired event and prioritize risk mitigation methods based on the severity of the events. Each event can be appointed severity and a probability of occurrence. These sub-probabilities can be used to calculate probability for leading to the undesired or hazardous event due to a precise sub-event. [15, 20]

3.4 Certification

Certification is a way to prove that the functionality of a device, service or process is in correspondence with some national requirement or law or that these services reach a certain level of quality. Certification is generally carried out by an independent, national body against usually national or international standards [34]. In Europe these organizations are for example TÜV - Technischer Überwachungs-Verein (Technical Inspection Association) and EASA - European Aviation Safety Agency.

Considering safety, a system can be verified to different levels of safety integrity defined by to ISO 13849 and IEC 61508. A device can however be certified also to an application standard according to its type. The certification process ensures that different process phases have been executed according to requirements presented by law, either by strictly following the guidance given by safety standards or otherwise meeting the requirements. In case of possible deficiencies found by the certification body, should these be corrected and re-inspected with all other affected parts. VTT recommends co-operation with a certification body throughout all project phases to gain useful guidance before even making deficiencies that may lead to extra certification inspections. This kind of co-operation will cost more on the project implementation phase, but will pay itself back during the certification and possible modification phases.

For a system to reach a certain SIL- or PL certificate, all its safety functions should reach an appropriate safety integrity or performance level. For a safety function to reach that level, all its components should have an appropriate SIL claim limit according to the desirable safety integrity level (taking redundancy principles into account) [38]. This concerns all sensors, actuators, communication and energy transmission channels, drivers and software. A separate software application doesn't have a safety integrity level itself, but integrating it into a system makes it a part of a safety function thus making it possible to define it a SIL. The reachable safety integrity level is affected by the conditions of design, implementation and verification phases, its software characteristics and other properties defined by safety standards.

The significance of researching certification of software modules and components has increased with the growth of automation software applications. If some generic application modules implementing safety functions could be certified for common use, the cost of certificating a complete system would decrease and the value of reusable software components increase significantly. PSafeCer-project has been researching on differences between traditional top-down de-compositional approaches forming modules and bottom-up compositional approaches with pre-certified components [43]. By forming the base of safety-related software application from certified modules would enable great savings in software development expenses.

Even if the software application could be formed with certified software modules, it still needs to be validated according to safety standards. This demand arises from the requirement that the certified module can only be used in precisely the same function it is certified to. The linking of different blocks and the validity of their parameters

need to be verified even if the inner implementation of the module was already certified. International standards ISO 13849-1 and EN 62061 give guidance on validation of software modules. [30, 32]

On top of the used software and hardware IEC 61508-7 recommends that the tools used in programming, simulation and design processes should also be certified. Tools should be certified in order to have some level of confidence regarding the correctness of their outputs. Translators are usually certified according to the programming language standards, not so much according to their safety functionalities. [34]

An interview held for three large Finnish OEM manufacturers of automated mobile machinery brought up that safety certificates on this branch have mostly only sales promotional value. The customers don't require a safety certificate given by an official body and often mere declaration of conformity given by the manufacturer is sufficient. On top of this the high requirements of safety standards increase significantly the design expenses and thus the product price. The present economic state has led to a situation that the lowest grade units have the greatest sales making it too expensive for manufacturers to produce or develop more expensive models. However the interest in researching safety issues has risen and the situation may change in coming years.

4 IMPACT OF SOFTWARE ENVIRONMENT

Software environment in this aspect is the environment in which the machine control application is running. It can be Linux or Windows-based operating system, hardware-specific bare-bone or the system may comprise of all of these in the same time in a collection of virtual operating systems. The most important issue in selecting an operating system for machine control application is to ensure the stability of the system. Crashing operating system may leave the system in an unsafe state unless the control is designed in a way that the loss of control directs the system to a safe state or performs a safe run-down. This is why all the safety features should be run on as safe a system as possible.

The most significant negative effect of an unsafe operating system is that it represents a great potential CCF source. A single malfunctioning operating or management system may disable machine control and safety systems simultaneously if they are not running on separate independent systems. IEC 61508-3 annex F lists different methods for achieving independence between different modules on a single hardware. The techniques can be implemented to elements with difference between systematic capability, elements which contribute to the same safety function and software and elements contributing to safety function and non-safety-related software on same hardware. [34]

IEC 61508 requires that independence between safety-related elements should be achieved and be demonstrable in spatial and temporal domains. Spatial independence means that each element is responsible only for its own data. An element should not be able to change data in another element. Especially non-safety-related elements should not be able to interfere with safety elements. Methods for reaching spatial independence are as follows [34]:

- Use of hardware memory protection between different elements.
- Use of an operating system which reserves each element its own virtual memory space which is supported by hardware memory protection.
- Use of design, code and analysis methods which can demonstrate that elements do not use common memory banks, which could lead to unintended overwriting of data.
- Data hierarchy which protects data of a higher integrity element to be overwritten by a lower integrity element.

Ideally spatial independence means that elements shouldn't communicate with each other. Unidirectional interfaces such as messages and pipes should be preferred to use of shared memory. Temporal independence requires that an element should not cause another element to function incorrectly by taking too much available processor

time or locking another common resource of some kind. Temporal independence can be reached by [34]:

- Use of deterministic scheduling methods such as cyclic scheduling algorithms or time triggered architectures.
- Use of priority based scheduling implemented by a real-time executive with a means of avoiding priority inversion.
- Time fences which will terminate the execution of a non-safety element if it over-runs its allotted execution time or deadline.
- An operating system which guarantees that no process can be starved of processor time, for example by means of time slicing. However hard real time cannot be implemented.

Different operating systems implementing virtualization layers exist on the market. The main principle in these systems is the restriction of operating systems (OS), application programming interfaces (API) or runtime environments (RTE) on separate spatially independent virtual memory areas. Virtualization layers can also enable temporal independence by for example reserving a specific amount of processor time for each layer. This enables running several independent operating systems or run time applications simultaneously on single hardware. In systems implementing only one operating system and no virtualization properties the system need to be certified according to a single safety integrity level including parts that don't need to implement any safety features. This may be quite time consuming and laborious due to complex monolithic kernels that support all the properties required by the used components. Through virtualization the amount of certification work can be reduced by using smaller and simpler kernels tailored for different virtualization layers. Now only the modules implementing safety-related functionality need to be certified. [7, 42]

One manufacturer of virtualization solutions is Green Hills Software, which develops real time operating systems and embedded development tools. Green Hills' product INTEGRITY RTOS is a real-time operating system (RTOS) which enables use of multiple operating systems on single processor and peripheral hardware. INTEGRITY RTOS can separate main program from secondary functions as video, audio, internet connection, duplicated control applications or safety critical applications. Separating functionally different elements completely from each other clarifies the design process by eliminating unintended use of common memory banks for different elements. [7]

Memory areas and devices can be fixed or dynamically adjusted during runtime to different layers which can be used to reserve memory areas for different functions. Each layer is monitored by the main software for malfunction and changes in performance. These diagnostics enable restarting a failed operating system without affecting the other layers; however use of safety certified operating systems are required as the base of safety-critical applications. Temporal independence can be reached by binding different cores or multiple cores simultaneously to a single layer. INTEGRITY RTOS

supports multicore processors and hyper threading which ensures that each element can have full processor time every time needed. [7]

Another manufacturer in the virtualization RTOS business is SYSGO AG with its PikeOS concept. It combines real-time operating system, virtualization platform, and Eclipse-based integrated development environment (IDE) for embedded systems [42].The concept is validated according to safety standards such as DO-178B, EN 50128, IEC 62304, IEC 61508, ISO 26262 and IEC 61513 and allows security certification according to the Common Criteria (CC) standard up to EAL level 7. The virtualization architecture is similar to the architecture used by the Green Hills solution. The partitioning concept (Figure 4.1) is described in the system partitioning and scheduling specification ARINC 653 commonly required in safety-critical systems in the avionics industry. [42]

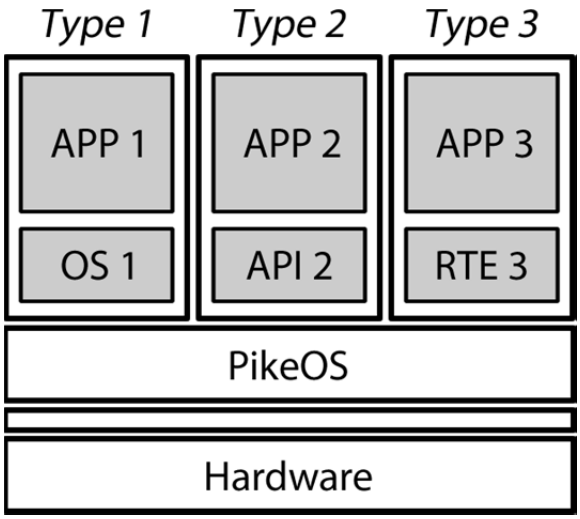


Figure 4.1. *PikeOS partition according to ARINC 653 [42].*

Each partition can have its own OS, API or RTE controlled by its own micro-kernel provided by PikeOS. These micro-kernels provide only basic functionalities making the partitions (virtualization layers) easily customizable and easy to certify according to safety standards. Each virtualization layer receives its own system resources such as memory, I/O devices, CPU-time and such. These resources can also be divided into several subsets to completely separate the layers from each other making it impossible for the layers to affect each other. The concept includes a health monitoring system which can be used to detect address violations, timing violations and illegal instructions and handle these according to system configuration. PikeOS comes with its own Eclipse-based integrated development environment CODEO, which can be used for system configuration, target monitoring and timing analyses. [42]

These virtualization solutions present different software-based redundant systems, but they also have their weak spots. As they may utilize multicore processors and can divide memory banks for different applications, they still will be using the same physical hardware. The hardware can use a single memory element and power source,

same motherboard and other components which can fail the complete system if malfunctioned. At the moment bare-bone safety PLCs have all their components duplicated for survival over a failure of a single component. These virtualization layer solutions are however SIL3 capable as a component.

5 COMMUNICATION ISSUES

Communication technology in machine automation has moved from multiple separated analogous channels to single or double channel bus and Ethernet systems. In the older analogous communication systems the messages were transmitted in separate cables as voltage or current messages, which requires great amounts of wiring even in relatively simple devices. Bus and Ethernet communication systems are based on digital technology in which the system signals are implemented into standard message frames defined by a certain communication protocol. These message frames are transmitted into a network which single transmission channel can include messages from multiple devices. With this kind of communication network can great savings be reached in the amount of required cabling, device weight and in larger devices also in communication system costs. Also the risk of cable breaks diminishes and the serviceability of the communication cables gets easier. [40]

The amount of different bus- and Ethernet protocols is vast, and no single technology has been standardized for common industrial use. This is due to the fact that in the early days of digital industrial communication methods several big companies started to develop their own standards and competition has led to a great variety of different techniques. Beside different companies, also different industrial branches have their own protocols. [40]

Compared to old analogue technologies, digital network traffic can however lose individual messages or even the complete connection after a certain failure threshold due to different communication failures or disturbances if poorly implemented. Different failure types are presented in IEC 61508-2, IEC 62280-2, IEC 61784-3 and EN 61000-series standards. In book *Sichere Bussysteme in der Automation* [26] authors Reinert and Schäfer divide typical network failure modes into three classes: failures independent on human actions (failures 1 to 7 in Table 5.1), failures dependent on human actions (failures 8 and 9 in Table 5.1) and failures directly or indirectly dependent on human actions (failure 10 in Table 5.1). [26]

Table 5.1. *Typical network faults [26].*

1. crosstalk	6. aging
2. broken cable	7. temperature
3. EMC failure	8. human failure
4. Stochastic failure	9. wiring failure by human
5. Stuck at failure	10. transmission of non-authorized messages

Likelihood of failures dependent on human actions can be affected by proper training and instructions, and transmission of non-authorized messages into the network can partly be prevented with use of security principles like user identification. Failures 1 to 7 can be affected with proper shielding of devices and cabling, for which EU directive 2004/108/EY: *Sähkömagneettinen yhteensopivuus* (Electromagnetic Compatibility) lays Europe Union -wide requirements. When the network-based failures have been mitigated to an acceptable level, the network traffic itself can inflict different failure modes. To mitigate the risks of these failures the methods provided by safety communication protocols are presented.

Tapio Siirilä states in his book *Koneturvallisuuden perusteet* [37] that if safety-related information is to be transmitted on a bus, the bus technology needs to be a certified safety bus, a normal bus isn't reliable enough [37]. Matti Sundqvist mentions in his training material [41] that SIL3 is a satisfactory safety integrity level for communication. The safety inducing effect of safety communication protocols is based on ensuring the receiving of messages and verifying the message information. These methods are used to avoid systematic, continuous and thus easily detectable errors and detect stochastic, random errors [15].

Safety bus technology can be an independent network consisting only of safety-related messages or a safety add-on implemented on top of a normal communication protocol [40, 41]. Safety protocols based on add-ons work as a "white channel" implementation on ISO OSI level 7 on top of standard bus protocol and thus do not require separate cabling or affect the standard protocol. *White channel* implementation means software implementable solution, and *black channel* symbolizes the communication equipment hardware. White channel solutions are compatible with standard bus technologies and many protocols also allow simultaneous transmission of non-safe and safety-related messages in the same network [15]. To enable simultaneous transmission, different types of messages are diversified from each other to enable different message handling procedures.

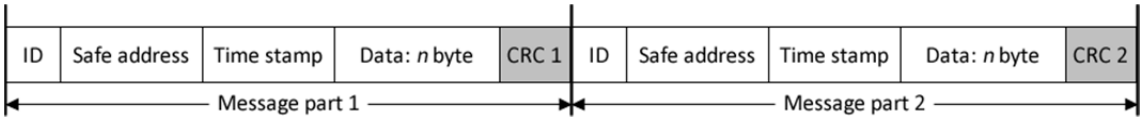


Figure 5.1. *Generic safety-related message format [15].*

Safety-related messages and all their verification method fields are embedded into the data payload field of standard bus messages. Figure 5.1 depicts a generic safety-related message format in which the message is duplicated into two messages. Embedded data is inverted in the second message and the messages are crosschecked in the receiving end. This type of messaging is implemented in for example CANopen Safety. Listed below are the most common verification methods used in safety-related messages in bus protocols [15]:

ID — ID identifies each message with a unique number. It is used to detect insertion of message from a faulty node and identify corresponding redundant messages.

Safe address — A special sender address is used to identify messages sent from a known and safe message producer to prevent transmission of non-authorized messages. Safe address must be known by the consumer; otherwise the data will not be processed.

Timestamp — A message can be delayed and therefore outdated if the time between sending and receiving the message gets too long. Adding a timestamp to the safety-related message can be used to detect delay caused by congestion, storing of messages by routers or wrong routing. It can also be used to detect wrong sequence and repetition of messages. Successful use of timestamps requires synchronization among the involved nodes.

Safety-related data — Data field includes the embedded safety-related sensor, actuator or other data.

Cyclic redundancy check — Before transmission, the messages are given a verification value which is calculated from the polynomial division of the message's contents. The check value is compared in the receiving end with the data to ensure data is uncorrupted. Cyclic redundancy check (CRC) can be used to detect stochastic faults like a bit fault leading to a corruption of the message.

Redundancy (Chapter 6.1) is featured also in the communication systems. It can be realized with a single safety-related controller implementing digital redundancy, or with dual controllers implementing physical redundancy. The communication medium

can also be single- or dual channel implementing MooN-architecture (Chapter 6.1) for example two bus cables or two wireless frequencies.

Digital redundancy Duplicating each message, verifying their CRCs and comparing their data together enables digital redundancy. This reduces the risk of message corruption to a minimum. [40]

Physical redundancy Physical redundancy is reached by processing duplicated messages by two safe controllers. Each field is verified and cross-compared between the controllers. Only if both controllers get the same positive result, the message is processed. Physical redundancy is used to avoid hardware-based systematic communication failures. [40]

If the amount of information to be transmitted through these safety messages is large, may the small size of message capacity lead to increased traffic on the channel. Because some safety protocols duplicate messages on a single channel, the risk of channel congestion rises [40]. This may lead to problems on systems requiring real time properties unless the state of the bus is monitored or the channel bandwidth is scaled big enough in the first place. This aspect needs to be taken into consideration especially if an old system is to be retrofitted with a safety bus.

Guidance on designing and using safety-related communication protocols can be found on IEC 62513: *Safety of machinery - Guidelines for the use of communication systems in safety-related applications.* Safety bus architecture, its failure types and methods for detecting and removing communication errors can be found on IEC 62280-1: Railway applications - Communication, signalling and processing systems - Part 1: Safety-related communication in closed transmission systems and IEC 50159-2: Railway applications - Communication, signalling and processing systems - Part 2: Safety related communication in open transmission systems.

5.1 Selection criteria

IEC 62513: *Safety of machinery - Guidelines for the use of communication systems in safety-related applications* guides system designer in choosing and defining a safe communication system. Before digging deeper into different communication protocols it is worth getting to know your own device and its requirements for communication.

The used equipment states requirements for the bus with the amount of connectable nodes. While scaling the system the present needs should be taken into account, but also consider possible future changes on the assembly. Also operating environment issues, such as environment and device temperatures, vibrations, shocks and possible

electromagnetic interferences from nearby cabling affect the physical requirements on the bus to be selected. [40]

Bus speed can be critical for the system to work safely. Equipment requiring real time properties or fast reaction may require time triggered protocols, which all bus technologies do not support. The bus may also overload because of too much traffic on the channel, when even the most critical messages can be received too late and thus cause a hazardous event. In this case buses with higher bandwidth, different network topologies or protocols that support prioritizing critical messages at non-critical messages' expense should be considered. [40]

When setting requirements for bus reliability it should be determined should the bus be able to survive losing a single message. Reliability can be increased by duplicating equipment or by using two or more channels. Reliability is also inversely proportional to communication cable width. This can be influenced by shortening the transfer lines or lowering the bus bandwidth thus diminishing the effect of disturbances on the quality of the transmission. [40]

Some safety buses permit standard communication messages to be sent simultaneously on the same channel. Restricting safety-related messages from non-safety-related messages decreases risk of channel overload and some safety-related bus protocols use separate channels to ensure sufficient bandwidth for critical messages. It is also advisable to consider security issues if unauthorized personnel or equipment may connect to the bus intentionally or accidentally to listen or modify its traffic. This is especially important on wireless protocols. [40]

5.2 CAN

Controller Area Network (CAN) protocols feature a few different safety extensions. They differ in their relation to normal communication, redundancy and restrictions in compatible hardware.

CANopen Safety is a safety add-on for CANopen networks from CAN in Automation (CiA) for the use of mobile machinery. It is based on using a new message type: SRDO (safety-related data object), which consists of two CAN-messages that are sent periodically. One of the messages is inverted and timestamp and CRC are used to determine the correctness of the messages. The framework allows using non-safety-related messages in the same network. [40]

Another approach for CANopen is used in *SafetyBUS p* developed by Safety Network International e.V. It sends messages only when their state has changed thus decreasing bus traffic. The protocol is open for different hardware but limits the amount of different connection to 64 nodes. The messages aren't duplicated and the channel isn't redundant but the bus is physically and logically redundant from normal communication traffic. [40]

A third approach is presented by *EsaLAN*. The protocol based on CAN bus is restricted to only one manufacturer equipment and limits the amount of connections to 7 nodes. The protocol is mainly used in machine control and robotics. Safety is reached by limited communication traffic and redundant and divergent structure and internal cross-reference. The protocol is also applicable with wireless systems.

5.3 Ethernet

One might think Ethernet solutions aren't suitable for machine and real time process control systems due to their nondeterministic nature. It is true that traffic through standard-Ethernet and TCP/IP connections isn't deterministic. One can determine an address, but not the time of arrival or route of the message. Therefore it is not reliable per se as a fast and reliable communication channel [36]. The response times can be up to 100ms [40]. Table 5.3 presented in *Teollisuusautomaation tiedonsiirtoliikenne: Turvaväylät* defines suitable reaction time requirements for machine control and other control connections. These requirements are significantly smaller than the response time reachable with standard Ethernet solutions.

Table 5.2. *Reaction time requirements [40].*

	Movement control	**Process control**	**HMI**	**Production management**
Delay or update interval	< 1 ms	100 ms	< 1 s	1s
delay variation	1 μs	5 ms	(predictable)	

One of the most important requirements to meet using safety versions of common industrial grade Ethernet solutions is to reach these response time requirements. The safety protocols are mainly built on top of industrial Ethernet solutions. The implementations are white channel solutions, which mean that standard industrial hardware can be used and only the software layer realizes the safety functionalities. However a widely standardized hardware base and easy connectivity to internet may raise serious security issues, which also have to be dealt with. The protocols may feature following properties to enable deterministic behaviour [40]:

- Transmission timing limits.
- Detection of wrong message arrival time.
- Detection of wrong message addresses.
- Watchdog.
- More precise self-diagnostics.
- Starting safety functions after detecting failures.

Different real time implementations usually divide the usable bandwidth for machine control and other traffic such as diagnostics, data logging or internet connectivity.

Teollisuusautomaation tiedonsiirtoliikenne: Turvaväylät by Matti Sundquist list four different methods for diminishing system response times [40]:

1. Using smaller networks with less equipment decreases the response time up to 20ms. 90-100% of the complete bandwidth is available for other traffic.
2. Using UDP instead of TCP connection enables response times of approximately 10ms. 90-100% of the complete bandwidth is available for other traffic.
3. Using mac addresses instead of IP-addresses enables response time of 1ms. 90-100% of the complete bandwidth is available for other traffic.
4. A special OSI model 2nd layer implementation on each of the equipment enables machine control eligible 0,2ms response time, but only 1% of bandwidth can be used for normal TCP/IP traffic thus slowing additional logging or monitoring capabilities.

As the amount protocols on top of industrial Ethernet is vast, so is the availability of different safety protocols. EtherCAT protocol has SIL3 certified Safety-over-EtherCAT, white channel –based CIP Safety has been SIL3 certified for Ethernet/IP and Sercos III protocols and PROFIsafe is a SIL3 option for safety PLCs or normal PLCs with combined safety features running PROFInet.

5.4 Wireless

Wireless control of machinery can be implemented with various methods: radio signal, infrared signal, micro waves, ultrasound et cetera. However the most common medium, radio link is also the most prone to bit errors according to Tapio Siirilä in Figure 5.2. These errors may be due to bad weather, strong electromagnetic disturbances, steel structures interfering signal transmission or other devices transmitting on the same frequency. These disturbances decrease device availability, induce delay in transmission, limit the transmission range or can overload the whole channel if multiple devices use the same channel. [36, 40]

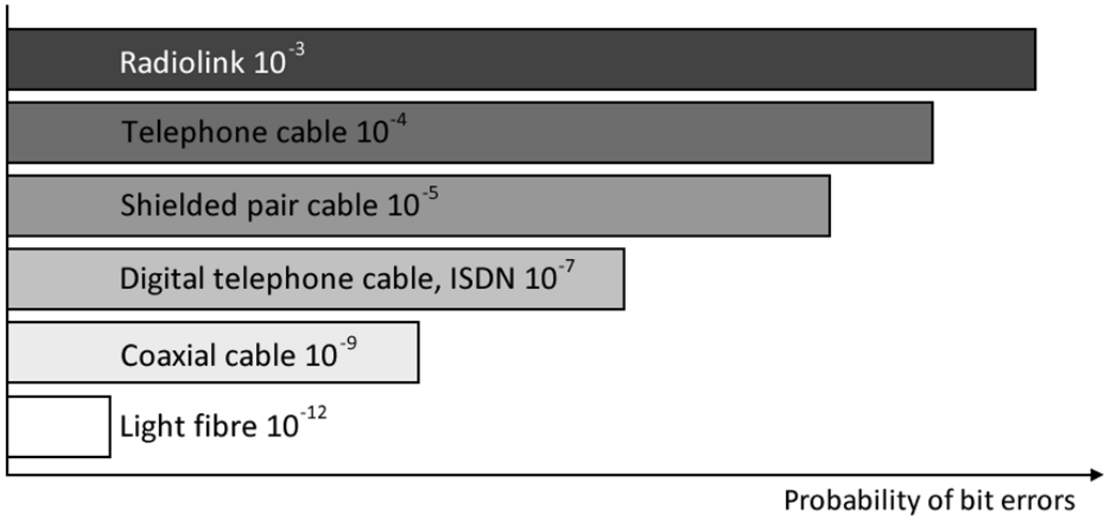


Figure 5.2. Probabilities of bit errors depending on the communication medium [36].

Most of these issues relate mainly to availability, but can also be connected to system safety and security. Safety features are mainly realised using the same methods as in wired safety protocols, but the system has to be more aware of its connectivity to the external controller unit. It has to react to signal loss or failure to receive a message in a safe way. As the communication medium is as prone to bit errors as it is, proper methods for detecting faulty messages and messages with faulty addresses have to be implemented [10].

System security features should be taken into account to prevent illegal connections to the machine. These connections may come from intentional intrusion to system or from other devices communicating on the same frequency. Wireless machines without any user identification can accidentally receive messages meant for other machinery which may lead to unforeseen consequences. User identification can be realised by allowing connections only from previously listed safe addresses. [10]

International standards give some guidance on safety issues related to wireless applications. EN 50159 describes different failure types and defensive methods to prevent these failures, IEC 61508 can be used to determine the required SIL level of the planned connection type. EN 60204 includes methods for revealing different connection failures.

6 IMPACT OF SAFETY REQUIREMENTS ON DESIGN PROCESS

This chapter handles various issues related to the development process of safe software applications. Different redundant architectures, lifecycle and process models, programming languages and coding rules are introduced and explained.

6.1 Hardware safety

Most of the failures in machine automation systems come from components which have moving parts, or are interconnected with the moving parts of the machinery. Table 6.1states that up to 78% of total amount of failures occur in sensors and actuators, and the rest occur in stationary equipment such as wiring or processing units.

Table 6.1. *Division of failures on machine automation components [36].*

Component	Share of all safety-related failures (%)
sensors	48
actuators	30
I/O	15
wiring	5
CPU	2

While selecting safety equipment for a safety-related system, components of small failure frequency and a reliable manufacturer should be favoured. It is however important to take the aimed level of safety into consideration. The SIL claim limit is determined by the lowest level component as stated in Chapter 3.2. Because of the higher price of higher level components rise according to their SIL capability there's no point in using a component if it doesn't contribute to the overall safety. As it is stated in ISO 13849, the requirements of category B functions should take the operating environment and conditions into account in component selection process. The components should function reliably in all the temperatures and moisture levels the machine is meant to function in. The positioning of the components in regard to heat, vibration, serviceability, cleanliness and physical contacts affects crucially to the reliability of a component. Because the previous factors affect the wear and tear of the components, the mission time (T_M) should be considered in the hardware and service design phases.

If the functionality of a safety function depends only on one component should it be a sensor, cabling, control logic code, actuator or such, could a single failure or incorrect functionality lead to a loss of the complete safety function. The architectural requirements on the higher performance levels and safety integrity levels require redundancy in all safety-critical components. Figure 6.1 clarifies which SIL levels can be reached with different redundancy-, architectural and diagnostic coverage combinations.

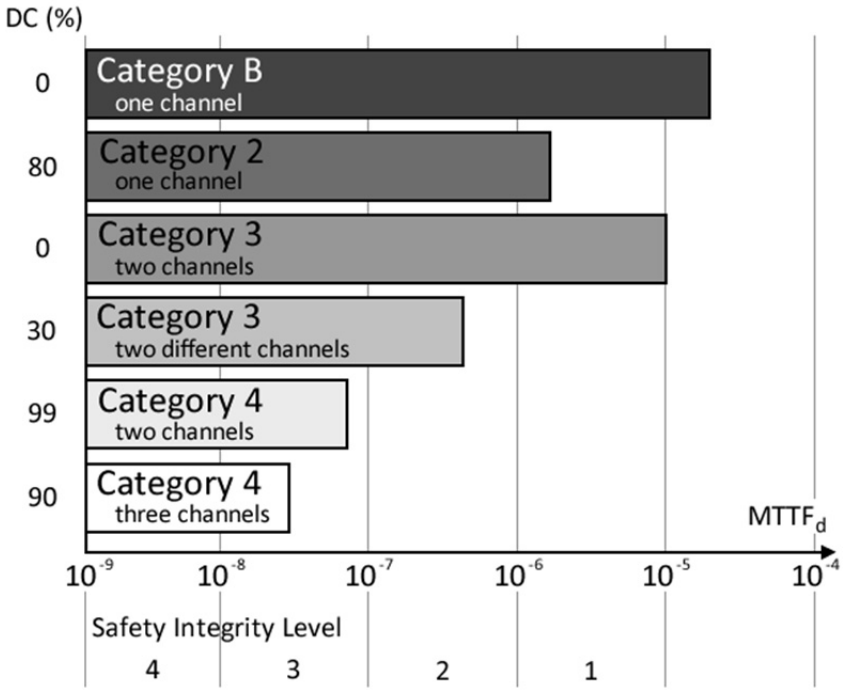


Figure 6.1. *Effect of redundancy and architecture categories on safety integrity levels. [36].*

Two main types of redundancy can be named: serial redundancy and parallel redundancy. These structures are based on the idea that a single signal or functionality is processed on separately in two or more occasions. It is recommended that the functionalities of these operations are based on different basic ideas and are implemented with different techniques and separate people. These procedures are to mitigate the risk of systematic design-based failures and bugs. However this multiplies the amount of work and number of people required for the design process and to ensure independence between the blocks.

Figure 6.2 depicts serial redundancy with blocks A and B. Serial redundancy occurs when two or more blocks are connected one after another (in serial). In the pictured structure block A processes the signal first, after which the same signal proceeds to block B where the signal is processed and the result of block A and B are compared. If the results differ, one of the blocks is working incorrectly and the whole result is rejected. Serial redundancy can be understood as a logical AND-gate. It is used to ensure the

correctness of information, but even the failure of a single block cripples the whole structure.

Blocks C and D in Figure 6.2 are redundant in parallel. The signal is processed in both blocks simultaneously, so failure in one block doesn't cripple the whole structure. Parallel redundancy can be seen as a logical OR-gate. Parallel redundancy reduces the risk of a dangerous failure in the system. For example if each block in Figure 6.2 has a failure frequency of 1/100, has the parallel redundant system a complete failure frequency of 1/10000 and the serial redundant system still the same 1/100.

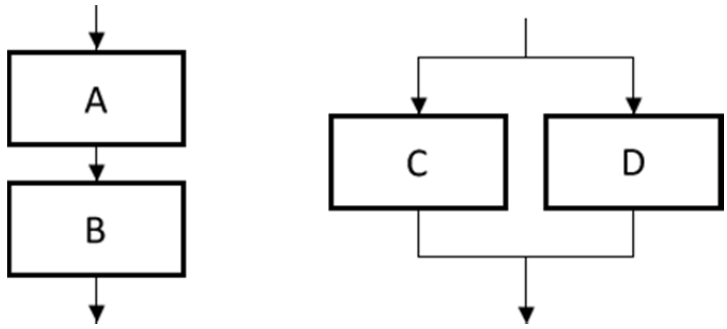


Figure 6.2. *Serial redundancy (left). Parallel redundancy (right).*

These architectures can also be combined, when the parallel redundant system has serially connected blocks in both its branches. Verified processing results and reliability in case of failures are both reached through this structure. This quadruple system is however very expensive and possibly too heavy architecture for use in mobile machinery.

Two ways of recognizing the failed device are presented by IEC 61131-6. If the used devices don't include failure detection features such as self-diagnostics, incorrect behaviour can be revealed by voting. These architectural methods are designated with an abbreviation MooN, M out of N. It describes a function consisting of N number of channels which requires at least M functional channels for normal functionality. Comparisons are carried out in the control device such as a PLC. If M channels do not agree on the result, the fault tolerance limit has been passed and the function will not be executed. The fault tolerance can be computed with a formula N-M+1. [13]

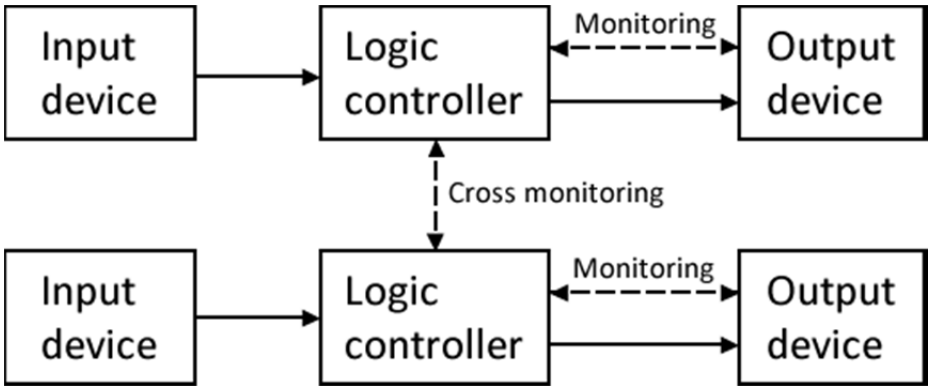


Figure 6.3. *1oo2D system for a Category 3 system [30].*

Faulty sensors and actuators can also be detected and isolated from the function with proper diagnostics. This method is designated with an abbreviation MooND, where 'D' stands for diagnostics. Figure 6.3 depicts a function which achieves the requirements of category 3 of ISO 13849. This function conforms to 1oo2D architecture which outputs are monitored and a faulty branch can be isolated from the function leaving the functional branch operational. [13]

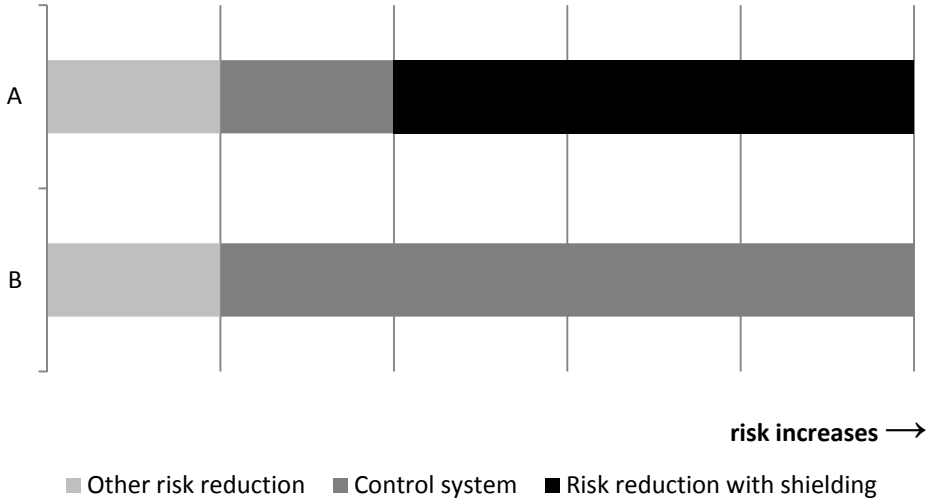
6.2 Software safety

Centralizing control system of machinery or automated system from multiple relays to a single physical device such as a PLC will mitigate the complexity of such machinery significantly. It also facilitates service and lowers service expenses due to smaller amount of components and hardware. Centralization also reduces the size and weight of the control system reducing the operating costs and energy consumption of for example mobile machinery. Making updates and changes to a control system are easier in software based control systems than in hardware-based systems. Software changes can be made in minutes even remotely, while hardware changes require specific tools and components.

On the other hand software-based control systems pose other types of threats. Making changes to software-based systems may be easier to execute, but side-effects from these changes may be hard to detect. Therefore proper effect analysis and version control are required by design process models. Large software-based control systems may also be harder to troubleshoot if documentation isn't comprehensive enough. All failures in software are inherently systematic in nature. They are caused by the way the software is conceived, written or compiled, not by its use or wear. Therefore the time consumed by system design and testing must be greater than in hardware-based design processes.

Centralization sets great safety requirements for the control system. Failure of control system or only a part of it would lead to the loss of all safety-related and non-safety-related functions. The greater the amount of safety is based on the control sys-

tem, the greater the requirements are set for the safety of the control system as depicted in Figure 6.4. In practice it means that performance level e, category 4 or SIL3 control systems are required [37].



Option A Most of the safety functions are realized with solid shielding and other methods. Only few functions are realized by the control system.

Option B Most of the safety functions are realized with the control system.

Figure 6.4. *Composition of risk reduction methods [35]. The greater the percentage of control system affecting system level safety is, the greater the requirements for control system performance level or safety integrity level.*

All software modules and elements or tools may not necessarily have a critical effect on system safety and therefore do not need as precise safety analysis as safety-related elements or tools. IEC 61131-6 determines software criticality on three levels: C1 - Interference free, C2 – Safety relevant and C3 – Safety critical. [13]

C1 – Interference free Software has only read-only interfaces with elements of criticality 2 or 3. For example debuggers and verification and test tools which are verified by the user only.

C2 – Safety relevant Combination of a deviation from normal functionality and a failure of another element may cause an unsafe situation. For example elements that are required to achieve a SIL-level and implement self-tests, compilers, software engineering tools with automatic executable code generation and verification and test tools without output verification can be rated C2.

C3 – Safety critical A single deviation from normal functionality can cause an unsafe situation. Such elements are those that are required to achieve a SIL-level and implement self-tests.

Literature presents multiple ways of handling with software integrity. Methods can be divided into two segments: avoiding failures and defensive design and programming. Avoiding failures includes methods of validation and verification, structural management and comprehensive documentation, quality control and applicable software change management. Defensive design and programming focuses on defining and using applicable tools and programming languages, modular and structural programming and use of pre-verified program modules, use of coding rules and detecting external failures. These aspects will be considered in the following chapters.

6.3 Process models

Safety-related standards describe different process models to aid with development process of safety-related machinery. Each process model has some differences between each other, but the main concept remains the same as with non-safety-related process models in which the process is run on basic sequence: specify-design-implement-test. This chapter will handle the process models presented by IEC 61508, EN 62061 and ISO 13849.

6.3.1 V-model and design lifecycle model according to IEC 61508

IEC 61508 sets out a generic approach for a few safety process models for developing systems comprised of electrical, electronic or programmable electronic elements. V-model (Figure 6.5) is a process management tool for software engineering. It is used as a design and verification tool in programmable system projects. The main idea in the model is to specify the software first on higher levels and then proceed to lower levels and module designs. Each state will be verified with the previous state after completion to ensure all the requirements are met. After finishing and verifying the lowest level descriptions, coding process is started.

The main advantage in the V-model is the systematic testing procedures. The process will execute testing of each module in reverse order comparing each state with corresponding design state. Development lifecycle ends with validating the product with its safety requirements. The model presented by IEC 61508 is only a framework for a V-model tool, and the model can be tailored to the needs of the company using it. [21]

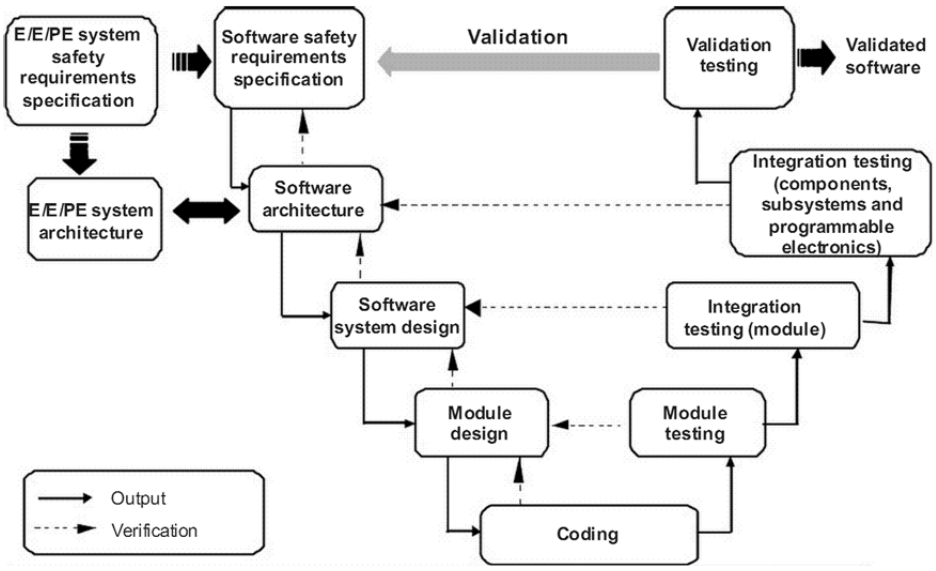


***Figure 6.5**. Software development lifecycle (V-model) [34].*

Another lifecycle mode presented by IEC 61508-1:2010 is a 16-part lifecycle model for designing and operating safety-related systems. This model can be divided into three main parts: analysis, realization and operation. Life cycle model is presented in Figure 6.6.

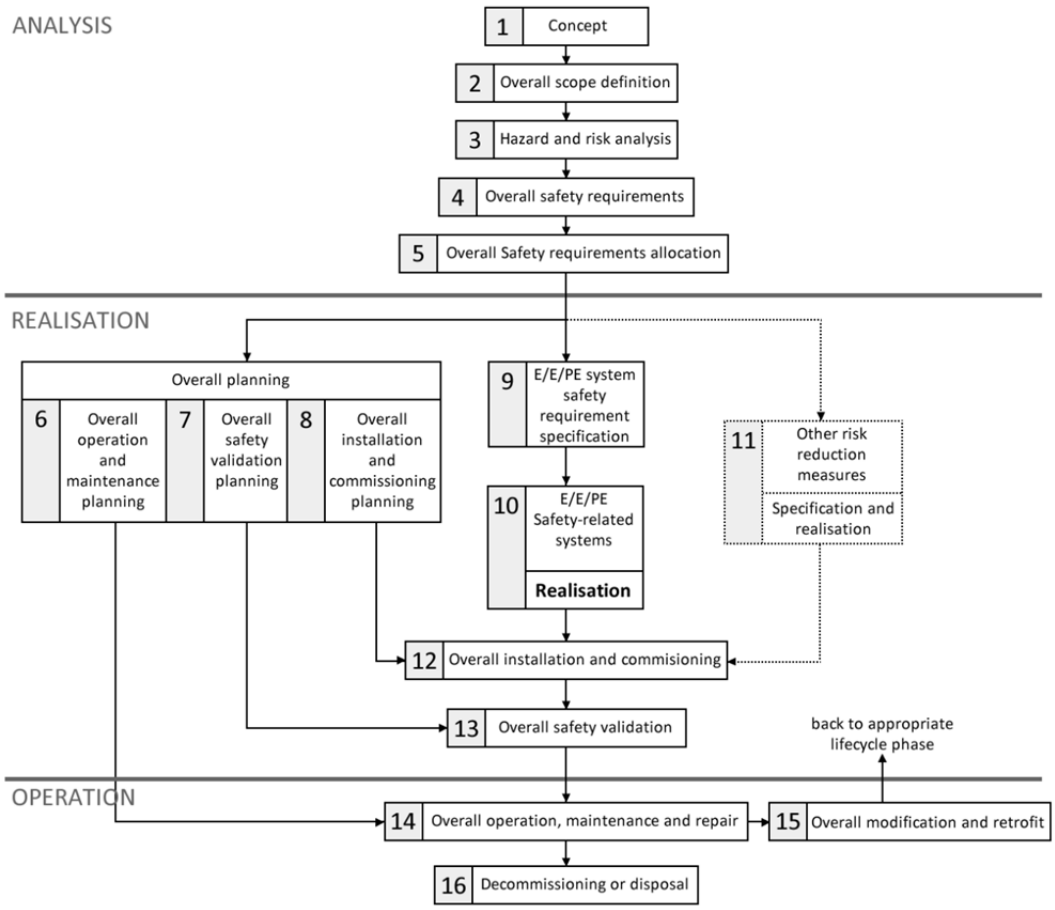


***Figure 6.6.** Overall safety design life cycle model according to IEC 61508-1 [34].*

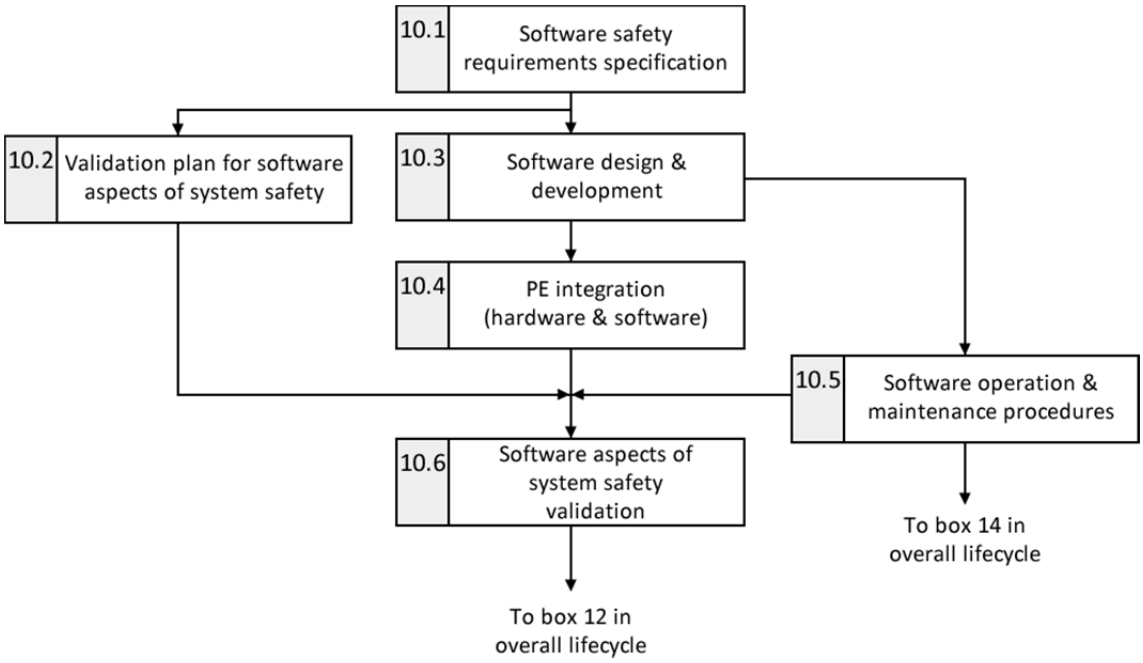


Figure 6.7. *Software safety lifecycle in realization phase (step 10) of Figure 3.10 [34].*

The analysis part consists of five steps in which risks are identified and corresponding safety requirements are developed. The next eight steps comprise the realization part. Here the safety requirements are used to create risk mitigation methods, which are used as a base for the system realization phase in step 10. The complete system is installed and commissioned, after which it is validated using various tests and analyses. IEC 61508-1 includes two separate implementations for realization phase in step 10: system safety and software safety lifecycle model which is presented in Figure 6.7. The system safety lifecycle forms the technical framework for IEC 61508-2, which handles mainly electric, electronic or programmable electronic (E/E/PE) based systems. The software safety lifecycle model forms the technical framework for IEC 61508-3 concentrating on software safety. Both of these models will be used in case a complete E/E/PE safety-related system is built.

The third and last part of overall safety lifecycle model consists of three steps which consider the operations that are carried out after the system has been taken into use. Such are maintenance and repair, modifications and decommissioning. [5, 21]

6.3.2 Control system design phase diagram according to EN 62061

Figure 6.8 presents a design process diagram introduced by the standard. This diagram can be used to mitigate system risks and design a safe control system using subsystems and modular function blocks. It uses the identified risks acquired from risk assessment as a base for the process, produces according safety functions, defines safety integrity levels for each safety-related control function and provides documentation during the design process.

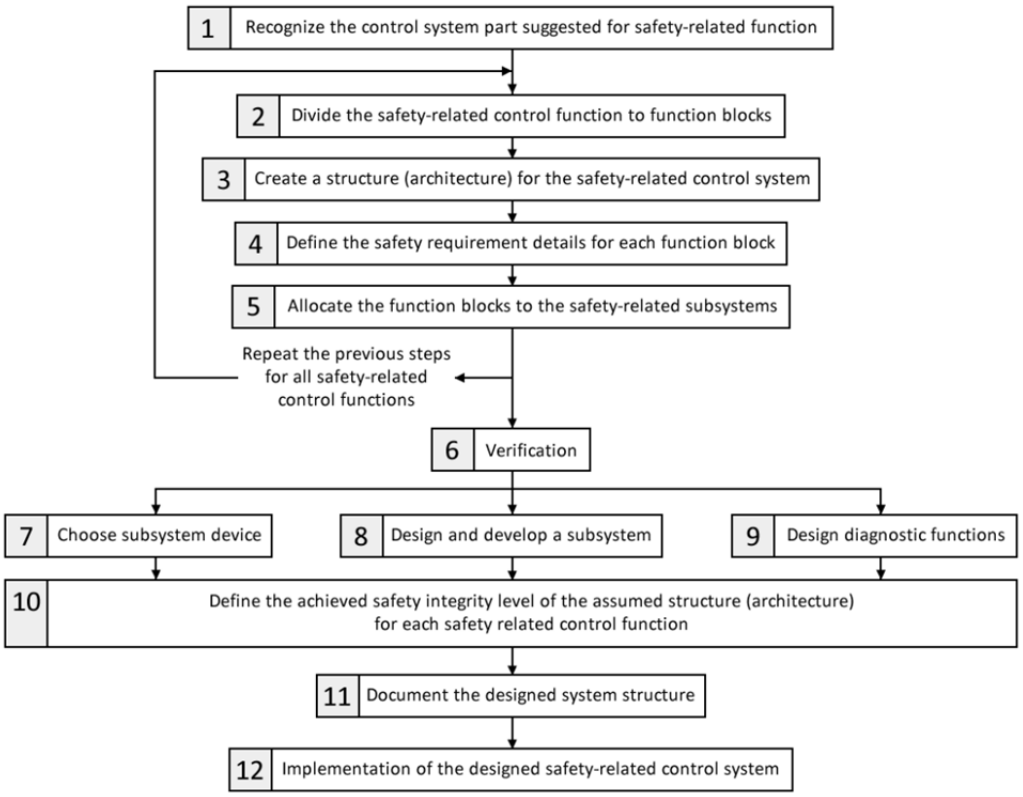


Figure 6.8. *Control system design phase diagram according to EN 62061 [36].*

In step 1 risk assessment results are used to recognize the control system's parts that will be addressed by safety functions. In steps 2-6 these control system parts will be designed safety functions with proper documentation according to the risk assessment. Safety functions will follow modular design principles using specific function blocks. Modular design principles will also be used to construct a clear structure/architecture for the whole safety-related control system using these safety functions. After verifying the structure and safety functions in step 6 other system subsystems and diagnostic functions will be designed in steps 7-9. In step ten will all the safety-related control functions be defined a safety integrity level. In the last two steps will the whole control system structure be documented and the design phase can be completed. [36]

6.3.3 Safety-related part design process model according to ISO 13849

The safety design process model presented by ISO 13849-1 is bound together with risk analysis standards ISO 12100-1 and ISO 14121. At first it analyses the system according to the analysis standards after which the risks are handled with primarily on the hardware side. If the remaining risks cannot be handled with hardware, they are analysed and processed further with a separate process model meant for control systems (Figure 6.9). The safety design process model is iterative, so after all identified safety functions are processed, an another analysis on the system is made to detect any remaining risks or if new risks have been generated on the system. [30]

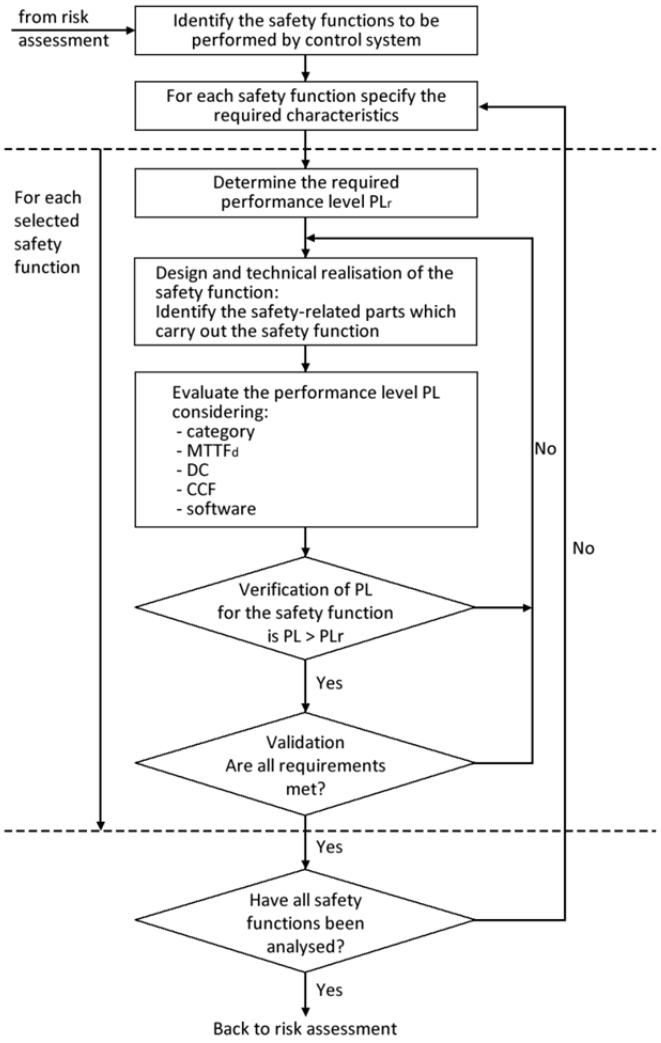


Figure 6.9. *Iterative safety-related part design process model according to ISO 13849-1 [30].*

The process model for designing safety-related parts of control systems starts with identifying all unsafe situations connected to the control system. Each situation is defined its safety requirements to mitigate their risk to a tolerable level. In the iterative phase of the model each of these functions will be defined a required performance level (PL_r) according to the risk it mean to mitigate. Level 'a' has a low impact from the control system; Level 'e' has a high impact. A method for defining PL_r is described in Figure 3.1. Each safety requirement will be designed a proper safety feature with technical implementation plan considering all the relevant parts which carry out the safety function. The formed safety function is evaluated taking all the methods described in Chapter 3.1 accompanied with software development rules describer later in Chapter 6. Performance levels of the safety-related functions are verified according to the defined PL_r and validated with the safety requirements specified during the process. After all the safety functions have been handled, is the whole system re-analysed for newly generated hazards. [30]

6.3.4 Agile models

Up until recent days, safety-related software projects have been quite small and projects have been able to be implemented through traditional process models. However, projects grow and the need for agile methods rise as does the need for standardized methods for agile processes. These methods facilitate gradual development in case of the specifications are updated during process and continuous assessment of risks. These methods emphasize verbal and literal communication between different quarters which may become crucial with projects of multiple technology areas. As normal "traditional" safety-related process methods require great amounts of documentation in the form of change authorizations, extensive specifications and test reports, agile methods significantly increase the amount of required documents.

Jani Paalijärvi states in his Thesis *Development of Safety-Critical Software using Agile Methods* [24] that the large amount of produced documentation doesn't prevent the use of agile methods. One only has to select a suitable process model according to existing company models and policies. Factors that affect the selection process according to Paalijärvi are the amount of personnel, severity of possible failures, company size, culture and the dynamicity of the project. Agile methods are suitable for designing safety-related software, only orderliness, care, documentation, clearness of architecture and implementation and traceability need to be given more attention. [24]

According to VTT research *Ohjelmaturva* released in late 2011 [37], the main requirement of produced documentation is its evidential value. This means that somewhere, somehow there needs to be evidence that some safety- or non-safety feature was planned, assessed and authorized according to the appropriate safety standards. Certification bureaus accept all kinds of evidence of these documents; they can be formal written documents, photographs, videos, meeting minutes, hand-written notes and such. Safety standards do not give much information on how to implement agile methods with safety-related projects, so this field of research has grown its interest globally.

6.4 Documentation

Comprehensive documentation is an important part of developing manageable and serviceable safety-related software. Thoroughly implemented documentation enables effective backward and forward traceability, creates a solid base for service and user manuals and enables safe modification of software. Documentation may easily be left on a lower priority on projects with tight schedules, and it may not be kept up-to-date all the time, but projects with safety features need comprehensive documentation during its whole lifespan.

Safety standards provide information on issues that should be handled on project documentation. No formal format is required, and important information can be presented in text, diagrams, figures or even meeting minutes. The importance of proper documentation comes from the evidential value of the documented information. It re-

veals the motivations and reasons for made decisions, gives comprehensible information on system properties and gives evidence in certification process that all risk sources have been handled properly. An unambiguous specification restricts different interpretations of system breakdown. A comprehensive specification doesn't leave any unspecified aspects in the system that could lead to safety black areas and unhandled risks. [30]

An important part of documentation on safety-related projects is risk assessment documentation. ISO 13849 require that it should include all identified risk factors and all dangerous situations they could lead to. Chosen measures to remove or diminish probabilities of identified risks and goals that are aimed with these measures need to be determined. If the risk cannot be completely removed, residual risks need to be documented. The risk assessment made may not give a positive result on all use environments and tasks, and such deviations from original plan and original intended use need to be reported and documented. Different sources of information considering for example failure probabilities for components may give different information, so either the used sources need to be reported or more preferably one should use only one source of information for failure data. ISO 13849-2 also presents a table for different document types that are required by safety categories (Table 6.2). The amount of different documents required rise according to the aimed safety integrity level. The work needed to produce the required amount of documents in the higher safety integrity level projects is a major factor that leads to higher project costs. [30]

Table 6.2. *Documentation requirements for categories [30].*

Documentation requirement	**Required in cat.**				
	B	**1**	**2**	**3**	**4**
Basic safety principles	x	x	x	x	x
Expected operating stresses	x	x	x	x	x
Influences of processed material	x	x	x	x	x
Performance during other relevant external influences	x	x	x	x	x
Well-tried components	-	x	-	-	-
Well-tried safety principles	-	x	x	x	x
The check procedure of the safety function(s)	-	-	x	-	-
Checking intervals, when specified	-	-	x	-	-
Foreseeable single faults considered in design and the detection method used,	-	-	x	x	x
The common mode failures identified and how prevented	-	-	-	x	x
The foreseeable, single faults excluded	-	-	-	x	x
The faults to be detected	-	-	x	x	x
The variety of accumulations of faults considered in the design	-	-	-	-	x
How the safety function is maintained in the case of each fault(s)	-	-	-	x	x
How the safety function is maintained for each combination(s) of faults	-	-	-	-	x

EN 62061 appendix C states that on top of normal requirements on the system requirements specification, safety-related project's specifications should include information concerning safety-related hardware such as sensors, actuators and other equipment. Performance capabilities such as memory capacity, response time requirements and other time-related restrictions and their uncertainties for the control system and safety functions need to be specified to ensure that the system meets the performance requirements. Description of self-diagnostics such as watchdog, I/O monitoring and memory error correction methods are also required by EN 62061. [32]

As the amount of required documentation grows according to the desired safety integrity level, a common information management system will come handy to keep all documents updated and available for all personnel involved. Some interviewed local companies had taken to use or were planning to take use a common safety knowledge base or a database in which all common and project-related documents will be added. These knowledge bases or databases could be based on common information (for example standards) and company experience and they should include at least:

- Design guidelines and rules
- Generic checklists for testing, implementation and verification
- Coding rules and standards
- Common company practices
- Document templates and manuals
- Project specific areas for documents

Change management is a project phase which creates a lot of different types of documents. Project specifications are seldom completed when the system design and implementation phases are started, so an efficient change management system is important to any company. Therefore requirements stated by safety standards for change management are handled in a separate chapter.

6.5 Change management

An effective change management system gets more and more important as the software ages. If the application and machine has a long lifespan it is quite probable that it's working environment, materials processed, its components or actual use changes in some part of its lifetime. As the machines lifespan be tens of years, the original designers' employment in the manufacturer company may not be as long. Therefore an up-to-date documentation and a detailed change management process are in its place. Changing the code on-site without any kind of impact analysis of the change produces easily new bugs or design flaws. Software is so easy to alter, that wider effects of the change are not given enough thought and modifications may cause critical failures in unintended parts of the system. [36]

Each time a change needs to be done on safety-related software a change plan needs to be done. The plan includes reasons why the change is needed, and impact analysis on all the parts that this change may have influenced and decision and authorizations for doing each change. Authorizations need to become from a person responsible of the system safety. Implementing the changes can only start after the authorization has been received. [32, 34]

After the change has been implemented all the functions that the change has affected according to the effect analysis need to be validated and verified and meeting the safety requirements has to be checked. The code made for the change itself need to fulfil the same safety requirements than the rest of the software. Also the need to redo some lifecycle states needs to be analysed. After the change all the documents that may have been affected by the change need to be updated. [32, 34]

A change report is made based on the change plan. According to EN 62061 the report shall include at least:

- Risk factors that this change can have an effect on.
- Description of the change request.
- Motivations for the request.
- Decisions and authorizations for doing each change.
- Impact analysis of the change.
- List of all of the actions done during the change process.
- All responsible personnel or organizations involved.

Change analysis process need to be implemented every time safety-related machinery is affected by [32]:

- Change of safety requirements.
- Conditions of actual use.
- Incident or accident experience.
- Change of material handled.
- Physical or use case changes.

In the time of agile development methods a functional and fluent change management system will come handy. Agile software development produces great amounts of documentation every time a development cycle is run. Therefore systems depicted in chapter 6.4 are especially useful while following an agile process model.

6.6 Failure types

Figure 6.10 depicts the proportions between different types of failure sources on machine automation devices according to VTT research [35]. More than half of the amount of detected failures derives from malfunctioning hardware such as broken sensors or actuators. One quarter derives from design and implementation flaws and the rest from external factors or user activity failures.

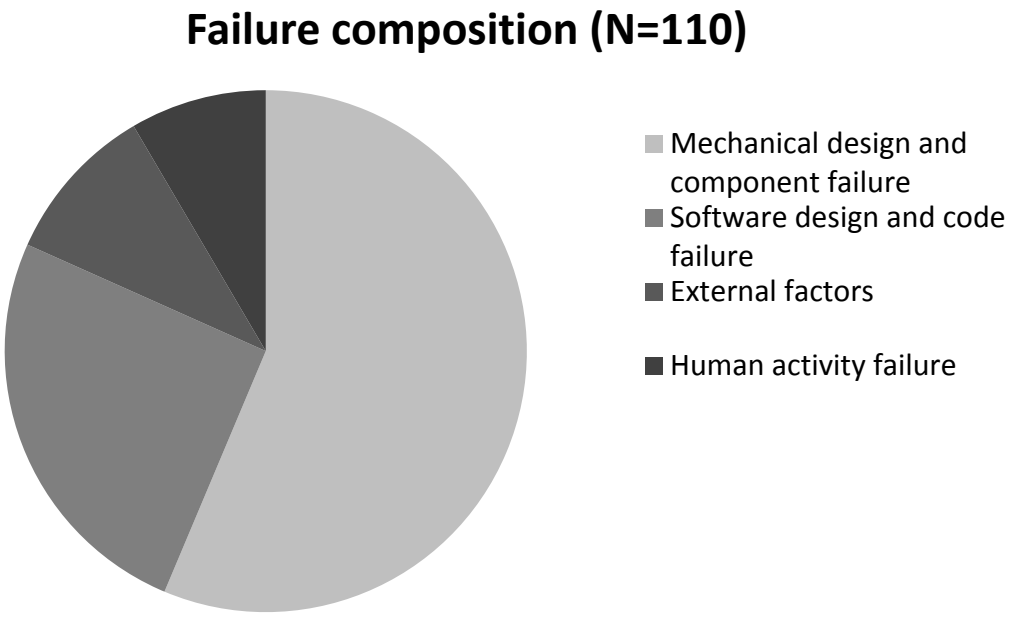


Figure 6.10. *Failure composition according to VTT research [35].*

Program code itself hardly ever fails (however a powerful shortwave radiation can inflict bit-level changes in the hardware), and usually in case of a software-based failure it derives from a human error, and in more precise, a design or implementation-based failure. In software engineering projects most of the bugs remaining in the final and released product are due to requirement and design phase deficiencies or errors. Bugs are created also during the programming phase, but these kinds of failures are easy to detect and fix before customer delivery. [23, 37]

It can be said that the expenses of repairing a malfunctioning program code increases as the project proceeds. In requirement specification phase fixing a single erroneous requirement doesn't require much work, but fixing a near complete program code to meet this same requirement change might need many times more time to fix and may affect other software components as well. It may also require re-designing the whole software architecture and if not thoroughly analysed, it may lead to severe safety risks through new bugs or inconsistencies.

Incorrectly executed software changes are a great source of failures. Implementing changes is required every time an old error needs to be fixed or new functionality is to be added to an application already released or being late in its development process. All changes affecting safety-related components need to be analysed according to appropriate safety standard and properly documented as depicted in chapter 6.5.

Another failure type that is hard to detect are common cause failures. CCF can be caused by application or function using the same malfunctioning software or hardware components, erroneous compilers, defective requirement specification or programmers with the same programming technique. *Ohjelmaturva* final report [23] states different methods for avoiding common cause failures:

- Separating physically different critical components.
- Using diversified components.
- Use of well-known design- and operating experiences.
- Use of analyses and reviews.
- Assuring good competence and education for designers and implementers.
- Taking operating environment into account (operating systems and hardware).

Software failures are usually of systematic nature, which means that with the same starting values the application leads in the same way to the same erroneous state. The inverse of a systematic failure is a random failure which is a single erroneous event in a series of functions executed with the same starting values. EN 62061 and ISO 13849-1 list different methods for avoiding systematic failures [30, 32, 37]:

- Use of a systematic implementation process, such as a lifecycle model.
- Ensuring compatibility between different modules and elements.
- Modularization and limiting the module size.
- Limiting the amount of different structures to:
    - sequences,
    - loops,
    - choices and
    - subroutine calls.
- Using small design diagrams in design phase.
- Use of common coding, naming and documentation rules.
- Using manufacturer information and guides.
- Use of computer aided design tools and simulation.
- Limiting the amount of iterations used or the depth of recursion. Iteration can simplify the program code or lock the system up if misused [2].

Some undetected failures may still be left in the released application, so its functionality needs to be monitored during use. Among others ISO 13849-1 lists a few methods for handling systematic failures during runtime [30, 37]:

- Monitoring program execution sequence in case of incorrect execution order.
- Detecting defective system clock.
- Preparing for communication disturbances.
- Proper actions during power distribution problems:
    - Voltage (over-, under-, variation of voltage and voltage interruption).
    - Hydraulic fluid, air pressure or other power distribution problems to actuators.
- System reaction to hazardous events before they occur.
- Use of self-diagnostics.
- Oriented mode of failure in case of jamming, loss of energy or other.

Bus-based communication systems have various failure types, induced by external interference, message collisions and timing errors, but they are not handled in this thesis' scope. Different failure types in various communication protocols can be found on *Teollisuusautomaation tiedonsiirtoliikenne: Turvaväylät* by M. Sundqvist [40].

6.7 Programming languages

Used programming language and compiler affect software safety and easiness of generating safe applications. The languages defined by IEC 61131-3 can be roughly divided into two groups: graphical and textual languages. Structured text (ST) and instruction list (IL) can be considered textual as sequential function chart (SFC), ladder diagram (LAD) and function block diagram (FBD) as graphical languages. IEC 61131-3 prefers use of graphical languages to textual due to easier visual diagnostics and monitoring capabilities [2].

Standards IEC 61508-3 and EN 62061 define two main classes of languages for safety-related applications: full variable language (FVL) and limited variable language (LVL). FVLs are used in "normal", non-safety-related applications when LVLs can be used in developing safety-related applications. Limited variable languages are actually FVLs with their instruction set reduced by unsafe and hard to analyse structures and variables [2, 34].

When selecting a suitable programming language for a project, the selection should be based on the application policy or how the program could be diagnosed and serviced the easiest more than based on the hardware used. The lower level languages like Assembly belong to the latter group being hardware dependent. This makes porting the software to different hardware more difficult or even impossible. IEC 61508-7 recommends use of LVL languages due to their portability to different types of hardware and ability to restrict generating unsafe program code.

EN 62061 recommends that the language to be used should have features that facilitate use of abstraction, modularity and other complexity mitigating structures. The language should be able to illustrate its functionality and sequence, data flow between different modular elements and functionality of time-related functions. It should also support time limitations and information integrity checking. IEC 61508-7 states recommendations for the ability of use of exceptions and interruptions in real time systems. Object oriented languages are making their way to safety-critical software, but IEC 61508-7 doesn't yet make a statement considering which of these languages are recommended in its second edition. [32, 34]

Table 6.3 is a table about recommended programming languages for different safety integrity level systems according to IEC 61508-7. Abbreviation 'NR' stands for *not recommended* and these languages should not be used for implementing safety-related applications for the appropriate safety integrity. 'R' stands for "recommended" and 'HR' for "highly recommended". R-classified programming languages can be used

in programming safety-related applications, but their use requires stricter monitoring and validation that HR-classified languages. [34]

Table 6.3. *Preferred programming languages according to IEC 61508-7 [34].*

	Programming language	SIL1	SIL2	SIL3	SIL4
1	ADA	HR	HR	R	R
2	ADA with subset	HR	HR	HR	HR
3	Java	NR	NR	NR	NR
4	Java with subset	R	R	NR	NR
5	PASCAL	HR	HR	R	R
6	PASCAL with subset	HR	HR	HR	HR
7	FORTRAN 77	R	R	R	R
8	FORTRAN 77 with subset	HR	HR	HR	HR
9	C	R	–	NR	NR
10	C with subset and coding standard, and use of static analysis tools	HR	HR	HR	HR
11	C++	R	–	NR	NR
12	C++ with subset and coding standard, and use of static analysis tools	HR	HR	HR	HR
13	Assembler	R	R	–	–
14	Assembler with subset and coding standard	R	R	R	R
15	Ladder diagrams	R	R	R	R
16	Ladder diagram with defined subset of language	HR	HR	HR	HR
17	Function block diagram	R	R	R	R
18	Function block diagram with defined subset of language	HR	HR	HR	HR
19	Structured text	R	R	R	R
20	Structured text with defined subset of language	HR	HR	HR	HR
21	Sequential function chart	R	R	R	R
22	Sequential function chart with defined subset of language	HR	HR	HR	HR
23	Instruction list	R	–	NR	NR
24	Instruction list with defined subset of language	HR	R	R	R

IEC 61508-7 sets requirements also for the used compilers. An incorrectly working compiler inflicts systematic errors into the software which are impossible to detect from the source code. Because of this, compilers which functionality is not well known, or which are known to produce certain errors should be avoided. Small compiling errors can be allowed if they are identified and documented and can be avoided during use. Only well-tried compilers can be recommended for use with safety-critical software. [34]

6.8 Architectural methods

The final report of VTT's *Ohjelmaturva*-project emphasizes simplicity on the development process of safety-critical software. It highlights the significance of modularity, structural programming and the use of object oriented programming if applicable. Favouring the use of small software components can easily be justified; smaller software modules are easier to understand and test. Their functionality is easier to anticipate, and their re-usability grows as their area of functionality is restricted. Figure 6.11 shown also in the report depicts the amount of errors per function points in software as a function of function points. This figure shows how the testability and maintenance of a large application gets harder as the program size grows. [23]

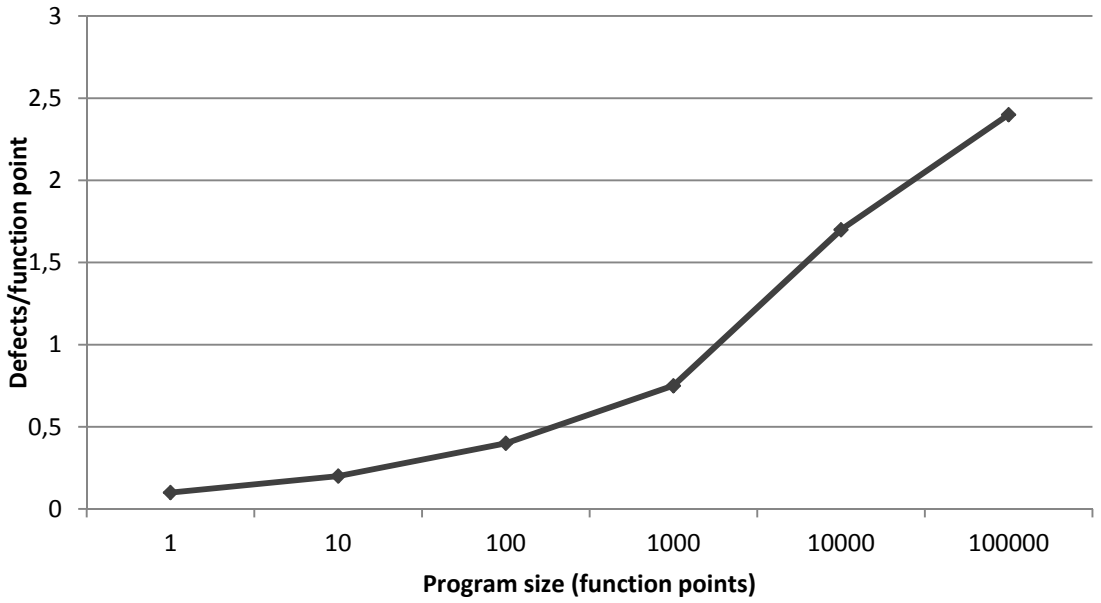


Figure 6.11. Number of defects to the amount of function points [23].

ISO 13849 recommends that a clear program structure is designed, in which all the different functional blocks and their functionality can easily be identified. This can be done for example by dividing the application structure to at least three parts: input, process and output. Each module should handle only the functions its name depicts and they should be set in a logical order according to the working sequence of the application. Each module and sub-module should be properly documented for its functionality. All the used functions and function blocks should be described whenever called. The program sequence need to be easy to follow and should not depend on jump instructions calculated on run-time. [12, 30]

It is recommended to use manufacturer verified and checked function blocks in controlling safety-critical devices to ensure their correct functionality. When using own safety-critical functions and function blocks it is worth noting a few suggestions stated

by ISO 13849 and *Ohjelmaturva* final report to keep the blocks simple and manageable: [12, 23, 30]:

- Use at maximum eight digital and two integer inputs and one output.
- Functional code should include maximum 10 local variables and maximum 20 Boolean equations.
- Function blocks shouldn't change global variables.
- Function blocks should be able to detect variable inconsistency.
- Error codes and proper descriptions should be used to distinguish different error states (also relevant for other parts of the program).
- Interfaces between modules should be kept as small and simple as possible.

Features and restrictions between function block and PLC interfaces have to be properly documented to ensure proper use and functionality of the given software. Used communication protocols, input and output frequencies, input and output voltage and current areas, I/O formats and use of reverse logic (is the signal normally high or normally low) should be clearly indicated in the project documents. [32, 37]

The final report of *Ohjelmaturva*-project and IEC 61508-7 recommend also other architectural methods to develop safety-related software: [23, 34]:

- Use stateless design: input always results in the same associated output.
- Avoid difficult structures such as recursion and dynamic objects.
- Backward recovery, which enables return to previous safe state or well-defined checkpoint in case of a failure.
- Use of graceful degradation: safety-critical functions are given priorities over normal functions in case of a failure to maintain safety.
- Utilize error detecting software (parity bits, CRC and others).

In appendix A of this thesis a table presented in IEC 61508-3 lists different architectural methods recommended for different SIL. Appendix B represents a similar table presented in EN 50128 for SWSIL.

6.9 Coding rules

Certain coding rules should be followed every time safety-related code is to be implemented. An agreed company-level set of rules not only make the code easier to read for everyone in the company, but it also standardises code structure and facilitates verification and validation processes. Common non-safety coding rules are widely available for different languages, but they aren't sufficient for generating safety-related code. Safety standards do not take non-safety-related software coding rules into account and specify only rules applicable for safety-related software. Therefore it is possible for companies to append these safety-related suggestions into their existing set of rules. There are also commercial coding rules available that include also safety-related requirements. One

example for C-languages is MISRA C, which was originally developed for automotive industry.

ISO 13849-1 ANNEX J, IEC 61508-7 and EN 62061 list different suggestions for a set of safety-related coding rules. Below is a set of rules selected from these standards and own observations on how to simplify and unify program code.

Defensive programming:

- The effects of output signals should be checked in the actuator end [34].
- Existence and accessibility to the expected hardware is to be checked [34].
- Completeness of software should be analysed (especially after maintenance) [34].

Modular approach:

- Structuration should be implemented in a way that facilitates centralized variable updating [12, 30].
- Information hiding and encapsulation methods should be used [34]:
    - Preventing accidental or incorrect modification of critical or complex software.
    - Allow modification of hidden module without affecting the remaining software.
    - Use clear and simple interfaces.
- Data verification of parameters with assertions [34].
- Compose control flow: sequences, iterations and selection [34].
- Small amount of possible paths through the software [34].
- Simple relations between input and output signals [34].
- One entry and exit point in subroutines and functions [34].
- Avoid complicated branching and unconditional jumps (goto) [34].
- Avoid using complex calculations as the basis of branching and loop decisions [34].
- Well defined tasks for modules [34].
- Limited use of side effects not obvious from function name and its documents [34].
- Use of several levels of modules [34].
- Modules should be well documented (functionality and use) [34].
- Safety-oriented blocks should be separated from normal blocks.
- Safety-oriented blocks should be identified through a prefix (for example S_xxx).
- Call of non-safety blocks from safety-oriented blocks is prohibited.
- Call of safety-oriented blocks from non-safety blocks is limited to standard functions.

- Write access to safety-oriented variables from non-safety blocks is not permitted.
- Write access to non-safety variables from safety-oriented blocks is not permitted.
- Main program code should have only functions/function block calls in it. Plain logic code is not allowed.
- Same global function block should be called from only in one place
- Access to global variables from function blocks is prohibited.
    - Function block should only change the state of its own instance.

Code understandability and readability:

- Each variable should have an explicit mnemonic name and a description commented to it [12, 30].
- Unambiguous and meaningful naming conventions [34].
- The amount of common variables should be kept to a minimum [32].
- Limited number of rows per block (500).
- Limited number of characters per row (150).
- Use of symbolic names for numerals [34].
- Document why something was done instead of what was done [34].
- Document caveats and possible side effects [34].

Variables and I/O

- Activation or de-activation of any output should take place only once [12, 30].
- Use prefixes for I/O- & global variables.
- Safety-oriented global variables should be separated from normal global variables and identified through a prefix (for example S_xxx).
- I/O-addresses of safety oriented application parts and non-safety parts are separated into different ranges. Safety-oriented addresses come first to the lower addresses to facilitate use of smaller address values.
- The value of a global variables change only in one place.
- No address declarations or references in program code. Use variables.
- Only one purpose per variable. No re-use even if the previous purpose is no longer important.
- Prefer local variables to global variables.
- Favour simple data types (Table 6.4).

Table 6.4. Suggested data types for safety-related code.

Simple data types	
Keyword	**Suitable for safety programming**
BOOL	Yes
BYTE, SINT, USINT	Yes
WORD, INT, UINT	Yes
DWORD, DINT, UDINT	Yes
TIME, TOD, DATE, DT	Yes
STRING	Yes, in case a safety-oriented interface is used
LREAL, REAL	To a limited extent (prone to errors due to rounding errors)
Complex data types	
Keyword	**Suitable for safety programming**
ARRAY	To a limited extent (only with explicit range check before write access to fields)
STRUCT	Yes
Listing types	Yes
Subrange types	Yes
POINTER	To a limited range (no pointer arithmetic, use range check, new pointer value allocation at the start of each cycle)

Verifiability, testability and structure

Where the application software is to contain both non-safety and safety-related functions, the whole code will be treated as a safety-related code, unless independence between the functions can be demonstrated [32].

- Avoid use of dynamic variables and dynamic memory allocation [34].
    - Memory use cannot be verified by compilers or other off-line tools and it cannot be guaranteed that there is free memory or nothing is overwritten.
    - Failures can possibly be avoided with on-line checking during creation of dynamic variables or objects and checking if the memory is free before allocation. Memory should also be freed after the variable or object has been used.
- Limit the use of recursion and other forms of circular dependencies [34].
- Limited the number of iterations [34].
- Interrupts should not interfere with critical software parts [34].
- Limited use of states.
    - Favour stateless design.
    - State variable should be described only once per cycle.
    - State transitions should be encapsulated into a function block.
- Allow no compiler warnings.

Language subsets:

- Limited use of pointers [34].
    - Data type and value range to be checked.
- Limited use of C-like unions [34].
- Limited use of Ada or C++-like exceptions [34].
- No unstructured control flow in higher level languages [34].
- No automatic/implicit type conversion which may lead to loss of information [34].
- Limited or only documented use of compiler-specific features [34].
    - If these features are used nonetheless, rules should be applied and documented.

Good programming practice:

- Floating point comparisons use only inequalities instead of equality [34].
- Bracketing if operator precedence is not obvious [34].
- Catching of all situations (default in switches, else in if-clauses) [34].
- Coding guidelines should comply with known compiler errors [34].

Use of trusted/verified software elements:

- If previously used software library functions are to be used as part of the new design, their suitability for the design should be justified by evidence of satisfactory operation on similar environments and applications. Otherwise the function has to go through the same validation and verification procedures as the other new software functions do [32].
- Requirements for use of proven-in-use elements [34]:
    - The specification of the element has to stay unchanged.
    - The element has to have at least one year of service history.
    - The element should also work in current application.

The formed coding rules should be made commonly available throughout the company and make sure that all employees follow the rules. The document database depicted in Chapter 6.4 could again come handy in keeping all safety-related documents in one place.

6.10 Validation and verification

The amount of work required for validating and verifying of a safety-related application is relative to the coverage of the formed documentation. In the development process of safety-related software the significance of documentation is emphasized even more than in non-safety-related software. It is crucial in achieving a reliable and accurate verification and also in order to create plausible evidence for certification inspections.

Validation and verification are made based on the documents formed in previous process phases. Each function in the safety requirement specification needs to be veri-

fied, for which information and guidance can be found in EN 62061. To get a reliable and impartial result, the inspection should be performed by a person independent from the project. The method used for verification can have an effect on the verification result. Auditions and readings usually follow the programmers' way of thinking, thus finding errors may be hard. On the other hand, quality assurance assures correct project management, but finding errors is not guaranteed. [21, 32]

7 SAFETY-RELATED INTEGRATED DEVELOPMENT ENVIRONMENTS

Software developers have noticed the need for safety-related software development environments for PLCs. As many logic programmers might not be specialised in software safety issues and as the requirements presented by the law and safety standards are vast, comprehensive and spread around many standards and documents, all aid from software environments is well welcomed. Developing a safety certified development environment is a time-consuming process not only as the development process, but also in certification processes. Therefore the availability of different safety-related development environments is quite low and is focused on the biggest developers and most common communication protocols.

This chapter will handle CoDeSys Safety from *3S – Smart Software Solutions* and SAFEPROG with its safety runtime components from *KW-Software*. These solutions differ from each other with their requirements on the hardware, but on the software side the main principles are quite similar. Safety-related components and variables are visually and structurally separated from non-safety-related components and variables, compilers are well tested and certified, easily debuggable languages are used and user actions are restricted in a way that the user can implement code only in a controlled, safe way. These considered applications can be used to produce software up to SIL3 and PLe, but both manufacturers have also less restricted SIL2/PLd solutions.

IEC 61508 has some requirements for the programming environments. The programming environments to be chosen should facilitate modular programming, have translation verification and runtime parameter type- and overflow checks. It should encourage use of small and easily manageable modules, enable user access restriction to some program components, allow definition of variable sub-ranges and also include other error mitigating structures. All of these requirements were met by both of the considered development environments. [34]

7.1 3S CoDeSys Safety

3S CoDeSys Safety is a programming system extension for editing safety-oriented applications and configuring safety I/O-modules in the runtime system. The application is an integrated development environment (IDE) based on CoDeSys V3, and allows integration of safety- and non-safety control applications and I/O configuring in one project, unlike the KW-software SAFEPROG approach, which has separate applications for example control application and I/O configuration. Another major difference with

SAFEPROG is its ability to run on any manufacturer's central processing units (CPU) or OS that can run the normal CoDeSys runtime system making it suitable for manufacturers who have their own safety-PLCs, I/Os, drives or other hardware.

CoDeSys Safety can be used to produce control applications for systems up to SIL3 or PLe, which means that redundant hardware as dual CPU and 1oo2 dual channel architecture are required. The system supports shared safe communication buses which can include safety- and non-safety messages and can exchange data between safety PLCs and non-safety PLCs. At the moment supported communication protocols are PROFIsafe over PROFIbus, FSoE (FailSafe over EtherCAT) over EtherCAT and CIP-Safety over SercosIII. As the development environment is still new, the company states that other protocols may also become supported upon request [1].

The software approaches safe application coding with reduced functionality by disabling dynamic memory allocation and coding language selection. Only PLCopen defined subset of function block diagram (FBD) with basic and extended level is supported. This enables easy graphical debugging of safety-related code and lowers the learning curve with the need to understand only one coding language. The delivered pre-certified function blocks and user-defined safety blocks and -variables are separated visually from non-safety blocks and variables with different colours. Certified safety-libraries, -guides and -manuals are delivered to aid with the coding process.

Software architecture methods influence greatly on system safety, and in CoDeSys Safety these methods are used to reduce application functionality thus ensuring safe operation between safety- and non-safety-related modules. Safety-related function blocks are separated from global variables and other function blocks, which allows making changes on non-safety software without affecting safety-related part thus lowering the risk for common cause failures and unnoticed relations between different modules. The parameter and variable range and type checking is possible to detect incorrect function block inputs and variable assignments. Program and block size limiting through number of variables, number of FBD-networks and network size is optional and can be configured through application options. Unauthorized application altering can be prevented with password protection on project and function block levels.

CoDeSys Safety features a concept for system integration, testing and certification to aid with the verification process. Also debug-mode and project comparison features are included. The used compiler and runtime are certified for safety use.

7.2 KW-Software safety platform

KW-software safety platform is a collection of IEC 61508 –certified applications for developing and testing safety-related software and configuring safety equipment. The platform consists of six applications: SAFEPROG, SafeOS, SAFECONF, SAFEGRID and SAFEBtest. SAFEPROG creates a diversified software architecture which is continued in SafeOS, which is a 2-channel diversified safe runtime system on the safety

PLC. This 2-channel realises a SIL3 and PLe safety system, and one channel solutions for SIL2 and PLd systems can also be implemented. SafeOS requires specific type of processors for operation: ARM (Instruction Set V4.0), Intel X86, PowerPC. Other processor types are also available on request. PROFIsafe V2 over PROFInet is the only communication protocol supported by KW-software safety platform.

SAFEPROG is a safe software development environment for large and complex programs with communication systems. It supports graphical ladder diagram (LD) and function block diagram (FBD) languages for normal logic code and C-language for complex function blocks. It includes pre-certified PLCopen safety function blocks as does CoDeSys Safety. These safety function blocks and specific safety variables are separated from normal code with different colour. User actions and unauthorized modifications can be managed with user management, user action logging and access rights. Project and the PLC can also be locked with a password to prevent unauthorized access to the code altogether.

SafeOS is a diversified, 2-channel safe runtime that realises safe control system with SAFEPROG or SAFECONF. It supports integration of both safe and unsafe signals. It has memory error detection methods and diagnostic possibilities. SAFECONF is a SIL3 and PLe certified configuration application for medium number I/O devices without communication systems. It can be used to implement simple logic and certified function block to configure safety controls and drives. SAFEGRID is the smallest level configuration application. It is used to configure multifunctional safety relays, controllers or integrated safety functions on drives. No logic code can be implemented in SAFEGRID. SAFEBtest is an automated testing tool for testing function block made with SAFEPROG or SAFECONF. It generates test case steps using user given requirements thus reducing test effort and error sources. KW-software's SafeOS and SAFEPROG have been integrated and validated with the INTEGRITY RTOS and they can be used with virtualization environment [7]. [38]

7.3 Roadmap

During the writing process of this thesis CoDeSys Safety haven't been released and is still under TÜV (German technical supervisory agency) testing and may still change before release. 3S is going to release also a SIL3 and PLd runtime. It will run on a single CPU hardware and will feature full functionality. The runtime is a standard CoDeSys Control embedded runtime with additional safety features such as: diagnostics and memory error detection methods.

Applications for this runtime will be implemented with all language features enabled. Only a few restrictions will be presented by provided safety programming manual. Implemented code will be compiled with the standard compiler which is validated by TÜV and well-tried in extensive tests according to 3S. The system will require certified hardware and safe I/O-system and the system need to be completely validated.

8 CASE: SAFETY DESIGN PROCESS IN OIL EXPLORATION MACHINERY

Project to consider as an example project of safety-related design process is a mobile electro-hydraulic oil exploration machinery used in test-drilling promising oil wells. The machinery consists of three mobile units; a loading unit, a drilling unit and a control unit. The loading unit provides the drilling unit with drilling extension rods. Its functionality is limited to handling these rods and levelling the unit on rough terrain. The drilling unit has drilling, various winching and extension rod connecting functionalities. The machine doesn't have much automation in it, and its functions are controlled by two system operators from a two-desk control panel in the control unit.

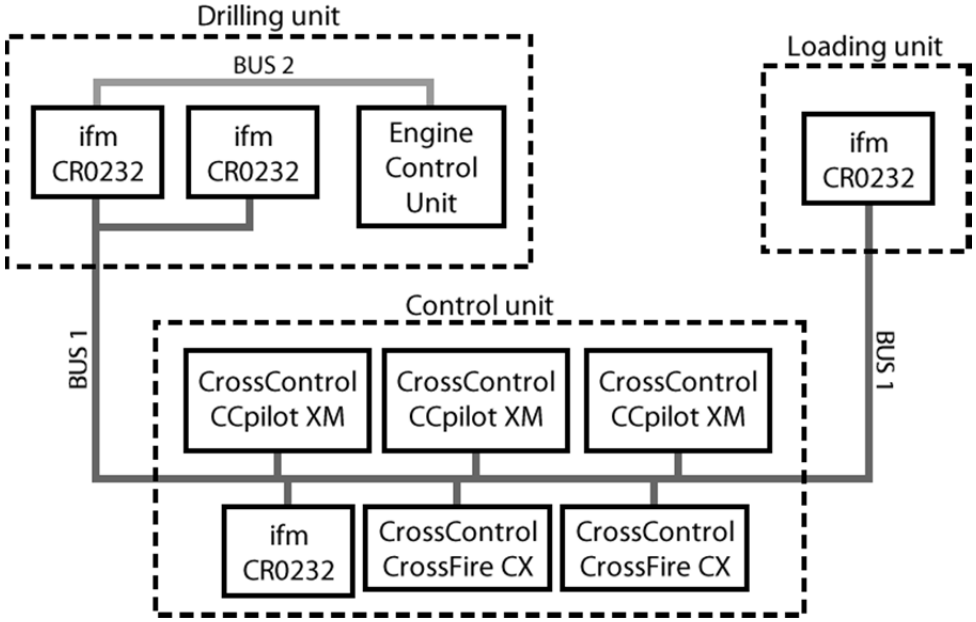


Figure 8.1. Oil exploration machinery PLC and I/O-module architecture

The machinery is controlled by ten different PLC and I/O-modules. Drilling unit includes two ifm electronics CR0232 PLC-modules [14] which control analogue and digital output signals. The ifm units are connected to other PLC and I/O-modules via CANopen bus indicated with BUS 1 in Figure 8.1. The unit also includes an engine control unit, which sends engine information via secondary bus implementing J1939 protocol (BUS 2 in Figure 8.1). The loading unit includes one ifm CR0232 module which controls drill rod loading functions with its analogue and digital outputs. The control unit includes all the input controls and three touch screens system control. One ifm CR0232 controls analogue joysticks with two CrossControl CrossFire CX modules [4].

The control unit include also three CrossControl CCpilot XM display computers [3] with touch screen capabilities which are used to display pressure gauges, certain valve states, system diagnostics and can be used to change system parameters.

At the moment the machinery is designed as a normal mobile machine automation system with some EX-certified sensors, and no safety standards considered in this thesis were used in the design process. The only feature implemented that could be considered either safety or reliability-related is that the controls are distributed throughout all the modules when normally all the controls are centralized on a single CCpilot XM display computer for its performance. However, if CCpilot XM were to be used as the centre of all controls and as the CANopen master which controls the bus communication single handed, crashing of the operating system on the display computer could paralyze the complete system and possibly leave the controlled valves in a position that could lead to a hazardous situation. Now when all the controls are distributed all over the system, the severity of a single module failure is restricted to its own area of control.

Distribution of control isn't however enough if one wants to make the machinery safe enough for any safety integrity level. The design process needs to be restarted and executed according to either the framework standard IEC 61508 or the application standard for safety of machinery EN 62061. The according process model diagrams are presented in Figures 6.6 and 6.7 for IEC 61508 and in Figure 6.8 for EN 62061. This case project will use the process model presented by IEC 61508 for overall design process and the model from EN 62061 in designing software modules in this chapter to describe the required approaches for a safety design process.

The latter part of this chapter will handle the different parts of the previously mentioned lifecycle models considering the oil exploration machinery. The analysis is carried out with the information that has been gathered without any safety analyses and knowledge of parts of the system mechanics that can inflict a risk of a hazardous situation. The following 16 steps are related to IEC 61508 lifecycle model depicted in Figure 6.6:

Step 1 – Concept

The first step of the process model presented by IEC 61508 is described in the standard as: "*To develop a level of understanding of the EUC and its environment (physical, legislative etc.) sufficient to enable the other safety lifecycle activities to be satisfactorily carried out*" [34]. EUC stands for "equipment under control". In this phase it is required from the designer(s) to find out in what environments the device is to be used and what legislation it is required to follow. In our case the environments are various rough outdoor areas and weather conditions vary from rainy weather to dusty winds. Many people work around the machinery while it is operated and it is required to follow explosive material and oil drilling laws and IEC 61508 and/or EN 62061 standards on safety issues.

Step 2 – Overall scope definition

The second phase defines the scope of safety issues handles, or as it is defined in IEC 61508: "*To determine the boundary of the EUC and the EUC control system; to specify the scope of the hazard and risk analysis*" [34]. In our project the scope will be ensuring that the actuators will not pose a threat to the people working near the drilling equipment even in a case of component failure. This includes wrong system state recognition through sensor failure and unexpected system start-ups in case of for example system service.

Step 3 – Hazard and risk analysis

This phase will produce as its output at least one risk analysis document which determines as many possible event sequences or component failures that can lead to a hazardous event as possible. Risks need to be specified associated to these hazardous events. IEC 61508 describes this phase as follows: "*To determine the hazards, hazardous events and hazardous situations relating to the EUC and the EUC control system (in all modes of operation)…*" [34]. Analysis could consist of FMEA (Chapter 3.3.2) and FTA (Chapter 3.3.3) for component and mechanics based events and software FMEA (Chapter 3.3.2) for software based risks. This state requires failure type and failure rate information on every component which failure could lead to a hazardous event. Also information on the system mechanics considering safe states of hydraulic cylinders and the need for EX-certified components near sources of flammable gases are needed. It is known that connecting the drilling extension rods is the most dangerous procedure in the machinery. Different rod connecting failure types need to be analysed thoroughly. The results of these analyses are collected to documents according to the used analysis methods. These documents will include all possible risks and risk mitigation methods related.

Step 4 – Overall safety requirements

The information gathered from the hazard and risk analysis studies in part three and other related safety information and requirements are gathered to an overall safety requirements specification. It will determine different hazardous events, their risks and different methods for risk reduction in order to achieve a required level of functional safety.

Step 5 – Overall safety requirements allocation

IEC 61508: "*To allocate the safety functions, contained in the specification for the overall safety requirements (both the safety functions requirements and the safety integrity requirements), to the designated E/E/PE safety-related systems and other risk reduction measures; To allocate a safety integrity level to each safety function to be carried out by an E/E/PE safety-related system.*" [34]. The allocation consists of dividing the safety measures to shielding measures, safety functions and restriction of reachability to dangerous areas and allocating required SIL for each method. The safety integrity

levels required for different risk reduction methods depend on the amount of risk they pose which can be determined according to the principle depicted by Figure 3.1.

Step 6 – Overall operation and maintenance planning
Step 7 – Overall safety validation planning
Step 8 – Overall installation and commissioning planning
Steps 6 to 8 consider overall planning of operation, maintenance, installation and commissioning. These steps produce documents to enable safe and controlled procedures of the mentioned processes phases. These plans can be company specific procedures deriving from previous practises and other process models accustomed to. The maintenance plan is especially important considering any future changes in the software. It could possible include instructions on successfully implementing changes according to Chapter 6.5 in this thesis.

Step 9 – E/E/PE system safety requirement specification
This safety requirement specification (SRS) includes the safety requirements for the control system equipment and software when the SRS made in step 4 handled the system as a whole. Safety requirements for different functions executed by the control system and all the requirements on hardware and software related issues are listed in this specification. In this oil exploration machinery project possible requirements on the hardware come from the use of safety PLC modules whenever safety functions are to be implemented with them. Also some modifications to the drilling extension rod connection device may be appropriate. As all the control signals from joysticks and switches come from dedicated modules (one ifm CR0232 and two CrossControl CrossFire CX – modules), also those modules and the communication protocol between the modules need to be safety certified. As the present system uses CANopen protocol, least changes in the hardware is required by using CANopen Safety -protocol to transmit safety-related messages on the bus without changing the hardware for communication system. For the control application code to be safe, it should be implemented with a safety certified software development environment such as 3S CoDeSys Safety, which is compatible with the used 3S CoDeSys V3. Some safety features may require redundant architecture sensors or actuators. These types of changes need to be planned and analysed for risk mitigation methods in this step.

Step 10 – E/E/PE Safety-related systems realisation
In step 10 the specified system is to be realized in both hardware (sensor and actuator architecture) and software (application programming). The software part will be considered in more detail by the EN 62061 process model after step 16 of IEC 61508 lifecycle model.

Step 11 – Other risk reduction measures
If other risk reduction methods outside the scope defined in step 2 of the IEC 61508 process model need to be specified, they can be defined in this step. This step is however optional.

Step 12 – Overall installation and commissioning
Step 12 realises the E/E/PE system installation and commissioning according to the plan specified in step 8. In this step the implemented applications are downloaded into the according safety PLC modules and the designed hardware configuration is built and connected to the system.

Step 13 – Overall safety validation
The implemented system is validated to the overall safety requirements specified in step 4 taking into account the safety requirement allocations made in step. Validation is made according to the validation plans defined in step 7. According to ISO 14121 existing residual risks can be mitigated using shielding, user restrictions or instructions [31].

The following three steps of the IEC 61508 lifecycle model are considered after the machinery has been taken into use by the end-user. The system safety integrity needs to be maintained during the machine's lifespan, which requires certain rules and user and maintenance manuals.

Step 14 – Overall operation, maintenance and repair
Step 15 – Overall modification and retrofit
During the lifespan of the machinery it is crucial that the achieved level of safety isn't affected by machine use, maintenance and modifications. Therefore the operation and maintenance plans are to be followed and provided to responsible personnel in this phase of the project lifespan. In case of a modification of the machinery is required, safety impact analyses and implementation may require returning to a previous step in the lifecycle model.

Step 16 – Decommissioning or disposal
Decommissioning or disposal may require certain procedures if the unit contains hazardous materials for the user. This oil exploration machinery contains only hydraulic oil and fuel and lubrication for its diesel engine, so no special safety procedures are required other than following environmental regulations.

The following steps can be considered as sub-steps for the IEC 61508 lifecycle model step 10. The model has its own sub-steps for software development as depicted in Figure 6.7, but this case project will use the process model described by EN 62061 for it is more detailed than the model of IEC 61508 and thus is more useful as an informative example of safety-related software development process model. The following steps are

related to EN 62061 process model depicted in Figure 6.8 and are denoted as sub-steps to step 10 of the IEC 61508 lifecycle model:

Step 10.1 – Recognize the control system part suggested for safety-related function
The first step in application design and implementation process is to select one of the safety features listed in the safety requirement specification defined in step 9 of the IEC 61508 lifecycle model. In this step one has to identify the control system part that is to implement these safety functions.

Step 10.2 – Divide the safety-related control function to function blocks
The selected safety function is divided into function blocks that are later allocated into the related control system parts. This step and step 10.3 can be used in our case project to identify which I/O- and PLC-modules should be affected by the safety function at hand.

Step 10.3 – Create structure for the safety-related control system
In this phase the initial concept of the application architecture is created considering the safety function at hand. In the case of our oil exploration machinery this step would consist of specifying the possible control trails from joysticks to joystick control modules up to output modules. If redundant controls are needed, they are to be designed in this step. Also designing a safe CAN communication scheme between the modules is done here. This communication design process requires designing safety-related message scheme for all the safety-related information to be sent on the bus.

Step 10.4 – Define the safety requirement details for each function block
Each function block is defined in more detail in this step. Their inputs and outputs are specified and documented according to the knowledge acquired in Chapter 6 of this thesis. The modules should be kept simple and generic enough to be able to re-use them later during the handling of other safety functions.

Step 10.5 – Allocate the function blocks to the safety-related subsystems
The designed function blocks are allocated to subsystems, which could be a part of a larger network of subsystems or a device, such as a sensor, actuator or safety PLC. The subsystem in this step is more like an abstract device. The final devices are selected in step 10.7 and 10.9. The safety requirements for the subsystem are the same as the ones set on the function block(s) allocated to it. In this phase it is possible to return to a previous step in the process model to handle all the unhandled safety functions.

Step 10.6 – Verification
In step 10.6 the designed safety functions are verified to SRS.

Step 10.7 – Select subsystem device
In step 10.7 a proper device for the subsystem formed in step 10.5 is selected. The selected device must meet the requirements set to it in step 10.5. In practise this means that components of proper SILCL are selected for implementing the safety function.

Step 10.8 – Design and develop a subsystem
In step 10.8 a subsystem is developed to meet the requirement set to it by the subsystems allocated to it in step 10.5. This subsystem is one step lower than the subsystems handled in step 10.5 and is software-based. This structure can however be allocated to a device afterwards. Steps 10.7 and 10.8 are optional.

Step 10.9 – Design diagnostic functions
If the system requires diagnostic functions they can be designed in step 10.9. In our case project, diagnostic functions would consist of monitoring the CAN traffic for lost modules or transmission errors. These errors may lead to disabling certain functions and such behaviour can be designed in either this step of during the safety function phase in step 10.3. Wiring problems such as shortcuts and wire breaks need also be detected. Other safety function specific monitoring may also be required depending on the implemented safety functions.

Step 10.10 – Define the achieved SIL of the assumed structure for each safety-related control function
The designed structure is analysed and validated to SRS. If the requirements aren't met, returning to relevant step is required.

Step 10.11 – Document the designed system structure
After the system has been successfully validated and all the safety functions have been determined a SIL the system structure will be documented.

Step 10.12 – Implementation of the designed safety-related control system
Finally the designed control system can be implemented. This last step of EN 62061 process model can be considered as the step 12 of IEC 61508 lifecycle model and the last part in the design process of safety-related machinery.

9 CONCLUSION

Compared to “traditional” non-safe machine automation projects, safety-related projects require large amounts of documentation and supervision. If the project aims for a one of the SIL or PL safety certifications, all the required characteristics need be fulfilled with proper evidence of integrity. This might become a problem with projects requiring agile methods of project management. The safety standards state a precise process for change management which includes re-analysing all the affected parts of the system, which may multiply the amount of work needed compared to a project which specification remains unchanged during its whole design phase. The standards evolve and follow the trends in the industry and its system development methods, so a guide for safety-related system development with agile methods can be anticipated in the future.

An important and time consuming part of the development process is the risk analysis phase. It requires a lot of experience on system safety and knowledge on the mechanics of the machinery to be developed. It also requires preferably various analysers and at least two different approaches on the system to become comprehensive and reliable. Common cause failures should be appointed special attention for they may not be as easily detected as single source failures but may cause major sources of system failures. Therefore redundancy should be favoured in the most critical system parts and all the relevant parts connected to them, such as power sources or processors.

The amount of information to be adopted to be able to develop safety-related machinery and software is great. Therefore it is worth generating a common information source within the company from where all relevant information can easily be found. This source should include all the relevant safety standards for a more detailed information source and also simplified versions of the most important topics for the company for a lighter and more readable source of information. Many companies have their own coding rules for non-safe programming, but these rules enhanced with the safety-related coding rules should be made available perhaps with some agreed companywide coding conventions.

As the amount of required documents in different phases of the project is large, document templates of all the required document types should be made available with proper use manuals to ensure all required information is filled. All project material including document and meeting minutes should be made available to all personnel involved in the development process to ensure that all relevant information can be taken into account in the development process. It is really important to keep the generated documentation up to date to enable traceability, which is one of the most important issues in the certification process.

As the amount of different requirements on all the separate safety integrity level differs greatly, should all the requirements be made available according to the integrity level aimed for. The standards present the data based on its topic, but browsing through all of these standards looking for the requirements only for a single integrity level may become quite laborious. Therefore the company-based information centre should also include sections for each of the safety integrity levels (or performance levels is following ISO 13849), which would include information concerning only the relevant integrity level.

The amount of work required to produce safety-related project may seem really laborious compared to “traditional” non-safe projects, but if the requirements and information concerning the process is made easily available and the documents easy to produce, will the process become easier to implement. Also working alongside a certification body and/or any other safety specialist body during the whole project design process makes it faster and easier to go through successfully.

10 FUTURE

The oil exploration machinery used as the case chapter may not be implemented as a safe project in the near future as the markets don't require safety certification from the equipment. Also the economic situation around the globe is developing in a direction that may not favour these more expensive products over cheaper and non-certified products. However, as the amount of automation in mobile and process machinery and the use of autonomous vehicles in for example harbours and warehouses increases, so will the demand for safety certified hardware.

At the moment the development processes presented by safety standards may be quite unfamiliar to developers and the amount of different standards is quite vast and colourful. The requirements presented by law on safety of machinery may differ from country to country. European Union is unifying these requirements within its member countries and parallel standards are being unified for example ISO 13849 and IEC 61508. Unified practises and requirements will simplify and reduce the amount of work required to follow all the different sources of requirements and guides. The standards are going to develop as their use becomes more common and more guidance is required from them. For example at the moment they still lack precise rules for agile process methods and use of object oriented coding languages.

The increased call for safety certified equipment boosts the market for different types of certified hardware, increasing the competition and decreases the cost of such equipment. Perhaps at some point more than half of the equipment on the market has at least some safety capabilities. At the moment CrossControl Oy has a SIL2-capable display computer available and in use in railway applications and a compatible I/O-module is coming.

The final report of VTT research project *Ohjelmaturva* made an assumption that multicore processors may overcome the need to use multiple separate processors in safety-certified PLCs. It also predicted that automatic safety code generation from design models may become available in the future. New safety functions for autonomous vehicles in the areas of situation and environment awareness, collision prediction and fleet management may become available as the usage experience on these areas increase. There are several projects running at the moment that research these topics, one of which was FIMA FAMOUS for which this thesis was made for.

REFERENCES

[1] 3S-Smart Software Solutions GmbH. CoDeSys Safety - Integrated Safety solutions for all areas of application. [WWW]. Available: http://www.3s-software.com/index.shtml?en_CoDeSysSafety_e

[2] Anttila, R. 2006. Master of Science Thesis. The Application of a PLC-based Safety-related System. Tampere. 88 p.

[3] CrossControl Oy. CCpilot XM. [WWW]. Referenced: 5.2.2012. Available: http://www.crosscontrol.com/en-US/Products/Display-computers/CCpilot-XM.aspx

[4] CrossControl Oy. CrossFire CX. [WWW]. Referenced: 5.2.2012. Available: http://www.crosscontrol.com/en-US/Products/I/O-controllers-and-devices/CrossFire-CX.aspx

[5] CrossControl Oy. Internal educational material: SP 61508 kurs 2.

[6] Goddard, P.L., Software FMEA techniques, Proceedings Annual Reliability and Maintainability Symposium, 2000, 118–123 p.

[7] GreenHills Software. INTEGRITY Multivisor. [WWW]. Available: http://www.ghs.com/products/rtos/integrity_virtualization.html

[8] Gruhn, P., Cheddie, H., Safety Instrumented Systems: Design, Analysis, and Justification, 2nd Edition. ISA. 2006.

[9] Haapanen, P., Helminen A., Failure mode and effects analysis of software-based automation systems, STUK, Dark Oy, Vantaa, 2002, 37 p.

[10] Hietikko, M. 2003. Turvallisen tekniikan seminaari: Langattoman ohjauksen ja etäohjauksen turvallisuus. [WWW]. Available: http://koti.mbnet.fi/asaf/05_hietikko.pdf (in finnish)

[11] Hietikko, M., Malm T., Alanen J. 2009. KOTOTU. Koneiden ohjausjärjestelmien toiminnallinen turvallisuus – Ohjeita ja työkaluja standardien mukaisen turvallisuusprosessin luomiseen. VVT Tiedotteita 2485. Espoo. 99p. (in finnish)

[12] Huelke, M., BGIA. Lubineau, P., Renault D., CETIM. Lyhyt yhteenveto ohjelmistovaatimuksissa standardissa ISO 13849-1. 2005. Translation: Sundquist, M. [WWW]. Available: http://koti.mbnet.fi/asaf/OhjelmistotMS.pdf. (in finnish)

[13] IEC 61131-6. 2010. Programmable Controllers – Part 6: Functional Safety. 91p.

[14] Ifm electronic. CR0232 Extended Controller. [WWW]. Referenced: 5.2.2012. Available: http://www.ifm.com/ifmus/web/dsfs!CR0232.html

[15] Irwin, J.D. 2011. The Industrial Electronics Handbook: Industrial Communication Systems. Taylor and Francis Group. Available: http://www.crcnetbase.com/doi/abs/10.1201/b10603-23?prevSearch=%255BFulltext%253A%2Bbus%2Bsafety%255D&searchHistoryKey=

[16] ISO - International Organization for Standardization. [WWW]. Available: http://www.iso.org/iso/home.htm

[17] Koneasetus. Valtioneuvoston asetus koneiden turvallisuudesta 12.6.2008/400. [WWW]. Available: http://www.finlex.fi/fi/laki/ajantasa/2008/20080400. (in finnish)

[18] Konedirektiivi. Euroopan parlamentin ja neuvoston direktiivi 2006/42/EY. Euroopan unionin virallinen lehti. 2006. 63p. (in finnish)

[19] KW-Software. Safe software solutions according to IEC 61508 (SIL2 up to SIL3). [WWW]. Available: http://www.kw-software.com/com/iec-61508-safety/2990.jsp

[20] Malm, T., Kivipuro, M. 2000. Safety validation of complex components – Validation by analysis. VTT Research notes 2022. Espoo. 46 p. Available: http://www.vtt.fi/inf/pdf/tiedotteet/2000/T2022.pdf. (in finnish)

[21] Malm, T., Kivipuro, M. 2004. Turvallisuuteen liittyvät ohjausjärjestelmät konesovelluksissa - Esimerkkejä. VTT Research notes 2264. Espoo. 99 p. Available: http://www.vtt.fi/inf/pdf/tiedotteet/2004/T2264.pdf. (in finnish)

[22] METSTA - Metalliteollisuuden Standardisointiyhdistys ry. Standardien EN 954-1 ja ISO 13849-1 vertailu. [WWW]. Available: http://www.metsta.fi/ipubs/docs/machinery/news/2009/Vertailu_954_vs_13849.pdf (in finnish)

[23] Ohjelmaturva. 2011. Ohjelmaturva: final report. 109p. (in finnish)

[24] Paalijärvi, Jani. 2010. Master of Science Thesis. Development of Safety-Critical Software using Agile Methods. Tampere.

[25] Pilz GmbH & Co. KG. 2011. Toiminnallista turvallisuutta käsittelevät standardit. [WWW]. Referenced 30.5.2011. Available: http://www.pilz.com/knowhow/standards/standards/functional_safety/articles/00242/index.fi.jsp (in finnish)

[26] Reinert, D., Schaefer, M., Sichere Bussysteme in der Automation, Hüthig Verlag, Heidelberg, Germany. 2001, 32 p.

[27] Ristord, L., Esmenjaud, C., FMEA Per-oredon the SPINLINE3 Operational System Software as part of the TIHANGE 1 NIS Refurbishment Safety Case. CNRA/CNSI Workshop 2001–Licensing and Operating Experience of Computer Based I&C Systems. Ceské Budejovice. 2001.

[28] SFS-EN 954-1. 1997. Koneturvallisuus. Turvallisuuteen liittyvät ohjausjärjestelmien osat. Osa 1: Yleiset suunnitteluperiaatteet. 1st Edition. Helsinki. Suomen standardisoimisliitto SFS. 68 p. (in finnish)

[29] SFS-EN ISO 12100. 2003. Koneturvallisuus, perusteet ja yleiset suunnitteluperiaatteet. 1st edition. Helsinki. Suomen standardisoimisliitto SFS. (in finnish)

[30] SFS-EN ISO 13849. 2008. Koneturvallisuus. Turvallisuuteen liittyvät ohjausjärjestelmien osat. 2nd edition. Helsinki. Suomen standardisoimisliitto SFS. (in finnish)

[31] SFS-EN ISO 14121. 2007. Koneturvallisuus. Riskin arviointi. Osa 1: Periaatteet. 1st edition. Helsinki. Suomen standardisoimisliitto SFS. 67 p. (in finnish)

[32] SFS-EN EN 62061. 2009. Koneturvallisuus. Turvallisuuteen liittyvien sähköisten, elektronisten ja ohjelmoitavien elektronisten ohjausjärjestelmien toiminnallinen turvallisuus. Helsinki. Suomen standardisoimisliitto SFS. 201 p. (in finnish)

[33] SS-EN 50128. 2009. Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems. Stockholm. SIS Förlag AB. 106 p.

[34] SS-EN IEC 61508. 2010. Functional safety of electrical/electronic/programmable electronic safety-related systems. 2nd edition. Stockholm. SIS Förlag AB.

[35] Siirilä, T. 2008. Koneturvallisuus, EU:n direktiivien ja standardien soveltaminen käytännössä. 2nd revised edition. Keuruu. Otavan Kirjapaino Oy. 462 p. (in finnish)

[36] Siirilä, T. 2008. Koneturvallisuus, Ohjausjärjestelmät ja turvalaitteet. 2nd revised edition. Keuruu. Otavan Kirjapaino Oy. 472 p. (in finnish)

[37] Siirilä, T., Kerttula T. 2009. Koneturvallisuuden perusteet. 2nd revised edition. Keuruu. Otavan Kirjapaino Oy. 206 p. (in finnish)

[38] SKS Automaatio Oy. SIL-turvallisuustasot. [WWW]. Available: http://www.sks.fi/inet/sks/flow.nsf/docs/sil_turvallisuustasot?OpenDocument&ExpandSection=1%2C3%2C2#_Section1 (in finnish)

[39] Storey, N. Safety-Critical Computer Systems. Addison Wesley, 1996.

[40] Sundqvist, M (toim.); Hietikko, M; Seppälä, J; Nieminen, J. 2008. Teollisuusautomaation tiedonsiirtoliikenne. Turvaväylät. Espoo. Inspecta Koulutus Oy. 159 p. (in finnish)

[41] Sundqvist, M. 2008. Sundcon-training slides. 05_verkottuminen.ppt. (in finnish)

[42] SYSGO AG. PikeOS ROS and virtualization concept. [WWW]. Referenced: 5.2.2012. Available: http://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/

[43] The SafeCer Consortium. 2011. State-of-practice and state-of-the-art agreed over workgroup. Version 1.1. [WWW]. Available: http://www.safecer.eu/documents/pSafeCer_Deliverable_D1_0_1_StateOfThePracticeAndTheArt_v1.1.pdf

[44] United Kingdom Ministry of Defence. Defence Standard 00-58 - HAZOP Studies on Systems Containing Programmable Electronics - Part 1 Requirements. Issue 2. 2000. 24 p.

APPENDIX A: SOFTWARE ARCHITECTURE RECOMMENDATIONS ACCORDING TO IEC 61508

Technique/measure	SIL1	SIL2	SIL3	SIL4
Fault Detection	-	R	HR	HR
Error detecting codes	R	R	R	HR
Failure assertion programming	R	R	R	HR
Diverse monitor techniques (with independence between the monitor and the monitored function in the same computer)	-	R	R	-
Diverse monitor techniques (with separation between the monitor computer and the monitored computer)	-	R	R	HR
Diverse redundancy, implementing the same software safety requirements specification	-	-	-	R
Functionally diverse redundancy, implementing different software safety requirements specification	-	-	R	HR
Backward recovery	R	R	-	NR
Stateless software design (or limited state design)	-	-	R	NR
Re-try fault recovery mechanisms	R	R	-	-
Graceful degradation	R	R	HR	HR
Artificial intelligence - fault correction	-	NR	NR	NR
Dynamic reconfiguration	-	NR	NR	NR
Modular approach	HR	HR	HR	HR
Use of trusted/verified software elements (if available)	R	HR	HR	HR
Forward traceability between the software safety require-ments specification and software architecture	R	R	HR	HR
Backward traceability between the software safety require-ments specification and software architecture	R	R	HR	HR
Structured diagrammatic methods	HR	HR	HR	HR
Semi-formal methods	R	R	HR	HR
Formal design and refinement methods	-	R	R	HR
Automatic software generation	R	R	R	R
Computer-aided specification and design tools	R	R	HR	HR
Cyclic behaviour, with guaranteed maximum cycle time	R	HR	HR	HR
Time-triggered architecture	R	HR	HR	HR
Event-driven, with guaranteed maximum response time	R	HR	HR	-
Static resource allocation	-	R	HR	HR
Static synchronisation of access to shared resources	-	-	R	HR

R = recommended, NR = not recommended, HR = highly recommended

APPENDIX B: SOFTWARE ARCHITECTURE RECOMMENDATIONS ACCORDING TO EN 50128

Technique/measure	SWSIL0	SWSIL1	SWSIL2	SWSIL3	SWSIL4
Defensive programming	-	R	R	HR	HR
Fault detection & diagnosis	-	R	R	HR	HR
Error correcting codes	-	-	-	-	-
Error detecting codes	-	R	R	HR	HR
Failure assertion programming	-	R	R	HR	HR
Safety bag techniques	-	R	R	R	R
Diverse programming	-	R	R	HR	HR
Recovery block	-	R	R	R	R
Backward recovery	-	NR	NR	NR	NR
Forward recovery	-	NR	NR	NR	NR
Re-try fault recovery mechanisms	-	R	R	R	R
Memorising executed cases	-	R	R	HR	HR
Artificial intelligence - fault correction	-	NR	NR	NR	NR
Dynamic reconfiguration of software	-	NR	NR	NR	NR
Software error effect analysis	-	R	R	HR	HR
Fault tree analysis	R	R	R	HR	HR

R = recommended, NR = not recommended, HR = highly recommended