Safety Argumentation for Machinery Assembly Control Software

Julieth Patricia Castellanos-Ardila(✉), Sasikumar Punnekkat, Hans Hansson, and Peter Backeman

Mälardalen University, 721 23 Västerås, Sweden
{julieth.castellanos,sasikumar.punnekkat,hans.hansson,
peter.backeman}@mdu.se

Abstract. Assemblies of machinery commonly require control systems whose functionality is based on application software. In Europe, such software requires high safety integrity levels in accordance with the Machinery Directive (MD). However, identifying the essential regulatory requirements for the safety approval is not an easy task. To facilitate this job, this paper presents a process for *Safety Argumentation for Machinery Assembly Control Software* (SAMACS). We are inspired by patterns provided in the Goal Structuring Notation (GSN) and the use of contracts in safety argumentation. SAMACS contribution is aligning those methods with the MD by adopting EN ISO 13849. In particular, we define safety goals based on expected software contribution to control system safety and the standard guidance. Software safety goals are detailed into software safety requirements and expressed further as contracts, which shall be verified with prescribed techniques. We apply SAMACS to a case study from a European mining company and discuss the findings. This work aims at helping practitioners compose the safety case argumentation necessary to support machinery integration approval in Europe.

Keywords: Software Safety Case · GSN · Control systems · EN ISO 13849

1 Introduction

Machinery in Europe has to be CE-marked to be approved for operations. The CE (*"Conformité Européenne"*) is granted to machinery assemblies (i.e., machinery integrated to function as a whole [15]) if machinery and their protective functions conform to the health and safety requirements of the Machinery Directive (MD) [26]. EN ISO 13849:2023 [16] is a newly released version of the standard for safety-related parts of control systems (SRP/CS), which provides guidance that can be used to show conformance with the MD. In particular, protective measures based on control software require high safety integrity, which can be

This Research is supported by Vinnova via the project ESCAPE-CD, Ref: 2021-03662.

A. Ceccarelli et al. (Eds.): SAFECOMP 2024, LNCS 14988, pp. 251–266, 2024.
https://doi.org/10.1007/978-3-031-68606-1_16

represented with PLs (Performance Levels). PLs limit the probability of dangerous failures of the safety functions and are defined in EN ISO 13849:2023. This standard is expected to be considered by the European Commission in the harmonization process of the Machinery Regulation [8] (to be enforced in 2027).

In general, the site integrator can demonstrate safety-related confidence levels for machinery assemblies by using safety assurance cases, i.e., an argument structure showing that the system is acceptably safe in a specific context [18]. For software, in particular, the assurance case can be constructed by arguing that software failure modes, i.e., the failures that can give rise to, or contribute to, hazards at the system level, are mitigated [20]. However, composing the software safety arguments for SRP/CS can be challenging. In particular, it is difficult to identify the mandatory aspects that must be strengthened at design time since creating convincing arguments requires skills and experience [7]. Thus, standard recommendations and mandatory requirements can be considered as a starting point to facilitate the definition of such arguments, which can be populated later with more refined arguments as the development life cycle evolves.

Following the previous reasoning, this paper presents a process for *Safety Argumentation for Machinery Assembly Control Software* (SAMACS). We took inspiration from the structural reasoning capabilities provided by GSN (Goal Structuring Notation) [25]) and the use of contract-based desing [21] in safety argumentation (e.g., [2,11]). SAMACS's particular contribution is aligning those methods with the MD requirements by adopting the guidance provided in the standard EN ISO 13849:2023. As a result, we present two levels of arguments. At the top level, we consider the definition of an argumentation structure based on expected software contribution to SRP/CS safety, i.e., providing and protecting the intended functionality. The contributions are mapped to the standard requirements to provide safety goals. At the lower level, we present supporting arguments. In particular, safety goals are detailed into software safety requirements and expressed as contracts. For safety, contracts provide a description made up of assumptions (pre-conditions) and guarantees (post-conditions) that must be ensured by executing the safety function [24]. Safety contracts facilitate verification techniques as prescribed by the standard's performance levels (PL). We also present a case study from a European mining company and discuss our findings. This work aims at helping practitioners compose the safety case argumentation necessary to support machinery integration approval in Europe.

This paper is structured as follows. Section 2 presents the essential background information required in this paper. Section 3 presents a detailed description of SAMACS. Section 4 presents an application of SAMACS to a case study. Section 5 presents a discussion of the findings. Section 6 presents related work. Finally, Sect. 7 presents conclusions and future work.

2 Background

2.1 EN ISO 13849:2023

EN ISO 13849:2023 [16] provides requirements for designing and integrating safety-related parts of control systems (SRP/CS) for machinery. It uses performance levels (PL), i.e., a level between *a* to *e*, with *e* being the most stringent, to specify the ability of a system to perform a safety function. In clause 7, the standard includes requirements for embedded and application software, allocated to the activities included in a V-like lifecycle model. All relevant activities, i.e., those defined as essential during the specific development, must be documented, including traceability links between those activities.

From the risk assessment at the system level, the designer decides the contribution of the SRP/CS to the risk reduction, i.e., the safety function specification. Based on this input, the software specification, which shall be expressed using the criteria in Table 1, is created and used during the software development. Verification activities are chosen according to the assigned PL. In particular, functional testing and reviews are required regardless of the PL, while extended functional testing is prescribed for PL *c* to *d*. Semi-formal methods must be used to describe data and control flow in software with PL *c* to *e*.

Table 1. Safety-related Software Specification Criteria

No.	Criteria
1	Safety Functions with required PL and associated operating modes
2	Performance criteria, e.g., reaction times.
3	Communication interfaces
4	Detection and control of hardware failure to achieve the required DC and fault reaction

2.2 Assurance Cases

Assurance is the ground for justified validity of a claim [14] (i.e., a true-false statement about the limitations on the values of a property). Assurance information is commonly collected in an assurance case, a document that presents arguments with specified confidence levels supporting the claims. Assurance cases are used in safety, where the argumentation structure is done to demonstrate that the system under consideration is acceptably safe [18]. The validity of the arguments is bound to the context in which the system will be operating, as well as specified assumptions and justifications regarding such operations. In particular, to demonstrate safety, the arguments have to be in accordance to the risk reduction expected by the system. For software, the arguments are commonly oriented to justify that its functionality does not contribute to system-level hazards [28].

There are different notations to document assurance cases. We focus on the Goal structuring Notation (GSN), which uses graphical elements (see Fig. 1) [25]. GSN argumentation starts with a top-level goal supported by a strategy that connects the goal with subgoals and solutions. Goals and strategies require assumptions and justifications in a particular context to explain why the claim in a goal is acceptable. Those elements (see Fig. 1a) are connected with two types of relationships: SupportedBy (link claims with strategies/solutions) and InContextOf (link claims/strategies with contextual information). GSN provides decorators (see Fig. 1b). For example, the hollow diamond, added to a goal, represents an undeveloped goal, i.e., a goal to which the line of argument still needs to be developed. GSN structures can also be partitioned into separate packages (see Fig. 1c), e.g., an away goal represents a claim presented in another module.

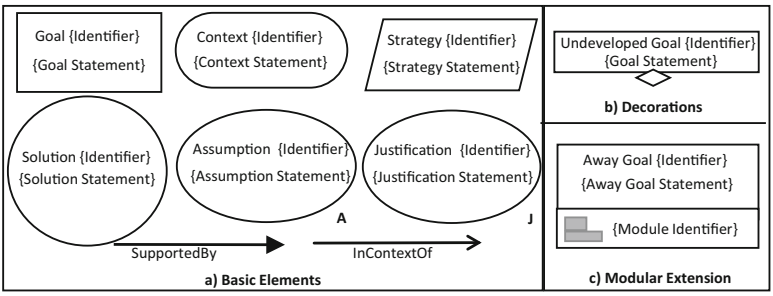

Fig. 1. GSN Elements.

2.3 Contract-Based Design

Contract-based design is an approach where correctness requirements are expressed as a contract between a method and its callers [21]. Contracts are made up of pre-conditions (that must be true before the operation call) and post-conditions (that must be ensured by the execution of the call given the pre-conditions satisfaction). In this way, contracts can ensure that the behavior of such interactions occurs as expected. Contracts have been used in safety assurance [2,11,24]. In such a context, they are called safety contracts (C). They explicitly handle a pair of properties representing the assumptions (A) on the environment (pre-conditions) and the guarantees (G) of the system under these assumptions (post-conditions). Assumptions and guarantees in a contract, which are represented as the pair C = (A;G), can be used in the safety case to illustrate the agreed relationships in an argumentative way. Safety contracts support the generation of different types of evidence required to support confidence levels. In particular, each assumed safety requirement is satisfied by at least one safety contract, and each safety contract can have supporting evidence regarding consistency, completeness, and correctness regarding the represented requirements.

For example, evidence that supports contract correctness can be a report with analysis results used to derive the contract.

3 SAMACS: Safety Argumentation for Machinery Assembly Control Software

In this section, we present SAMACS, a practitioner-centric context-specific safety argumentation process targeting software in SRP/CS, which is used in the control system of machinery assemblies. In particular, software in SRP/CS needs to comply with the requirements included in the standard EN ISO 13849:2023 (see Sect. 2.1) to be in line with European regulatory frameworks, i.e., the Machinery Directive (MD). SAMACS is aimed at helping practitioners build safety assurance cases in GSN (see Sect. 2.2) supported with the definition of safety contracts (see Sect. 2.3), which facilitate the argumentation structure required for safety conformance. Figure 2 depicts an overview and the methodological steps of SAMACS, which include the elements previously mentioned to identify the information required to build a top-level and supporting arguments to demonstrate that the software in an SRP/CS is sufficiently safe.

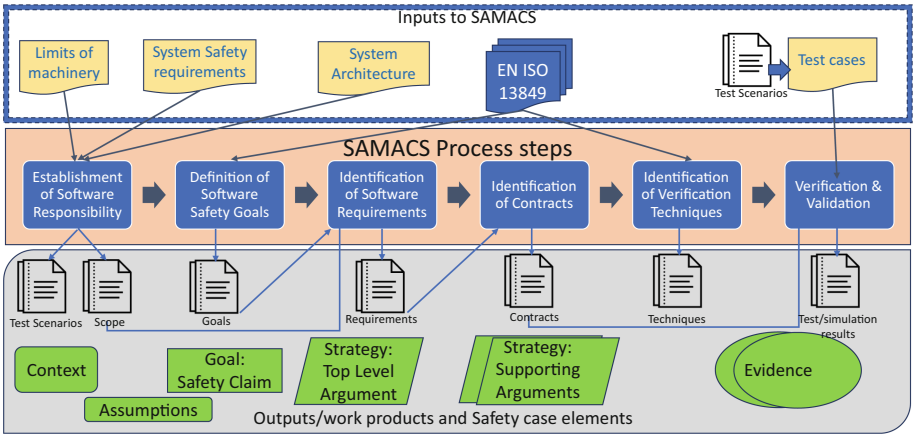

Fig. 2. SAMACS process overview

Task-1: Establishment of software responsibility
Protective measures identified at the system level are allocated to the control software via a safety specification (see Sect. 2.1). This specification, which contains the system safety requirements, the system architecture, and the limits of the machinery, is used in this task to frame the context of the top-level argument by providing the *scope* and the *test scenarios*.

Task-2: Definition of software safety goals
Software safety goals must ensure that the software does not contribute to system-level hazards. For this, the software in the SRP/CS needs to provide the intended (protective) safety functionality corresponding to the protective measurement established in the system-level risk assessment. As such functionality is essential (i.e., it shall not fail), it has to be protected from malfunctions and malpractices. Such contributions are mapped to the information requirements of EN ISO 13849:2023 (see Table 1). This information is used as the first argumentation strategy in the top-level structure.

Task-3: Identification of software requirements
Every software safety goal from Task-2 is then developed in supporting arguments. For this, each goal is detailed in terms of software safety requirements by considering the software scope resulting from Task 1.

Task-4: Definition of contracts
The software safety requirements resulting from Task-3 are expressed as contracts and are used to provide arguments regarding fulfilling such requirements by considering assumptions on the system (or the environment) and guarantees (the expected properties/functionality). As safety contracts are based on assumptions and guarantees (see Sect. 2.3), they facilitate input/output validation and the creation of error-handling specifications.

Task-5: Identification of verification techniques
Before proceeding with verification and validation, it is important to identify the right verification techniques as expected/prescribed by the standard (e.g., according to the specified PL). This is to ensure that the verification results/reports produced are in compliance with the applicable standards

Task-6:Verification and validation
Scenarios obtained in Task-1 are used to create the test cases, which are the basis for the actual validation and verification activities performed in this step. The resulting reports of this activity form the evidence and confidence levels required to support the safety claims.

4 Case Study

In this case study, we provide safety case arguments for the software of a Safety Control System (SCS) for traffic operations in an underground mine. The operations are mixed, i.e., autonomous haulers are used to transport the extracted ore, while manned-driven vehicles are used to transport personnel and materials.

4.1 Establishment of Software Responsibility

First, we collect system-level information. As depicted in Fig. 3a), the tunnel has an Autonomous Operating Zone (AOZ) (area in blue color) with entrance/exit areas where autonomous machines (shown in yellow) and manned vehicles (shown in orange) operate. Both types of vehicles have buffer areas (in gray) for waiting their turn to enter the AOZ. Manned vehicles have specific in/out

areas (in orange). Meeting areas (red rectangles) and prospected drilling areas (a side tunnel ending in a dead-end room) are alongside the tunnel. We assume that the AOZ does not have human operators on foot.

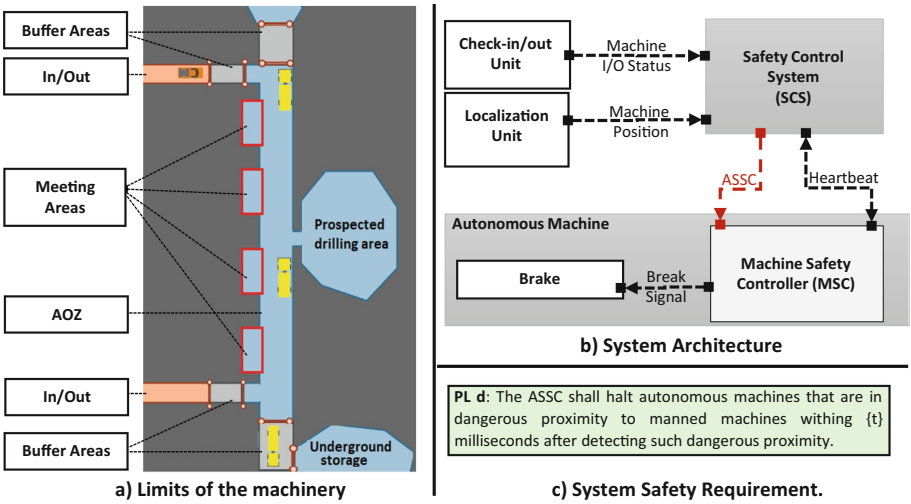

Fig. 3. Scope of the Control System.

The architecture (see Fig. 3b)) contains the system under consideration, i.e., a **Safety Control System (SCS)**, which is in charge of providing an automated safety stop command (ASSC) to meet the system safety requirement (see Fig. 3c)). The SCS receives inputs from the **localization unit**, i.e., machines position, and the **check-in/out unit**, i.e., a value indicating whether the machines enter and leave the AOZ. The SCS communicates the ASSC to the **machine safety controller (MSC)**, which converts it to a brake signal that is further sent to the **machine brakes**. The SCS and the MSC maintain a bidirectional heartbeat signal for communicating their operational availability.

Second, we determine the scope of the software function and the test scenarios. The scope is given by the system safety requirement (with PL d), which shall be allocated to the software. The limits of the machinery are used to select the test scenarios for verifying the software functionality. In our case, the control system is designed to halt autonomous machines that operate in an underground mine. So, the test scenarios can be derived systematically using the taxonomy describing the Operational Design Domain for Underground Mines (ODD-UM) provided in [5]. An excerpt of the ODD-UM related to the scenery with the case study specifications (colored in green) is depicted in Fig. 4. It says that the functionality of the SCS has to be tested in specific areas, i.e., moving and meeting, with one lane operating in both directions, i.e., up to down/down to up

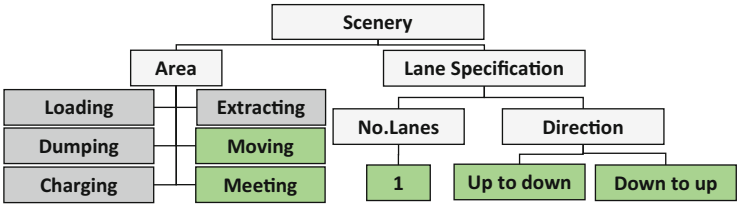

Fig. 4. ODD-UM-Scenery [5].

4.2 Definition of Software Safety Goals

Safety-related software goals have to be aligned with the mitigation strategies expected from the SCS. In general, the software is required to provide a control function (i.e., the ASSC). In that sense, the first expected contribution is that the software satisfies the intended functionality, i.e., the software provides the ASSC in the specified conditions (SC1). In addition, the software has to protect such functionality to avoid a hazardous action in mixed traffic, e.g., an autonomous machine does not stop because it is not detected or the detection parameters are out of range. Thus, the second contribution is that the software in the ASSC is protected from component malfunctions and malpractices (SC2). SC1 and SC2 can be seen in the rows colored with dark gray in Table 2.

Table 2. Software Safety Goals

ID	Goal
SC1: The software provides the ASSC in the specified conditions.	
SS1	Definition of the intended functionality (i.e., provision of the ASSC).
SG1	The intended functionality (i.e., the provision of the ASSC) is properly designed.
SG2	The intended functionality satisfies defined performance criteria.
SC2: The functionality is protected from components malfunctions and malpractices.	
SS2	Mitigation of system component's failure that contributes to software failure.
SG3	The SCS's communication interface with external components must ensure safe operation.
SG4	The software system provides detection and control of components failure.
SS3	Mitigation of systematic failure.
SG5	Relevant process requirements in compliance with EN ISO 13849:2023 have been followed.

Software contributions SC1 and SC2 are matched with strategies. In particular, the definition of the intended functionality, i.e., the provision of the ASSC (SS1), is the suited strategy for reaching SC1 since the safety control system does not have more responsibilities. Two different strategies were considered appropriate for SC2, i.e., the mitigation of system component failure that contributes

to software failure (SS2) and the mitigation of systematic software failure (SS3). SS1, SS2, and SS3 are shown in the rows colored with light gray in Table 2.

Finally, each strategy is mapped to the criteria in Table 1. In particular, criteria 1 and 2 are related to the provision of the expected functionality (SS1), while criteria 3 and 4 refer to mitigation of the system's component failure (SS2). Systematic failure (SS3) can be reached by providing evidence regarding applicable process-related requirements proposed by the standard. SG1 to SG5 are shown in the white rows in Table 2.

4.3 Identification of Software Safety Requirements

Initial brainstorming is done to identify the software safety requirements in alignment with the goals defined in Table 3. In particular, for SG1, which is related to the intended functionality, two requirements have been considered, i.e., SSR1.1 (i.e., monitoring the distance between the two types of vehicles) and SSR1.2 (i.e., provision of the ASSC in case of minimum safety distance violation). SG2 is related to performance criteria, which in this case concerns the response time (i.e., SSR2.1 and SSR2.2.) and availability (SSR2.3) of the safety function. In the case of SG3, two requirements are initially defined, i.e., monitoring and diagnostics of communication failures (SSR3.1), as well as validation of integrity data before processing (SSR3.2). For SG4, it is determined that self-tests shall be performed at startup and during operations to ensure component functionality (SSR4.1). In addition, the ASSC shall be issued in case of controller failure (SSR4.2) or input devices failure (SSR4.3). SG5 can be populated with process-related aspects that result from decisions made during software development. For this case study, the decisions corresponding to verification activities (see Sect. 4.5) were relevant. In particular, the functionality is developed in UPPAAL[1] for model checking and automatically translated to software (SSR5.1) that has to be properly included in a simulation tool (SSR5.2) for further verification. Both the UPPAAL and the simulation tool shall support the investigation of the system under consideration (SSR5.3). This could mean that quality control of such tools has to be provided.

Table 3. Software Safety Requirements

SG. ID	Software Requirement
SG1	**SSR1.1:** The SCS shall monitor the distance between the autonomous machines and the manned vehicles for the specified operating zones of the AOZ
	SSR1.2: The Automated Safety Stop Command (ASSC) shall be issued if an autonomous machine violates the safety distance with respect to a manned vehicle
SG2	**SSR2.1:** The ASSC shall be computed within $\{t_1\}$ milliseconds after a violation of the safety distance is detected
	SSR2.2: The ASSC shall be send within $\{t_2\}$ milliseconds after it is computed
	SSR2.3: The ASSC shall be sent if the periodic heartbeat signal from the MSC stops
SG3	**SSR3.1:** The SCS shall include real-time monitoring and diagnostics to detect communication delays, packet loss, or data corruption
	SSR3.2: The SCS shall validate the integrity of incoming data before processing it to avoid hazards caused by corrupted data
SG4	**SSR4.1:** The SCS shall perform self-tests at startup and during operation to ensure all components are functioning properly
	SSR4.1: The ASSC shall be issued in case of controller failure
	SSR4.2: The ASSC shall be issued if control inputs from the position and check-in/out units are missing or are out of range
SG5	**SSR5.1:** UPPAAL models shall be correctly modeled and translated into software code
	SSR5.2: The software code shall be properly included in the simulation tool
	SSR5.3: Tools (UPPAAL & Simulator) shall support the investigation of the system

[1] https://uppaal.org/documentation/.

4.4 Definition of Contracts

Contracts consider the expected behavior described in the software requirements, i.e., the guarantee, as well as the analysis required to identify assumptions. In Table 4, we show contracts for SSR1 (i.e., SC1.1) and SSR2 (i.e., SC2.1).

Table 4. Software Safety Contracts

ID	Contract
SC1.1	**A1.1:** (AM[i].I/O-Status = IN) AND (MM[j].I/O-Status = IN);
	G1.1: implies monitoredDistance(AM[i].position,MM[j].position);
SC1.2	**A1.2:** safetyDistance = {MinimumSafetyDistance}
	G1.2: monitoredDistance $\geq$ SafetyDistance implies (ASSC = TRUE);

Contract SC1.1 assumes that at least one autonomous machine (AM[i]) and one manned vehicle (MM[j]) are inside the AOZ (i.e., I/O-status = IN, provided by the Check-in/out unit). Such an assumption is essential since the SCS does not react to other configurations, i.e., only autonomous machines or only manned vehicles in the AOZ. This assumption establishes a guarantee regarding monitoring such vehicles to provide the current distance between them based on the vehicle's positions (i.e., AM[i].position and MM[j].position, provided by the localization unit). Contract SC1.2 assumes a previously defined minimum safety distance to be maintained between these two types of vehicles. The guarantee is that if the minimum safety distance is violated, i.e., monitoredDistance $\geq$ SafetyDistance) then the ASSC is issued (ASSC = TRUE).

4.5 Identification of Verification Techniques and Evidence Provision

In our case study, evidence regarding verification shall be aligned with PL d (see Sect. 2.1). In particular, expert reviews are suitable for work products that require manual analysis. i.e., specifications. The code also required reviews. Automated techniques are suitable for software unit testing. We decided to provide a higher level of confidence considering model-checking results for the software by using the UPPAAL model checker. Model checking also allows describing data and control flow in software, which is mandatory for PL d. Simulations are considered relevant at the component and system levels, so complete functionality is probed. A simulation is one of the extended functional techniques suitable for functionality with PL d. As we provide arguments at design time, we can also check the compliance of the process plans. This aspect is out of the scope of this paper, but examples of techniques for providing such compliance checking can be seen in our previous work (see, for example, [3]).

4.6 Composing the Safety Case Arguments

The top-level argument for the safety case is presented in Fig. 5. It starts with the main goal G1, i.e., the software is sufficiently safe in a given context. Sufficiently safe is a general assumption corresponding to the expected software contributions (SC1 and SC2) to safety presented in Table 2. The context C1 corresponds to the scope of the software function identified in Fig. 3. The arguments develop over the considerations of strategies SS1 to SS3 presented in Table 2, which result in the five away goals SG1 to SG5, also presented in Table 2.

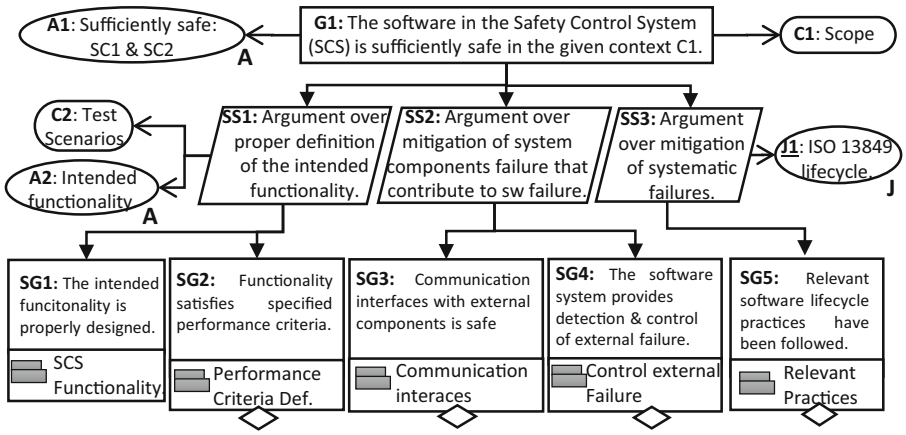

Fig. 5. Top Level Argument

Figure 6 presents a supporting argument for the away goal SG1. Three strategies are identified to support this goal. The first strategy (S1.1) is the proper definition of individual requirements for SG1, which is then developed further with two goals, i.e., SG1.1 and SG1.2, corresponding to the software requirements SSR1.1 and SSR1.2 (see Table 3). Those goals are further augmented with the contracts corresponding to each requirement i.e., SC1.1 and SC1.2 (see Table 4). The second strategy (S1.2) regards the sufficiency of requirements SSR1.1 and SSR1.2 in the implementation of the goal SG1. This argumentation branch reaches the final stages by showing two evidence elements, E1.1 (i.e., expert review) and E1.2 (i.e., simulations results). Finally, the third strategy (S1.3) is the integration of SG1 with other software goals. This branch also reaches a final stage by providing evidence E1.3 (i.e., simulation results). As E1.3 includes the whole integration, it can also be a validation test. Thus, extra evidence related to the user acceptance test results can also be added to strengthen the argument. The strategy S1.3. is the same for the away goals SG1 to SG4 developed at the top-level argument (see Fig. 5). Therefore, this strategy can instead be located in a specific branch related to integration testing at such an argumentation level.

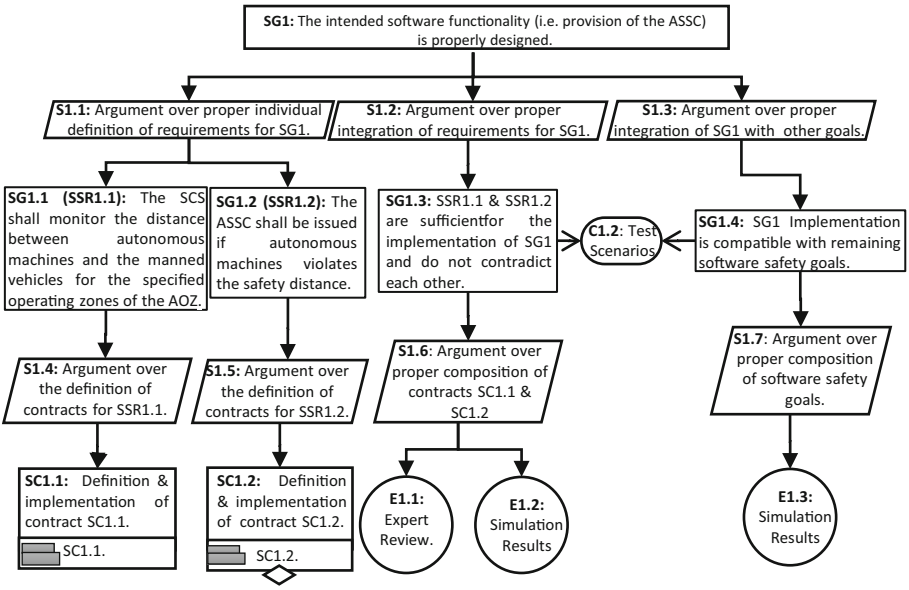

Fig. 6. Argumentation Structure of SG1.

Finally, in Fig. 7, we present the structure for the argument supporting SC1.1 (see Table 4), which corresponds to the definition and implementation of the safety contract linked to requirement SSR1.1. In this case, the argument starts by the contract assumption A1.1 (i.e., (AM[i].I/O-Status=IN) AND (MM[j].I/O-Status=IN)), which means that at least one machine of each type is inside the AOZ. Such an assumption supports the expected contract guarantee, G1.1 i.e., distance monitoring between AM and MM. This argument is extended with strategies regarding contract definition (i.e., S1.1.1), which is

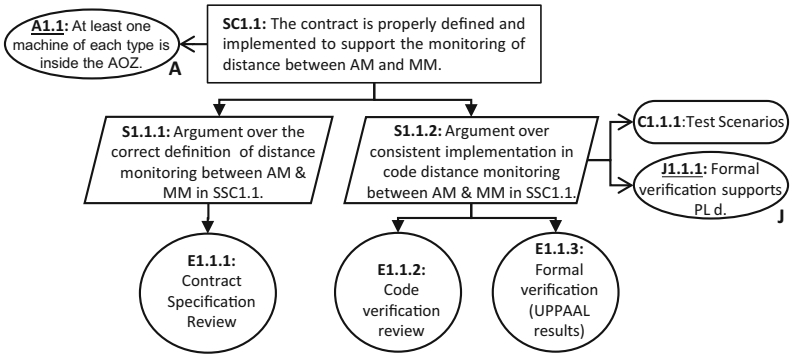

Fig. 7. Argumentation Structure of SC1.1.

manually reviewed (E1.1.1), and consistent implementation, which is also manually reviewed (E1.1.2) as well as formally verified (E1.1.3). The formal verification is done against the test scenarios (i.e., C1.1.1) and based on the justification that, according to the standard, formal verification supports PL d (i.e., J1.1.1.).

5 Discussion

It is challenging to provide a reasonable argument about the safety of software systems. The reason is that predicting quantifiable failure rates (commonly done for hardware) is difficult for software, which behaves uniquely. Thus, we need a strategy that helps us to think about what can go wrong in a specific context. This is especially important when the software function is safety-related and should not fail, as it occurs with software functions allocated to SRP/CS. The accepted practice is based on the qualified confidence given as performance levels (PLs) provided by industry standards. However, it could be challenging to select the appropriate standard, and once selected, its guidance could be difficult to interpret and use in a safety argument. In particular, the guidance for software construction in standards is often process-oriented. However, a careful view of such recommended practices can provide ideas of how to justify safety by considering the software as a product without leaving aside the required processes.

In Sect. 3, we presented a process where the guidance provided by the standard EN ISO 13849:2023 is used to reason about the software functionality. In particular, we departed from the safety-related software specification's structure as described in the standard (see Table 1) to align the safety case with the prescribed practices. From this guidance, we created the software safety goals and used them as safety case modules independent of each other. Such independence allows the creation of new arguments as the software process evolves without changing the initial safety case structure, a feature that can be also aligned with agile practices. The modules also provide readability to the software safety case since it can grow without looking excessively complex.

The structure of the resulting safety assurance case also permits practitioners to conduct brainstorming sessions regarding relevant software functionalities in the SRP/CS, as well as the implications in terms of supporting software functions to protect such functionality. For example, in our case study (see Sect. 4), we considered performance criteria related to response time and availability. However, further iterations can consider other relevant criteria, e.g., resource consumption. Cybersecurity implications also have a place in this argumentation since those practices aim at protecting software functionality. The last argumentation structure presented in Fig. 7 shows the information on the software unit that is traceable to the software safety requirements and component integration in Fig. 6 up to the claim in goal G1 in Fig. 5. In this way, we present a journey covering all the software development lifecycle steps. Thus, this process can be applied to concrete scenarios to create convincing stories that make the safety assurance argument communicable and comprehensible to all stakeholders.

6 Related Work

GSN has been used in the creation of safety argumentation patterns for software. In particular, Weaver [28] presents a framework for articulating software safety arguments based on evidence categorization. Argument patterns for COTS (Commercial-Off-The-Shelf) software are also proposed in Ye's work [29]. We have a similar view as these two approaches regarding evidence provision about the mitigation of software contribution to system-level hazards. However, these works are standards agnostic, which we consider essential in our approach. The Pegasus framework presents safety argumentation related to automated driving systems [19], which is interesting but difficult to apply in the argumentation related to SRP/CS, which is, in essence, less complicated than the required in autonomy systems (i.e., SRP/CS functionality is very reduced and punctual). Ayoub et al. [1] propose the common characteristics map with results in process-oriented evidence. It emphasizes evidence from verification activities considering the techniques, tools, human expertise, and artifacts produced. However, it does not consider the specific software description as we do in our work. General discussions regarding argument sufficiency for software safety are presented in [6,12,13] among others. However, we aim at proposing a context-specific argumentation process that can support practitioners in the machinery context.

Creating convincing arguments for demonstrating machinery safety requires experience in the field and argumentation skills. However, few published general works in this field are available for supporting practitioners. For example, in the work of Gallina et al., [9], there is a knowledge management strategy for handling process artifacts in compliance with the MD, which can be useful for creating evidence artifacts at such a level. In [10], a list of elements that can be used as evidence for creating a safety assurance case is provided. However, these two works do not present structures that support the argument development. In [27], an overall safety assurance argument in GSN is sketched. However, it lacks the level of detail required for a complete argumentation structure. In addition, it only relates to a specific operation (i.e., transportation), which is difficult to extrapolate to machinery aspects beyond such operation. The works presented in [4,17] cover more argumentation details (including the criticality levels prescribed in standards) but do not cover software at the control level, which is our main focus. Methodologies for designing SRP/CS following EN ISO 13849 are provided in [22,23]. However, none of the previous works present explicit argumentation structures covering SRP/CS software in line with the standards harmonized with the MD.

7 Conclusions and Future Work

SAMACS is a process aimed at helping practitioners compose the safety assurance case arguments required to support machinery integration approval in the European context. In particular, SAMACS supports the creation of a safety case for the software included in SRP/CS by adopting current practices in software

safety argumentation, i.e., GSN structure and contracts, and the guidance provided by the standard EN ISO 13849:2023. This standard is relevant since it provides accepted guidance for conformance with the Machinery Directive and is expected to be considered for conformance with the Machinery Regulation.

Future work includes providing more case studies to assess the effectiveness of this work's methodological steps in practice. We will also present the process and the resulting safety cases to practitioners and safety assessors to evaluate their perceptions regarding the argument's comprehensiveness and level of coverage. In addition, tools for supporting our process are planned to be investigated.

References

1. Ayoub, A., Kim, B.G., Lee, I., Sokolsky, O.: A systematic approach to justifying sufficient confidence in software safety arguments. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012. LNCS, vol. 7612, pp. 305–316. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33678-2_26
2. Bate, I., Hawkins, R., McDermid, J.: A contract-based approach to designing safe systems. In: 8th Workshop on Safety-critical Systems and Software (2003)
3. Castellanos Ardila, J.P., Gallina, B., Governatori, G.: Compliance-aware engineering process plans: the case of space software engineering processes. In: Artificial Intelligence and Law, pp. 1–41 (2021)
4. Castellanos Ardila, J.P., Punekkat, S., Hansson, H., Grante, C.: Arguing operational safety for mixed traffic in underground mining. In: 18th Annual System of Systems Engineering Conference (2023)
5. Castellanos Ardila, J.P., Punnekkat, S., Fattouh, A., Hansson, H.: A context-specific operational design domain for underground mining (ODD-UM). In: European Conference on Software Process Improvement, pp. 161–176. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-15559-8_12
6. Chechik, M., Salay, R., Viger, T., Kokaly, S., Rahimi, M.: Software assurance in an uncertain world. In: Hähnle, R., van der Aalst, W. (eds.) FASE 2019. LNCS, vol. 11424, pp. 3–21. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16722-6_1
7. Cheng, J., Goodrum, M., Metoyer, R., Cleland, J.: How do practitioners perceive assurance cases in safety-critical software systems? In: Workshop on Cooperative and Human Aspects of Software Engineering, pp. 57–60 (2018)
8. Europen Parliament and the Council: *Regulation (EU) 2023/1230* (2023)
9. Gallina, B., Olesen, T.Y., Parajdi, E., Aarup, M.: A knowledge management strategy for seamless compliance with the machinery regulation. In: European Conference on Software Process Improvement, pp. 220–234. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-42307-9_17
10. Global Mining Guidelines Group: Systems Safety for Autonomous Mining (2021)
11. Graydon, P., Bate, I.: The nature and content of safety contracts: challenges and suggestions for a way forward. In: 20th Pacific Rim International Symposium on Dependable Computing, pp. 135–144. IEEE (2014)
12. Habli, I., Hawkins, R., Kelly, T.: Software safety: relating software assurance and software integrity. Int. J. Crit. Comput.-Based Syst. **1**(4), 364–383 (2010)
13. Hawkins, R., Kelly, T.: Software safety assurance-what is sufficient? In: 4th IET International Conference on Systems Safety 2009. Incorporating the SaRS Annual Conference, pp. 1–6. IET (2009)

14. ISO/IEC JTC 1/SC 7: ISO/IEC/IEEE 15026:2019. Systems and software engineering - Systems and software assurance (2019)
15. ISO/TC 199: ISO 12100:2010. Safety of machinery - General Principles for design - Risk Assessment and Risk Reduction (2010)
16. ISO/TC 199: EN ISO 13849-1:2023. Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design (2023)
17. Javed, M.A., Muram, F.U., Hansson, H., Punnekkat, S., Thane, H.: Towards dynamic safety assurance for Industry 4.0. J. Syst. Arch. (2021)
18. Kelly, T.P.: Arguing safety: a systematic approach to managing safety cases. Ph.D. thesis, University of York (1999)
19. Maus, A.: Pegasus safety argumentation (2018). https://www.pegasusprojekt.de/files/tmpl/pdf/PEGASUS%20Safety%20Argumentation.pdf
20. McDermid, J.A.: Software safety: where's the evidence? In: 6th Australian Workshop on Safety Critical Systems and Software, pp. 1–6 (2001)
21. Meyer, B.: Applying design by contract. Computer **25**(10), 40–51 (1992)
22. Porras, A., Romero, J.A.: A new methodology for facilitating the design of safety-related parts of control systems in machines according to ISO 13849:2006 standard. Reliabil. Eng. Syst. Saf. **174**, 60–70 (2018)
23. Söderberg, A., Hedberg, J., Folkesson, P., Jacobson, J.: Safety-related Machine Control Systems using standard EN ISO 13849-1 (2018)
24. Söderberg, A., Johansson, R.: Safety contract-based design of software components. In: International Symposium on Software Reliability Engineering (2013)
25. The Assurance Case Working Group (ACWG): GSN Community Standard. Version 3 (2021)
26. The Council of the European Parliament: Machinery - Directive 2006/42/EC (2006)
27. Volvo Technology AB - Advanced Technology & Research: Automated Safe and Efficient Transport System - VINNOVA Project- Ref: 2015-00612 (2015). https://www.vinnova.se/en/p/automated-safe-and-efficient-transport-system/
28. Weaver, R.A.: The safety of software: constructing and assuring arguments. Ph.D. thesis (2003)
29. Ye, F.: Justifying the use of COTS Components within safety critical applications. Ph.D. thesis, Citeseer (2005)