

# Characterizing the Chain of Evidence for Software Safety Cases: A Conceptual Model Based on the IEC 61508 Standard

Rajwinder Kaur Panesar-Walawege,  
Mehrdad Sabetzadeh, Lionel Briand  
Simula Research Laboratory, Oslo, Norway  
{rpanesar,mehrdad,briand}@simula.no

Thierry Coq  
Det Norske Veritas  
Paris, France  
thierry.coq@dnv.com

**Abstract**— Increasingly, licensing and safety regulatory bodies require the suppliers of software-intensive, safety-critical systems to provide an explicit software safety case – a structured set of arguments based on objective evidence to demonstrate that the software elements of a system are acceptably safe. Existing research on safety cases has mainly focused on how to build the arguments in a safety case based on available evidence; but little has been done to precisely characterize what this evidence should be. As a result, system suppliers are left with practically no guidance on what evidence to collect during software development. This has led to the suppliers having to recover the relevant evidence after the fact – an extremely costly and sometimes impractical task. Although standards such as the IEC 61508 – which is widely viewed as the best available generic standard for managing functional safety in software – provide some guidance for the collection of relevant safety and certification information, this guidance is mostly textual, not expressed in a precise and structured form, and is not easy to specialize to context-specific needs. To address these issues, we present a conceptual model to characterize the evidence for arguing about software safety. Our model captures both the information requirements for demonstrating compliance with IEC 61508 and the traceability links necessary to create a seamless chain of evidence. We further describe how our generic model can be specialized according to the needs of a particular context, and discuss some important ways in which our model can facilitate software certification.

## I. INTRODUCTION

Safety-critical systems such as those found in the avionics, automotive, maritime, and energy domains are often required to undergo a safety certification process. The goal of certification is to provide an assurance recognized by society (and in some cases by law) that a system is deemed safe by the certification body.

The justification for safe operation of a system is usually presented in what is known as a safety case [1-5]. Kelly [1] describes a safety case as being composed of three principal parts: safety objectives, arguments, and evidence. Demonstrating the satisfaction of the objectives involves gathering systematic evidence during development and constructing well-reasoned arguments that relate the evidence to the objectives.

With the growing use and complexity of software in safety-critical systems, licensing and safety regulatory bodies increasingly require system suppliers to provide an explicit software safety case. A software safety case is a part of an

overall safety case, which provides assurance that the software elements of a system are sound, and that these elements are used correctly within the overall system.

While the argumentation aspects of software safety cases (and generally, safety cases) have been studied for a long time [6]; little has been done to precisely characterize the evidence that underlies software safety arguments. As a result, suppliers of safety-critical software have been left without proper guidance on what evidence to collect during development. This has led to the suppliers having to recover the relevant evidence after the fact, which can be extremely costly or even impractical. In addition, the quality of the overall safety case is bound by the quality of the weakest link. Hence, current practices for managing software safety evidence can severely limit the effectiveness of safety cases.

Although standards such as IEC 61508 [7] – which is widely viewed as the best available generic standard for management of functional safety in software – provide some guidance for collecting safety and certification information, this guidance is mostly textual, not expressed in a precise and structured form, and is not easy to specialize to context-specific needs.

The goal of this paper is to address the above issues by providing a conceptual model that characterizes the evidence necessary for arguing about software safety. Our model captures both the information requirements for demonstrating compliance with IEC 61508, and the traceability links necessary to create a seamless continuum of evidence information, called the *chain of evidence* [5].

In real-life projects, multiple rules, regulations and standards apply; therefore, our conceptual model needs to be further specialized according to the safety needs of the application domain (e.g., national and international laws, and class society regulations in the maritime domain [8]), the development process, and the technologies used to express requirements and design decisions (e.g., SysML[9]). A specialized version of the conceptual model can in turn be used for constructing an evidence repository. Such a repository can be utilized for automating various development and analysis tasks associated with safety-critical software, including safety report generation, checking of various compliance rules, and impact analysis.

The remainder of this paper is structured as follows: In Section II, we give a brief introduction to the IEC 61508 standard. We provide a detailed exposition of our conceptual model in Section III; and in Section IV, we exemplify some key aspects of the model. In Section V, we explain how the model can be specialized according to the needs of a

particular context. In Section VI, we describe some important applications of the model in software certification. Section VII provides initial validation of the usefulness of our model. Section VIII compares our work to related research; and Section IX concludes the paper with a summary and directions for future work.

## II. BACKGROUND

This section provides background information on the IEC 61508 standard (version published in 1998). The standard is concerned with improving the development of safety-related electrical/electronic/programmable electronic systems (E/E/PES) whose failure could result in harm to people, equipment, and/or the environment. IEC 61508 is a generic standard and can either be used directly or for the creation of domain-specific standards in industries that require an equivalent level of safety.

The standard applies to both low-demand and continuous mode systems. In a low-demand system, the frequency of demands for operation is low (the standard specifies a precise range). An example of a low-demand system is a fire & gas protection system, which alerts personnel if a fire or gas leakage is detected and initiates protective actions either automatically or through manual intervention. A continuous (or high-demand) mode system is one with a high frequency of demands for operation. An example would be the dynamic positioning system that continuously controls a vessel's movement when the vessel is near a port or rig.

The goal of the standard is to ensure that safety-related E/E/PES systems operate correctly in response to their inputs. This is referred to as *functional safety*. Functional safety is not all there is to safety. For example, the activation of an alarm in response to a fire breakout is a functional safety measure, whereas the use of fire resistant walls to control the spread of fire is not, although the latter measure protects against the same hazard. IEC 61508 deals only with functional safety. A function that a control system performs to ensure that the system remains in a safe state is referred to as a safety function. Each safety function specifies what safety objective is to be achieved (safety function requirement) and the level of integrity with which the safety function is implemented (safety integrity level).

To systematically deal with the activities necessary to achieve the required level of safety, the standard adopts an overall safety lifecycle. The lifecycle starts with establishing the concept and overall scope of a system, and then conducting a hazard and risk analysis to determine the hazards that can occur and the risks that they pose. Together, these activities determine what has to be done to avoid the hazardous situations (derivation of safety requirements) and the level to which safety has to be provided (derivation of safety integrity levels).

In the next step, the safety requirements are allocated to the various designated E/E/PE safety-related systems, other technology safety-related systems, and external risk reduction facilities (only the E/E/PE allocations are within the scope of the standard). Once the allocations are made, the realization phase begins for both the hardware and software aspects of the E/E/PE safety-related systems. In tandem,

planning begins for the installation and commissioning, operation and maintenance, and the final overall safety validation of the system. During the realization phases, the standard calls for a number of overarching verification, management, and assessment activities. The life cycle further takes into account the eventual, safe, decommissioning or retrofit of the system.

In this paper, we deal with the activities that take place during the realization of the software part of a programmable electronic safety-related system. The standard requires an explicit *software* safety lifecycle, shown in Figure 1, for the development of a PES.

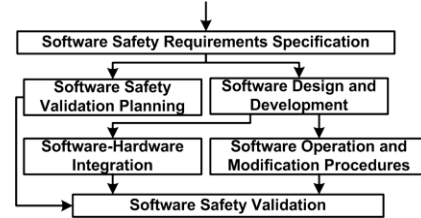


Figure 1: IEC 61508 Software Safety Lifecycle

The lifecycle for the realization of the hardware in the E/E/PES is similar except that it applies to the hardware. It is important to realize that the hardware and software development lifecycles are happening in parallel and certain hardware architectural assumptions will have to be in place before the relevant software lifecycle can be started.

The software has to be implemented such that it fulfills the safety requirements allocated to it. In order to be able to show this during software safety validation and assessment, it is crucial to maintain traceability between the software safety requirements, and the decisions taken during design, and the actual implementation in code. This is a complex task and needs to be performed whilst the system is being developed, not once the development has finished. Providing an accurate description of the safety information that needs to be preserved during software development is the main motivation behind our work in this paper.

The software safety lifecycle in Figure 1, together with the overall lifecycle activities (verification, management and assessment of safety) specialized to software, form the basis of the conceptual model in Section III.

## III. CONCEPTUAL MODEL

Figure 2 formalizes our conceptual model as a UML class diagram. The concepts in the model are only succinctly and intuitively defined here and precise definitions are provided in a technical report [10]. To manage the apparent complexity of the model, the concepts have been divided into ten packages. We describe these packages next. Note that this conceptual model is meant to define, in a precise way, information requirements to demonstrate both compliance with the standard and, perhaps more importantly, ensure the safety chain of evidence is collected.

### A. System Concepts

The System Concepts package describes the basic elements needed to conceptualize safety-related control systems that involve both hardware and software. A

Programmable Electronic System (PES) is a block made up of one or more hardware blocks and controlled by a number of software blocks. A hardware block may represent a mechanical, electrical or electronic entity, both programmable and non-programmable. Both hardware and software blocks can be hierarchically decomposed into lower-level blocks. For software, the typical decomposition levels are: module, component, subsystem, and system. The links between blocks and the corresponding development artifacts (see Section III.E) are captured through the association between the Block and Artifact concepts.

Interactions between the blocks are expressed as interfaces. Making the interfaces explicit is necessary to minimize mismatches and deadlocks during integration. For arguing software safety at the level of an individual PES, the interfaces of interest are those that have a software block at least at one end (i.e., no hardware-to-hardware interfaces). For integration of system-of-systems, interfaces between PESs are crucial as well.

Interactions between a PES and the human elements are modeled through user roles. Safety issues can arise due to misuse or unauthorized access to a system. Mitigating these issues requires an accurate description of how different groups of users can interact with the PES.

Each block is traceable to the requirements allocated to it. At the PES level, the allocations are made during the safety requirements allocation step of the IEC 61508 overall safety lifecycle. The PES-level (safety) requirements are used to derive requirements for the software and hardware blocks. We discuss requirements in Section III.C. Blocks can evolve over time and are thus versioned and placed under configuration management. Configuration management is addressed in Section III.G.

### *B. Hazard Concepts*

The Hazard Concepts package captures the hazards and the risks they pose, which then constitute grounds for safety requirements and safety integrity levels. A hazard is any real or potential condition that can cause injury, illness, or death to personnel; damage to or loss of a system, equipment or property; or damage to the environment.

The potential for a hazard to occur exists whenever the system has some hazardous element in it – this is the basic hazardous resource creating the impetus for the hazard. An example could be a hazardous energy source such as explosives. The hazardous element in itself is not sufficient to trigger a hazard. The trigger is captured using the concept of an initiating mechanism. An initiating mechanism is a sequence of events that leads to the actualization of a hazard. Hazards are the basis for deriving safety requirements.

Each hazard is analyzed to assess the risks it poses, using risk assessment techniques. In essence, a risk is the combination of the probability of occurrence of a particular harm and the severity of that harm to a person or object, usually referred to as the target.

The probability of occurrence is referred to as the likelihood and is sometimes qualitatively divided into: frequent, probable, occasional, remote, improbable and incredible. The level of harm caused is referred to as the

consequence and can be qualitatively rated as catastrophic, critical, marginal or negligible. Together, these are used to give a tolerance level to a risk. The level of tolerance of a risk is then used to derive a safety integrity level. The results of hazard and risk analysis are presented as a Description. Hazards and risks can be referred to in various other development artifacts such as requirements specifications.

### *C. Requirements Concepts*

The concepts necessary to describe the requirements for creating, operating, maintaining and decommissioning a system are included in the Requirements Concepts package. Traceability from requirements to the corresponding PES, system blocks, hazards and artifacts forms an important part of the chain of evidence.

A requirement is a statement identifying a capability, characteristic, or quality factor of a system in order for it to have value and utility to a user. Requirements are one of the central concepts of system development and are thus naturally connected to concepts in many other packages. A requirement is typically concerned with some particular aspect of the system (functionality, usability, performance etc.). This information is captured in the ‘type’ of the requirements. Each requirement is linked to the block(s) that must realize it. A rationale item might be affixed to a requirement to justify why that requirement exists. If an issue is raised about a requirement at some stage of development, the issue is recorded and linked to the requirement as well.

The source of a requirement may be a person, organization, standard or recommended practice. A requirement may apply to certain operating modes of the system such as normal operation, maintenance, shut down, and emergency. Each operating mode may have a set of designated states, which would render the system safe or unsafe. For example, it might be unsafe to run a boiler engine during maintenance.

A particular class of requirements is that which concerns safety. Safety requirements are used to ensure that the system carries out its functions in an acceptably safe manner. These requirements are derived from hazards, and are intended to mitigate the risks posed by these hazards. Each safety requirement is assigned a safety integrity level based on the likelihood and consequences of the risks it mitigates.

Safety integrity is defined as the probability of the system to successfully perform a required safety function. Usually, the dual notion of probability of failure (instead of probability of success) is used. The failure rate unit can be “failure per hour” for high demand or continuous operation and “failure on demand” for low demand operation. When a safety requirement only partially addresses a risk, the residual risk (i.e., the risk fraction remaining after the protective measures have been taken) is recorded.

A requirement may relate to other requirements in a number of ways. Example relationships include: when a lower-level requirement (e.g., module requirement) is derived from a higher-level requirements (e.g., system or component requirement), when a requirement contributes positively or negatively to the satisfaction of another

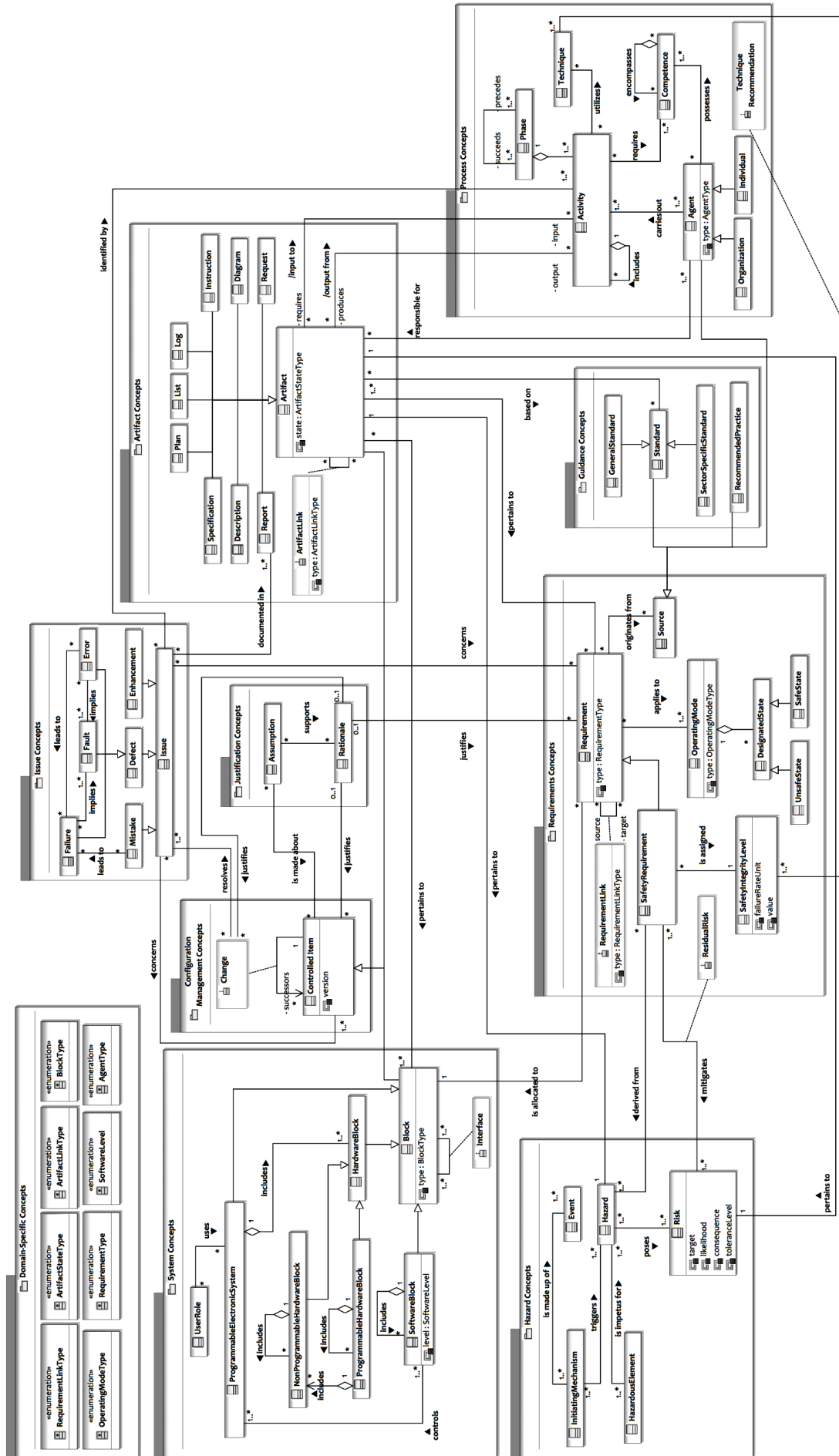


Figure 2: Core Concepts and Relationships

requirement, and when a requirement conflicts with or overrides another requirement. In these cases, we need to maintain traceability between the involved requirements. This is done using a reflexive association for the Requirement concept.

A requirement can have various development artifacts associated with it. Particularly, a requirement is specified in some requirements specification, and referenced in many other artifacts such as design and architecture specifications, test plans, source code, and also other requirements specifications where related requirements are captured.

#### D. Process Concepts

Development of software for a PES follows a certain process. This is expressed using the Process Concepts package. Further refinements of the process concepts would have to be performed in specific contexts of applications, accounting for the specifics of the process in place.

The notion of activity is the central concept in this package, representing a unit of behavior with specific input and output. An activity can be further decomposed into sub-activities. A (lifecycle) phase is made up of a set of activities that are carried out during the lifetime of a system, starting from system inception to decommissioning. To be able to accommodate iterative development processes, we do not restrict activity types to particular development phases. Restrictions will be expressed externally where necessary, for example using OCL constraints [11].

Each activity utilizes certain techniques to arrive at its desired output, given its input. The selection of techniques is intimately related to the safety integrity level that needs to be achieved. For example, if the activity in question concerns software verification, constructing formal proofs of correctness is usually unnecessary for low integrity levels, whereas, formal proofs are highly recommended (and often necessary) for the highest integrity levels. Specific technique recommendations (e.g., recommended, not recommended, highly recommended, mandatory) are made based on the overall standard guidelines, and the requirements of the certification bodies in charge of assessing functional safety.

Each activity requires certain kind of competence by the agents performing it. The agent itself can be either an individual person or an organization. In either case, the agent is identified by the type of role it plays, for example the agent may be the supplier of a system or the operator. Agents can be made responsible for certain development artifacts.

#### E. Artifact Concepts

The Artifact Concepts package characterizes the inputs and outputs of the development activities. The main concept here is Artifact, which describes the tangible by-products produced during development. IEC 61508 provides a high-level classification of the different types of development artifacts: a *specification* (e.g. requirements specification); a *description* (e.g. description of planned activities); a *diagram* (e.g. architecture diagram); an *instruction* (e.g., operator instructions); a *list* (e.g., code list, signal list); a *log* (e.g., maintenance log); a *plan* (e.g., maintenance plan); a *report*

(e.g., a test or inspection report); and a *request* (e.g., a change request).

An artifact might be built based on a standard, e.g., source code may follow a certain coding standard. Each artifact can pertain to requirements, blocks, hazards, and risks, as discussed in earlier sections. An artifact can be linked to other artifacts as well. For example, a design document may realize the requirements in the requirements specification, or a report could be the result of carrying out a plan. Issues that are identified during lifecycle activities are documented in reports. Like system blocks, artifacts can evolve over time and are therefore versioned and under configuration management.

IEC 61508 prescribes specific input and output artifacts for all the activities in the overall lifecycle. As an example, we have shown in Figure 3 the input and output artifacts for the Software Module Testing activity, whose goal is to verify that each software module performs its intended function and does not perform unintended functions. In the technical report version of this paper [10], we provide conceptualizations similar to that in Figure 3 for all the software lifecycle activities.

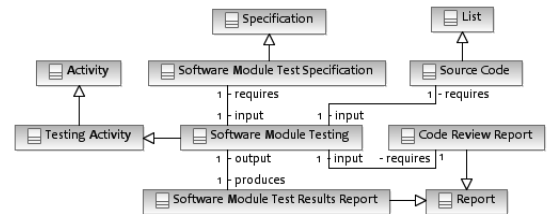


Figure 3: Software Module Testing Activity

Note that the links between the more specific subclasses of Artifact and these lifecycle activities (e.g., the link between Source Code and Software Module Testing in Figure 3) refine the high-level input and output links between Artifact and Activity in the conceptual model. Therefore, in Figure 2, the links between Activity and Artifact can be seen as derived (hence the ‘/’ before the link names). Further, note that the various artifacts in the standard need to be specialized in any given context. For example, the Software Module Test Specification in Figure 3 could be defined as being composed of test cases that exercise certain blocks or requirements. Similarly, the notions of test stub, and test driver could be made explicit for testing. Deciding about how much structure to enforce on each artifact is one of the key aspects of specialization (see Section V).

#### F. Issue Concepts

The concepts enabling the description of issues are modeled in the Issue Concepts package. Issue is the broad term we use to denote a point in question or a situation that needs to be settled in regards to a controlled item or a requirement (controlled items are discussed in III.G). Issues may represent defects, human mistakes, or enhancements and can be a result of activities concerned with Verification & Validation (e.g. testing and inspection) and safety assessment. In addition, enhancement may be proposed at different stages of development as a result of activities such

as requirements engineering and design, or in response to the findings of V&V and safety assessment. Defects can be further refined into errors, failures and faults. An error is a discrepancy between the actual and the ideal output. IEC 61508 distinguishes system errors from human errors, referred to as mistakes. Mistakes denote unintended results due to human action or inaction. A failure is defined as the inability of a unit to perform a required function, and a fault as the abnormal condition that causes a unit to fail (e.g., a software bug).

To illustrate these concepts, consider a boiler system. An error could be when the observed temperature is 80 degrees Celsius while the water is boiling, i.e., when the expected value is 100. If there is a safety requirement stating that the boiler should activate the pressure-release valve in case of over-heating (i.e., when the temperature has reached 100), then the error would lead to a failure, because the safety function would not be delivered. An error does not necessarily lead to a failure. In our example, if the actual temperature was 80 and the observed one was 60, there would still be an error but no failure. Failures and errors might imply faults. In our example, the fault could be a damaged sensor or the boiler's control unit incorrectly interpreting the temperature sensor output.

Mistakes made by an operator of the system can lead to failures. For example, if the safety function requires manual intervention and the operator fails to notice the alarm indicating an over-heating boiler, he would not engage the safety function. Mistakes may lead to changes to the operating procedures, or even the system. For example, the operating procedure may be changed to ensure that at least one operator is monitoring the control panel at all times; or the system's user interface may be revised to reduce the possibility of alarms going unnoticed.

The decision made about an issue (whether it is valid, and if so, how it has been resolved) is documented in a report. The resolution of an issue may induce change to some controlled items. Note that issues can be raised not only through the development stage, but also during operation, maintenance, decommissioning, etc.

#### *G. Configuration Management Concepts*

Valid issues need to be addressed through change. The concepts required for management of change and for ensuring that the safety requirements continue to be satisfied as the system evolves are captured in the Configuration Management Concepts package. Demonstration of accurate change management is necessary for compliance with IEC 61508. The central concept here is a controlled item, which is any item for which meaningful increments of change can be documented. In our model, blocks, artifacts and PESs are controlled items. Each controlled item may have some rationale to justify its existence, and assumptions to describe constraints or conditions about the item. Assumptions and rationale are further explained in Section III.H. Changes to controlled items are made in response to issues, as discussed earlier, and can be justified by rationale.

#### *H. Justification Concepts*

System development entails various decisions which need to be justified by reasoning and based on assumptions about the domain and the artifacts. The basic concepts to enable justification are provided in the Justification Concepts package. There are two concepts here, assumption and rationale. An assumption is a premise that is not under the control of the system of interest, and is accepted as true. A rationale is the reason that explains the existence of a controlled item or a requirement in the system. The rationale may rely on some of the assumptions that have been made about the concerned block or artifact. An assumption about a PES as a whole will have overarching affects whereas assumptions regarding a particular block may affect how it is designed and implemented. In safety-critical systems, assumptions play a key role. In particular, most statements about the safety of a system are tied to the assumptions made about the environment where the system will function [5].

#### *I. Guidance Concepts*

Many aspects of development are influenced by guidance from external resources. For example, a sector-specific standard or a recommended practice may mandate certain requirements that must be fulfilled by the PES; or the implementation source code may be expected to be based on a certain coding standard. Such external resources are captured using the Guidance Concepts package. The guidance package describes the various sources of advice and recommendations used throughout development. A standard provides formal recommendations on engineering or technical criteria, methods, processes and practices and can be either general such as IEC 61508 or sector-specific such as ISO 17894 [12] that provides principles for the development and use of PESs in marine applications . The recommended practice on the other hand may be much more prescriptive and specific, providing sound practices and guidance for the achievement of a particular objective. Either may be used as a measure of compliance.

#### *J. Domain-Specific Concepts*

Finally, the Domain-Specific Concepts package contains enumeration types that can be customized by defining specific enumeration values for a given context. The concepts behind the enumerations have already been described in the other package descriptions. In Figure 4, we show examples of the kinds of values that can be used for each enumeration type.

### IV. ILLUSTRATING THE CHAIN OF EVIDENCE

The conceptual model described in the previous section gives an overall view of the safety evidence pieces and the interconnections that need to be established between these pieces during the lifecycle of a safety-critical system.

Figure 5 shows a partial instantiation of the concepts in the model and their links. The hazard shown is the breakout of fire on an oil platform. The hazardous element involved is the combustible gas on the platform. The initiating mechanism leading to a fire breakout is the presence of a gas

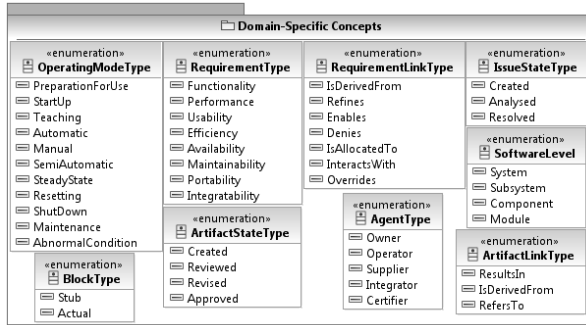


Figure 4: Example Values for Domain-Specific Enumerations

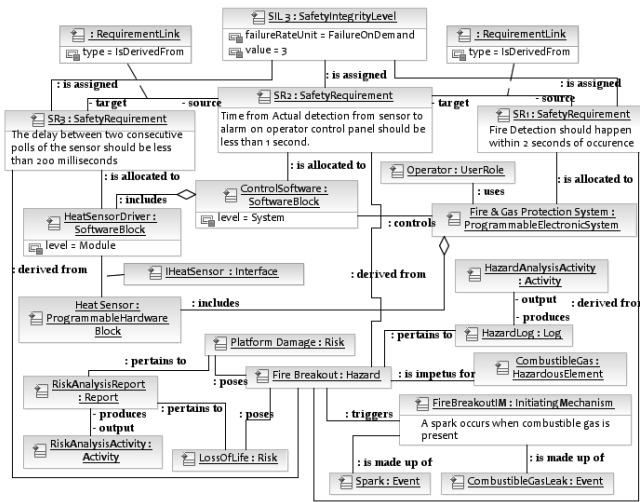


Figure 5: Example Evidence Information

leak and a spark in the vicinity of the leak. The hazard is identified during a hazard analysis activity and documented in a hazard log. For every hazard, a risk analysis activity is conducted and a report indicating the risks to mitigate is created. Two of the potential risks that such a fire can pose are damage to the platform and loss of life.

Based upon the hazard, safety requirements are derived and allocated to the various risk mitigation facilities. One such facility is the fire & gas protection system. The safety requirement allocated to this PES is that it must detect a fire breakout within two seconds of occurrence. A safety requirement for the software system is then derived for the software system that controls the PES, stating that the time from the actual detection of fire from the sensor until an alarm (visual and/or aural) is presented on the operator control panel is less than one second. This requirement is further partitioned between the control software and the heat sensor driver. The requirement allocated to the sensor driver is that it must keep the delay between two consecutive polls of the sensor to less than 200 milliseconds.

In this example, we can see the relationships between the different blocks, the requirements associated with each block, the derivation of lower-level requirements from higher-level requirements, the root hazard and associated risks, and the lifecycle activities. The example could have

been expanded to show a variety of other activities (e.g., design and testing) and artifacts (e.g., design specifications, test specifications and test results). All this information needs to be accounted for when a software safety case is being developed.

## V. SPECIALIZATION OF THE CONCEPTUAL MODEL

IEC 61508 is a generic standard and can be implemented and augmented in a variety of ways depending on contextual factors, including the characteristics of a particular application domain, and the development process and technologies to be used. Specialization is an important prerequisite for developing a coherent, IEC 61508-compliant safety information model, which can guide data collection and support analysis in a particular development context. The generic conceptual model we developed in Section III provides an intuitive and technically rigorous basis for describing specializations. As an example, we show how to define a special type of the Diagram artifact (see Section III.E), and use this specialized diagram for expressing Assumptions (see Section III.H).

In a safety-critical system, it is important to state the assumptions (e.g., about the operating environment) in a way that permits systematic analysis. This helps ensure that we can assess the validity of requirements, specifications, and design decisions and to verify that there are no conflicts between the required system properties [5]. A powerful and flexible notation for formalizing assumptions is the Parametric Diagram in the SysML modeling language [9]. This type of diagram is used for representing constraints on a system's property values. In Figure 6, we have shown an example parametric diagram.

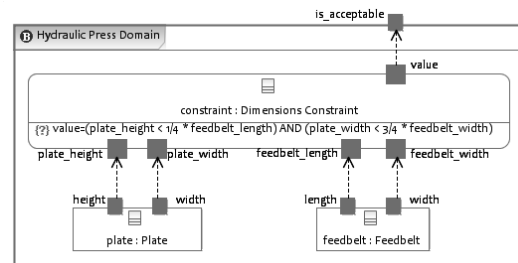


Figure 6: Parametric Diagram for an Assumption

The diagram describes a domain assumption about the physical dimensions of the plates that are fed to a hydraulic forging press. The assumption states that the height of a plate is no larger than  $\frac{1}{4}$  of the length of the feed belt that conveys the plate to the press, and that the width of a plate is not larger than  $\frac{3}{4}$  of the width of the feed belt. The former constraint is to ensure that the plate is small enough to be picked up by the robot arm that places the plate on the press table, and the latter – to ensure that plates would not fall off the edges of the feed belt while in motion.

If we want to develop a specialized standard or recommended practice requiring that a parametric diagram should be constructed for every assumption, our conceptual model will be extended as follows: A Parametric Diagram is defined as a subclass of Diagram and an association is

established between Assumption and Parametric Diagram. This is depicted in Figure 7.

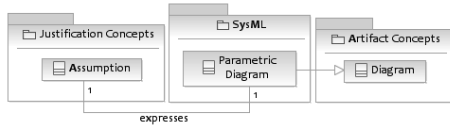


Figure 7: A Specialization of the Generic Model

In general, specialization refers to the extensions one makes to the conceptual model of Figure 2 in order to adapt it to a particular context. The extensions can be made by adding new classes (or subclasses), associations, attributes, and constraints. The example in Figure 7 already shows the addition of new (sub)classes and associations to the model. Below, we illustrate some simple extensions through new attributes and constraints. The model in Figure 2 is intentionally abstract, thus only providing the attributes that are fundamental to understanding the concepts. Any specialization of the model into an applicable, context-specific information model necessarily requires many new attributes to be defined. For example, most concepts need a universal identifier (uid), a name, and a description attribute. Constraints will be used frequently in the specializations of the model as well. For example, IEC 61508 highly recommends that module testing (see Figure 3) for safety integrity level 4 (SIL 4) should utilize probabilistic testing. If the certification body applying the standard wants to make this mandatory, it may choose to add the following OCL constraint to the model in Figure 2:

```
context SafetyIntegrityLevel
inv:
self.forAll(sil.value = 4 implies
sil.SafetyRequirement.Block->forAll(
b.SoftwareModuleTestResultReport.output.
Technique->exists(t.name = "Probabilistic Testing"))
```

The above constraint states that a module testing activity associated with a block that has SIL 4 requirements must utilize the probabilistic testing technique (we have assumed that each technique is identified by a name attribute).

A full specialization of our conceptual model will involve numerous extensions like the ones illustrated above. Once a full adaptation of our model to a particular context is arrived at, the resulting model can be used to drive data collection during the development process and to automate some of the most important and yet laborious tasks in the software certification process, as we discuss in the next section.

## VI. APPLICATIONS

Having described our conceptual model and how it can be specialized, we now discuss some important ways in which our conceptual model or its specializations can facilitate software certification.

### A. Aid to Understanding and Communicating IEC 61508

At the most basic level, the conceptual model we have developed helps improve understanding and communication of the IEC 61508 standard. Interpreting a standard like IEC

61508 is a daunting task for a software supplier. Even when attempted, the interpretation of a supplier is likely to be significantly different from that of the certifier. An issue we have noticed through our interactions with safety-critical software suppliers is that it is not always clear to them what documents and information, and at which level of detail, they are expected to provide in support of safety. Furthermore, it is frequently unclear what the scope of each document should be, and how these documents should be linked to hazards, requirements, activities, and so on. These issues typically lead to several unnecessary iterations to refine and complete certification documents as well as many misunderstandings that could have been avoided.

A concise but precise graphical representation of the core concepts in the standard such as the one we have developed here is a valuable and appealing aid for understanding and using the standard. In particular, the representation can be used by the certifiers to convey their expectations and to clarify the information requirements for demonstrating compliance.

### B. A Contract between Suppliers and Certifiers

After our conceptual model has been specialized to a particular context, it can be viewed as a formal contract between the supplier and the certifier. Specifically, from the specialized conceptual model, the supplier can determine what evidence the certifier expects, and can accordingly plan its data collection strategy. In the absence of such a contract, the supplier will not know a priori what evidence needs to be collected. This often leads to the supplier failing to record important information during the development process, and having to incur significant costs to recover the information later on. Having such a contract is also advantageous to the certifier, because it permits them to assume that the safety evidence provided to them has certain content and structure.

Hence, the certifier can optimize its safety assessment procedures according to the content and structure mandated by the contract. For instance, the specialization example we gave in Section V would bind the supplier to use a SysML parametric diagram for expressing assumptions. Hence, the supplier would know exactly how to express and record assumptions during development and the certifier would know exactly what they can expect during assessment and possibly build supporting analysis tools to analyze the consistency and completeness of assumptions.

Finally, the existence of a formal contract for safety evidence means that certifiers and suppliers may define a common language for the electronic interchange of data, for example based on XML. This offers an opportunity for automation of some laborious certification tasks, as we are going to describe below in Sections VI.C and VI.D.

### C. Automatic Safety Report Generation

Our conceptual model provides an abstract structure for the organization of software safety information. Once



tailored to a particular context through specialization, the resulting concrete structure can be used as the basis for a repository for storing development artifacts, process knowledge, hazard analysis data, safety audits, etc. This repository can be queried automatically for safety-relevant information, and the results then assembled into safety reports. For example, the links in Figure 5 can be traversed from the hazard to all the related elements, and a structured document can be generated to facilitate the verification of all the safety evidence related to the hazard. Modern model-driven technologies already enable the development of such infrastructure.

The main traceability requirement for generation of safety reports has to do with how software development artifacts (e.g., software requirements, architecture, design, and tests) are linked to the higher-level safety concepts such as hazards, environmental and domain assumptions, and overall safety requirements. Establishing this traceability is a key issue that one must consider when the conceptual model is being specialized to a given context and the specific artifacts to be used in that context are being defined.

We believe that using Model Driven Engineering (MDE) will facilitate the definition and exploitation of the traceability information that can be used for automatic software safety report generation. To illustrate this point, let us revisit the example we gave in Section V. The parametric diagram in Figure 6 not only alleviates any potential ambiguities about the textual description of the assumption, but also yields precise traceability links between the assumption and the involved system blocks, namely Plate, Feedbelt, and Hydraulic Press Domain (a super block). Hence, the chain of evidence is always maintained in a precise manner and can thus be queried automatically.

In contrast, if text-based documents are used, traceability cannot be strictly enforced. Further, the semantics of any traceability links established in text would not be machine-interpretable. As a result, the information cannot be precisely queried, without the danger of following false links or failing to follow the correct ones.

#### D. Automation for Rule Checking

An interesting use of our conceptual model is to define consistency and completeness rules based on the concepts in the model (or a specialization thereof) and then check these rules against the information in an evidence repository (see Section VI.C where we discussed such a repository). Here, we give two simple rules that can be articulated directly over the generic conceptual model of Figure 2:

- From the model, we see that each activity requires certain competence, and that each agent possesses certain competence. This coupled with the fact that competence itself can be defined in terms of encompassing other competence, can be used to define a rule for checking that an agent responsible for carrying out a given activity has the necessary competence to do so.

- Another example would be checking that each safety requirement derived from a hazard has indeed been allocated to a PES. This helps ensure that all derived safety requirements have been dealt with appropriately by some system component.

To fully realize this notion of automated rule checking, we need to have in place a specialized conceptual model based on which all the rules of interest can be articulated. We further need some type of rule checking engine that allows both the definition of the rules in some language and the verification of the rules written in that language against the development information of a system. For example, MDE technologies such as the Eclipse Modeling Framework [13] and its associated OCL engine [11] readily provide such capabilities. This ability is useful to both the suppliers and the certifiers of safety-critical systems. From the perspective of the supplier, the rules can be used to ensure that the system has been built according to some industry-specific standard or recommended practice, or even to perform impact analyses whereby specific rules could be defined to predict the impact of changes based on dependency information. From the perspective of the certifier, the rules could be defined such that the supplier provides the data from the model to the certifiers, according to some predefined interchange format, and the certifiers have some proprietary rules defined in order to partially check if the supplied information complies with its standard or recommended practice. The checking of whether the supplier is using competent agents to perform certain activities could be one such rule.

## VII. MODEL VALIDATION

Our conceptual model is based on a systematic analysis of the IEC 61508 standard and on the authors' experience. To validate the usefulness of the model, we participated in a number of safety-certification meetings for a safety monitoring system in the maritime industry. From the issues raised by the certification body during these meetings, we randomly selected 30 and analyzed whether the information captured in our model could have either prevented or helped to address the issues. These issues could be classified in seven categories as shown in Table 1. Categories in the first five rows could be addressed by information collected based on the conceptual model. These categories represent 56% of the issues (17/30). The last two categories correspond to completeness issues and argumentation flaws and are not directly addressed by our model.

TABLE 1: MODEL VALIDATION FINDINGS

Type of Issue	Count
Missing traceability links	2
Missing requirement type e.g. performance, availability	2
Missing mode of operation for requirement	3
Unaddressed certifier expectations (e.g. use of particular notation or technique)	3
Unclear delineation of system blocks and interfaces	7
Unstated requirements, procedures, assumptions	6
Argumentation problems (redundancy, ambiguity, and reasoning issues)	7

## VIII. RELATED WORK

Systematic development of safety cases (and more generally, dependability cases) is an important topic which is gaining increased attention in the dependability literature [2, 3, 5]. Kelly [1] provides an interesting and widely-accepted view on what a safety case is made up of. It divides a safety case into three parts: the safety requirements (or objectives), the argumentation, and the supporting evidence. The safety requirements are developed through various kinds of safety analyses (e.g., Fault Tree Analysis, and Failure Modes and Effects Analysis) and have been addressed extensively [14]. Building the argumentation in a safety case has been the focus of a lot of research in the past 15 years, e.g. [1, 3, 6, 15-17], with the Goal Structuring Notation (GSN) [1] as the basis for most of the work. However, there has been little research on providing a detailed characterization of the evidence underlying a safety case. What we presented in this paper is aimed to fill this gap for the software aspects of safety-critical systems.

The need for more effective collection and linking of safety evidence information has been noted before. In particular, Lewis [18] mentions the existence of a web of safety-relevant information covering not only the relationships between hazards and safety requirements but also between the operator of the system and operating procedures, the system functions and hardware elements, the system components and software tests, and so on. The conceptual model we developed in this paper provides a precise characterization of this web of information based on the IEC 61508 standard.

The sheer size and complexity of the documents comprising a safety case has been a constant challenge for safety assessors. The authors in [18, 19] propose the notion of electronic safety case, so that assessors can dynamically query a safety case for the information they need, instead of having to go through hundreds of pages of physical documents to find this information. As we discussed in Section VI.C, our conceptual model, when specialized to a particular context yields an information model for such electronic safety cases.

The authors in [20, 21] provide partial conceptualizations of IEC 61508, but they adopt a process-oriented view of the standard and thus focus on the processes involved when using the standard and assessing safety. The conceptual model we developed in this paper takes a much more holistic approach and captures all the key information concepts necessary to show compliance to the standard.

## IX. CONCLUSIONS

In this paper, we developed an extensible conceptual model, based on the IEC 61508 standard, to characterize the chain of safety evidence that underlies safety arguments about software. We showed through some examples how our conceptual model can be specialized according to the needs of a particular context and described some important ways in which our model can facilitate software certification. An analysis of a random sample of issues raised in certification meetings showed that a majority of

them would have been prevented or addressed by information collected according to our model. Applications of our model include: the precise specification of safety-relevant information requirements for system suppliers; defining a data model for developing a certification repository; the implementation of automatic, safety-relevant constraint verification (e.g., compliance with standard, recommended practice); and the automated generation of certification reports on demand. A detailed investigation of these activities and the development of appropriate tools to support them form facets of our future work.

## X. REFERENCES

- [1] T. P. Kelly, *Arguing Safety - A Systematic Approach to Managing Safety Cases*, PhD Thesis, University of York, 1998
- [2] S. P. Wilson, T. P. Kelly, and J. A. McDermid, "Safety Case Development: Current Practice, Future Prospects," in *Safety and Reliability of Software Based Systems: 12th Annual CSR Workshop*, 1995.
- [3] P. Bishop and R. Bloomfield, "A Methodology for Safety Case Development," *Proceedings of the 6th Safety-critical Systems Symposium*, 1998.
- [4] T. Kelly and R. Weaver, "The goal structuring notation – a safety argument notation," *Proceedings of the Dependable Systems and Networks - Workshop on Assurance Cases*, 2004.
- [5] D. Jackson, M. Thomas, and L. I. Millett, *Software for Dependable Systems - Sufficient Evidence?*: The National Academies Press, 2007.
- [6] J. A. McDermid, "Support for Safety cases and safety arguments using sam.," *Reliability Engineering and System Safety*, vol. 43, pp. 111-127, 1994.
- [7] "Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related Systems (IEC 61508)," International Electrotechnical Commission: International Electrotechnical Commission, 2005.
- [8] Rules for Classification of Ships, <http://www.dnv.com/industry/maritime/rulesregulations/dnvrules/rulesclassships/>, 2009
- [9] J. Holt and S. Perry, *SysML for Systems Engineering*: Institute of Engineering and Technology, 2008.
- [10] Lionel Briand, Thierry Coq, Shiva Nejati, Rajwinder Kaur Panesar-Walawege, and M. Sabetzadeh, Characterizing the Chain of Evidence for Software Safety Cases: A Conceptual Model Based on the IEC 61508 Standard., <http://modelme.simula.no/>, 2009
- [11] MDT/OCL, <http://wiki.eclipse.org/MDT/OCL>, 2009
- [12] "Ships and marine technology -- Computer applications -- General principles for the development and use of programmable electronic systems in marine applications (ISO 17894)," International Organization for Standardization, 2005.
- [13] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed.: Addison-Wesley, 2008.
- [14] C. A. Ericson, *Hazard Analysis Techniques for System Safety*: John Wiley & Sons, 2005.
- [15] T. Kelly and J. McDermid, "Safety Case Patterns – Reusing Successful Arguments," *Colloquium on Understanding Patterns and Their Application to Systems Engineering*, 1998.
- [16] M. Huhn and A. Zechner, "Analysing Dependability Case Arguments using Quality Models," *Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security*, 2009.
- [17] G. Despotou, D. Kolovos, R. F. Paige, and T. Kelly, "Defining a Framework for the Development and Management of Dependability Cases," *Proceedings of 26th International System Safety Conference*, 2008.
- [18] R. Lewis, "Safety Case Development as an Information Model," in *Safety-Critical Systems: Problems, Process and Practice, Proceedings of the Seventeenth Safety-Critical Systems Symposium*, 2009.
- [19] T. Cockram and B. Lockwood, "Electronic Safety Case: Challenges and Opportunities," *Safety-Critical Systems, Current Issues, techniques and standards*, 2003.
- [20] P. W. H. Chung, L. Y. C. Cheung, and C. H. C. Machin, "Compliance Flow – Managing the compliance of dynamic and complex processes," *Knowledge-Based Systems*, vol. 21, pp. 332-354, 2008.
- [21] Y. Papadopoulos and J. A. McDermid, "A Harmonised Model for Safety Assessment and Certification of Safety-Critical Systems in the Transportation Industries," *Requirements Engineering*, vol. 3, pp. 143-149, 1998.