CI: CSF-FFG-OT

Contextualized Safe Functions - building complex safety related systems

FFG CSF

August 24, 2021

Organisation:	OpenTech EDV Research GmbH
Working Group:	Safety Critical Linux Working Group
Author:	Nicholas Mc Guire

Release:	0
Revision:	1
Revision Number:	0.1
Date:	July 18, 2020
Expires:	—
Ref:	IEC 61508 Ed 2
Status:	**unreviewed, unauthorized, incomplete, Draft**
Format:	LaTeX
Tracking	GIT
QA:	initial review and authorisation
License:	Creative Commons

Contents

ChangeLog

Version	Author	Date	Comment
0.1	Nicholas Mc Guire	June 18 2020	structure, intro and complexity

Reviews:

Version	Reviewer	Date	Comment

Authorization:

Name	Organisation	Date	Signature

Status: Draft: incomplete, unreviewed, unauthorized

1 Introduction

Safety properties of systems and the ability to analyze the same are strongly interconnected. A system that can not be analyzed (decomposed into its parts) can not be understood. Not understanding a system is the root-cause of all uncontrolled hazards [1]. With growing complexity of safety related socio-technical systems new forms of analysis e.g. Sytem Theoretic Process Analysis (STPA)/Causal Analysis based on System Theory (CAST) [5] (see chapters 8 respectively chapter 11) have entered into wider use. Novel methods to structure highly-complex arguments have been developed in the past decade e.g. Goal Structuring Notation (GSN) [6]. What we felt lacking though was a robust process to analyze safety properties of Electric/Electronic/Programmable Electronic (E/E/PE) systems that no longer are simple enough that we can manage analysis purely be expert knowledge and experience (e.g. via Failure Mode and Effects Analysis (FMEA)/) With other words when we are to analyze systems that go beyond our ability to maintain a consistent mental picture of the overall system while inspecting details of a particular element. For such highly-complex system the method we are proposing - Hazard driven Decomposition Design and Development (HD^3) has been under development. In this companion document to the HD^3 process description and the tool manual we outline the conceptual underpinning of HD^3 in the context of IEC 61508 Ed 2 and how we propose to provide fully contextualized safety functions as a possible basis for building complex safety related systems.

The focus here is the conceptual side of layered analysis in relation to function level safety assurance of complex software. The intent is to develop the mapping from IEC 61508 Ed 2 to an Application Programming Interface (API) level consistency and completeness argument allowing to build complex systems safety arguments from a full decomposition of a system. The goal is not to allow for any notion of "out-of-context". We do believe that after a set of systems - notably if targeting a shared/domain context - a common function subset that shares the overall argument structure may emerge. This would potentially allow then to create a base-line in the long run. For now we will though only focus on achieving the initial set of fully contextualized safety functions.

1.1 Navigation

This structure of this concept document is as follows. We start by looking at a very specific way of looking at complex systems with a focus on systems that significantly utilize pre-existing elements. This focus is due to the assumption that no complex safety related system in the future will actually be a bespoke from-scratch development. After setting the stage with our view angle on complex systems we outline the system analysis method we propose — Hazard driven Decomposition Design and Development (HD^3). Finally we wrap up by looking at the notion of contextualized safety function which allows to specify API level elements in a specific context. This "boiling-down" to the API level is crucial as our underlying assumption is that complex software elements exhibit a high change rate and thus the actual challenge is not initial certification but maintaining the safe state over the life-time of the system.

The companion documents for this concept document are

- Hazard driven decomposition, design & development - Configuration Index (CI): HD3-SIL2-

[1] If one discounts the willingness to ignore known issues

OSADL

- HD^3 Tools Manual - CI: HD3Tool-SIL2-OSADL

These are not required for reading this concept document but may be consulted for specifics of the approach.

As pre-requisite for reading this concept manual we do assume some level of familiarity with IEC 61508 Ed 2 (2010) part 1 [1] and IEC 61508 Ed 2 (2010) part 3 [2].

We also would recommend reviewing IEC Guide 104 [3] and IEC Guide 51 [4].

1.2 Competency

Without that these quotations are exhaustive in any way - we cite the following clause from IEC 61508-1 Ed 2 which highlights an issues significantly aggravated by the complexity of the system. In complex system we must assume unfortunately that no engineer has an adequate competence encompassing the entirety of involved technologies. This is only compensated to some extent by calling in domain experts "at suitable points" since systems do not exercise technologies in isolation but in concert as much as they do so with the elements implementing these technologies. Thus an analytical framework suitable to manage complex systems must digress into technologies with a system specific focus to allow mitigating this systematic engineering weakness.

IEC 61508 Ed 2 points out this general issue explicitly in clause 6.2.14 i) of part 1.

> *The appropriateness of competence shall be considered in relation to the particular application, taking into account all relevant factors including:*
> *...*
> *i) safety engineering knowledge appropriate to the technology;*
>
> [IEC 61508 Ed 2 6.2.14]

Wile IEC 61508-1 Ed 2 6.2.14 i) pointed out the general need for technology related knowledge sub-requirement f) stresses this point even more clearly.

> *f) previous experience and its relevance to the specific duties to be performed and the technology being employed the greater the required competence, the closer the fit shall be between the competences developed from previous experience and those required for the specific activities to be undertaken;*
>
> [IEC 61508 Ed 2 6.2.14 f)]

For novel systems employing novel technologies that demand will overwhelm any individual. We see the strategic solution in building on teams while focusing the analysis on the specific context. This is not an optimization in any way - this is probably mandatory to manage complexities we are confronted with.

2 Complex Systems

The next generation of safety related systems is targeting inherently complex technologies be it AI/ML or sensors that exhibit data rates and diversity far beyond traditional safety related systems. Complexity of systems manifest it self primarily through the presence of element interactions. These interactions may be via formal interfaces or as side effects/side-channels [**?**]. We note that the term complexity as used in everyday life and in engineering does not really fit the way IEC 61508 Ed 2 and related standards use it. Complexity is perceived when functionality of the aggregated elements provides emerging functions that go beyond the elements capabilities - the view point is thus primarily from a functional angle. Before digging into the systems we want to build that are readily identified as "complex systems" we will try to develop an adequate view from the safety perspective.

2.1 Type-A/Type-B systems

A second viewpoint from which to look at these systems is from the safety engineering perspective. Safety standards often times seem highly abstract and over-formalized "rule-books" - actually they are sound derivations (in most cases) of first principles projected onto the technological realm of elements destined for use in safety related systems. One of these first principles is the intent of safety engineering to transform systems, which all start as unsafe, into safe systems. The role that the safety engineering process plays is to transform the initially unsafe system into a system with societally accepted residual risk. Risk is nothing absolute in any way [**?**] [**?**] but rather highly context dependent - we tolerate orders of magnitude higher risks in homes (personal responsibility and perceived control) than we do in air traffic [**?**]. This implies that this transformation from unsafe to safe is highly context specific and not "just" a technical issue. This also is one reasons why standards for different areas of use can differ quite substantially.

How does this transformation happen - how does an initially unsafe system evolve into a safe system ?

Every system conceived is to some extent a new system or we would hardly engage in its construction - this novelty is a source of uncertainty and consequently a potential source of uncontrolled hazards. Systems or elements of the same, that we do not understand fully - fully meaning in the context of the system - are the key source of uncontrolled risks. A system where we do not know - or equivalently - have not anticipated based on our causal knowledge - all potential failure modes and thus these can not be controlled. Something that is not understood fully - how it interacts with the other elements in the system and how it impacts them by its behavior (impact conversely to interaction describing the informal or unintended interfaces) can not be fully controlled with any defined assurance.

Essentially this is what the above noted transformation is about - gaining adequate understanding **in context** to allow identification of intolerable risks and subsequent elimination - or where this is infeasible - mitigation of these risks. What exactly "adequate" means is of course not quantifiable in any way but left to the responsibility of engineering and certification.

IEC 61508 Ed 2 has a simple hierarchy in its system model. This hierarchy is roughly that systems are build of elements which in turn are built from components. This hierarchy will pop up in a number of definitions so it is helpful to keep this in mind:

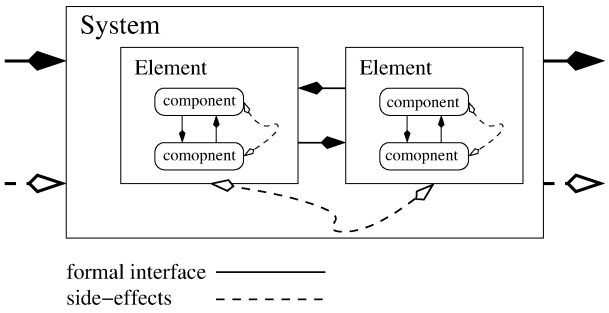

Figure 1: system-element-component relation

To achieve this transformation we need to gain full understanding of the system - only by this can we reasonably exclude unacceptable residual risks lurking in the system. To now do this we will approach the system from the element level.

Type-A element in IEC 61508 Ed 2 [2]

An element can be regarded as type A if, for the components required to achieve the safety function

- *a) the failure modes of all constituent components are well defined; and*
- *b) the behavior of the element under fault conditions can be completely determined; and*
- *c) there is sufficient dependable failure data to show that the claimed rates of failure for detected and undetected dangerous failures are met.*

b) is now where the interaction is "hiding" - the element is treated as aggregate of component but we are interested in the behavior of the element under consideration of the components behavior under **element** level fault conditions that were defined in a). Effectively this innocent looking clause captures the nature of complex system - for simple systems it naturally applies in its simplified form of $element = sum - of - components$ for complex systems this is no longer the case but the above clause does not limit type-A to simple elements in any way. What it does do is limit type-A systems to "fully understood" elements.

Note that c) is listed in the context of hardware and that IEC 61508 Ed 2 considers software to only fail due to systematic faults thus dependability data on the software it self does not really make sense - dependability data on the software process (which again permits for random (human) faults) may well make sense in the context of software though. As soon as incompleteness of analysis enters the picture though dependability data as substitute for causal understanding may play a relevant role in practice.

The crucial point to see is that it essentially is about full understanding of an element and by doing so gaining the ability to anticipate consequences initiated by faults in the element respectively failures resulting from incompatibility of the element with other system elements There is mounting evidence that it is no longer element level faults as in "did not meet its requirement" but rather interaction faults based on "it correctly implemented the wrong, in context of **this) system, requirement"**

[2] see IEC 61508-2 Ed 2 Clause 7.4.4.1.2 for type-A respectively Clause 7.4.4.1.3 for type-B

see Leaveson pp X. Once we have this ability established we, in principle, can build adequately safe complex systems. This buildup is by aggregation of fully understood elements which then allow the next level of analysis. Building this hierarchy of understanding is one of the key challenges we see in contemporary complex safety related systems. If how-ever the development of complex system can go bottom-up is a different issue addressed later.

Type-B elements on the other hand are then simply elements that violate **one of** the above requirements a-c. We need all three parts - knowing all failures, understanding the behavior under fault condition and the adequate data to justify made claims - to reliably anticipate the system level behavior. Note the restriction in - IEC 61508-2 Ed 2 7.4.4.1.3 *NOTE This means that if at least one of the components of an element itself satisfies the conditions for a type-B element then that element will be regarded as type-B rather than type-A.* - so while divide and conquer is one of our prime weapons to manage complexity - IEC 61508 Ed 2 deliberately limits this when it comes to the analysis of failure modes and behaviors under failure conditions. Notably this means that existing "well understood" elements - with all their benefits - need to be re-contextualized for any new system to **stay** safe [**?**].

Systems on the other hand are not just collocation of independent elements they are aggregations of interacting and thus dependent, elements. So the above noted failure modes are not context free - they are essentially the relevant failure modes in the context of the specific system. If we use a vehicle for training on a closed-private area where no person can be exposed to a failure mode of the system - we can legitimately ignore a large set of failure modes of the vehicle. The corollary to this is that re-using an element in the context of a different system might well "go out-of-context" and thus no longer cover all failure modes or we lack the understanding of the behavior under failure condition. Most notably this applies to c) above which addresses assurance of the claims - almost by definition this is not the case as soon as we build a novel system. To now get from the element to the system safety property we need to lock at a related concept - that of a low-complexity system.

The definition of low-complexity system differs seemingly only a little but in a crucial point. This crucial point is expressed well in the definition given in IEC 61508-4 Ed 2 Clause 3.4.3. While the elements level faults are in context of the element - even if activation of a failure mode may be from external sources the failure mode is inherently a property of the element. But the behavior of the element under fault condition does not necessarily reveal the system behavior under this fault condition. This is expressed in the second condition for low-complexity systems:

- *the failure modes of each individual component are well defined;*
- *the behavior of the system under fault conditions can be completely determined.*

The systems failure modes are thus at least initially (that is before any elimination/mitigation took place) the sum total of all element failure modes in addition to any failure modes that result from element interaction at the system level. With full understanding of the elements the interactions at system level are not yet covered - but adequate understanding at element level should let us anticipate interactions. Thus if our goal is to transit from a type-B to a type-A system we must also gain adequate understanding of the elements in context of the specific system. Explicitly: the sum of a set of Type-A elements is not automatically a low-complexity system! This means interaction

faults as well as impact of detected faults (e.g. alarm "showers") need to be considered [3]. The system after all is more than the sum total of the elements with respect to both intended behavior and unintended behavior.

The pre-condition to building a safe system is to know all possible and credible failure modes and understand the system under fault condition. The addition of "credible" is crucial here - there are infinitely many failure modes of any system - they all could be hit by a meteorite - but generally we simply neglect those failure modes or events that we simply consider infeasible. We have been wrong about this in the past - e.g. Titanic's sinking was considered infeasible - justifying the limited availability of life-boats. For a more recent example - Fukushima II and the risk of a tsunami unfortunately showed a similar pattern [4]. In the jargon of IEC 61508 Ed 2 we could restate this as "only low-complexity systems are safe" - note though that this is in stark contrast to what most people would consider to constitute low-complexity. The confusion put explicit is that technically high-complexity systems could be low-complexity from a safety perspective.

If one accepts this somewhat odd terminology - then we can move on to asking how we can actually achieve this state of system knowledge corresponding to IEC 61508 Ed 2s definition of "low-complexity system". Essentially it boils down to:

NOTE Behavior of the system under fault conditions may be determined by analytical and/or test methods.
[IEC 61508-4 Ed 2 Clause 3.4.3 Note]

Thus analysis and testing. Thats it. The critical question to ask now is what and how can we achieve the goal of full contextuallized understanding of the system - what constraints and conditions must analysis and testing then satisfy ? and what process can be used to actually allow attaining the overall goal of fully, or at least adequately, understanding the system ?

2.2 Novel System Decomposition

Analysis of system - and we learned this as a children - starts with with its initial Greek language meaning - to take apart or decomposition. Some parents may have viewed our first decomposition attempts as inappropriate destructions - but unless they were system engineers we should forgive them this misunderstanding. Humans have significant problems with complexity we are good at analyzing relatively small elements, understanding their observable behavior and - crucially - anticipating the unobserved behaviors we consider potentially feasible for the element under study. We do not need to drop a chair from the 5th floor to "know" that it will be destroyed at ground level impact. Essentially this is the key to our ability to analyze system - IFF we understand them as well as we do dropping chairs.

Technical, or more precisely socio-technical, systems are built up from existing elements (be it at the physical or conceptual level) which we anticipate functioning in concert with further elements to shape the desired novel system. This is a highly creative process and - as reward driven creatures - we quite naturally focus on the perceived benefits of the new system. To now create complex systems we decompose the same to a level at which we can populate each of the building-blocks with enough

[3] This needs to be expanded to undetected faults as well - which will be addressed later when we look at false-positives/false-negatives

[4] And we could ask if building a 260m tower in Tokyo - a known earthquake area - is justified no mater how good our understanding is - humans seem to enjoy pushing the limits until the limits push back [**?**]

perceptual assumptions so that our anticipatory capacity suffices to imaginate the high-level goal of the undertaking - then new capabilities/services that the novel system shall provide. I would consider this goal focused approach quite natural for humans - but it has some undesirable consequences when we are building systems that induce risks.

Essentially there are two paths to decomposition of systems that pose safety risks (and a plethora of intermediate variants of course).

- Functionality driven decomposition - safety superimposed as mitigations after the functional model is created.
- Safety driven decomposition - risk minimization selection of available alternatives and tradeoff with functional constraints.

The first is the "natural" approach with the generally undesired side-effect of considering safety too late and missing opportunities to eliminate risks without obscuring the goals of the novel system. One reason for this is our ability to re-use experience and known elements notably as many developments follow a evolutionary path. So initially we often see a new system in the context of known systems or elements and perceive the extension as not actually changing fundamentals of the system we know and understand (or at least think we understand...). This essentially starts out-of-context with respect to the new system and thus is unsafe (as all new systems are) If we now continue the decomposition at the functional level we may achieve optimal systems from the functional or economic perspective but are unable to eliminate risks. The reason I believe is that viewing systems in context of existing systems implicitly "resolves" a lot of hazards or issues without ever being made explicit. Explicating things though is the only way we have to make new systems, for which we have no actual experience, understandable to the point where we can anticipate all (or at least most) credible failure modes and understand the systems behavior under failure conditions. We will get to the details of safety driven decomposition - or as we will call it "hazard driven decomposition" a bit further down - first we need to look at how safety standards require us to gain adequate understanding of a system for which we anticipate lurking hazards by transforming it from a type-A system into a type-B system systematically.

2.3 IEC 61508 Ed 2: Build Understanding

IEC 61508-1 Ed 2 "Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements" is - as the title states - focused on safety related systems that intend to employ E/E/PE systems to provide safe functions. Part 1 is addressing the general or "technology agnostic" needs. Even though this standard is focused on E/E/PE elements as part of a safety-related system it starts out with a very generic safety requirements development (Clause 7) which we focus on here. To extract the main goal of IEC 61508 Ed 2 here we focus on the objectives or in the case of 7.1 the key points in each of the sections:

- 7.1 General
  - Systematic approach of dealing with safety functions and their integrity guided by a conformant overall safety life-cycle.
  - Coverage includes risk reduction and E/E/PE safety related systems.

  - To achieve the technical safety goals specification of related processes like management, documentation, assessment, etc. and their interfaces are outlined.

- 7.2 Concept
  - *"The objective of the requirements of this subclause is to develop a level of understanding of the EUC and its environment (physical, legislative etc.) sufficient to enable the other safety life-cycle activities to be satisfactorily carried out.*

  Nothing to be added in this case.

- 7.3 Scope definition
  - Determine the boundaries of the EUC and the control system
  - Likewise determine the scope/boundary of the related hazard and risk analysis - with other words which hazards and risks impact or are related to the system under study.

- 7.4 Hazard and risk analysis
  - Determine the *hazards*, *hazardous events* and *hazardous situations* for all *reasonably foreseeable* circumstances - with other words all credible hazard related information.
  - Find the event sequences for all credible hazards and the associated risks.

- 7.5 Overall (system level) safety requirements
  - Taking all the previous collected information and understanding as well as the anticipated potential developments one now is ask to develop the set of system level safety requirements so that none of these identified hazards poses an intolerable risk any more.

- 7.6 Allocation to hardware, software and other technologies
  - Finally allocate these safety requirements to the elements of the systems that shall provide the necessary risk control.
  - Given the hazard potential different integrity levels - roughly the rigor with which the technical and management process is to be executed are assigned to the different mitigation measures.

With the early DLC phases of IEC 61508 Ed 2 Part 1 one effectively is establishing adequate understanding of the systems hazard potentials, determining elimination and mitigation options and finally selecting the best suited and allocating them to specific elements in the system. At this point the system "behaves" like a type-A system conceptually:

- All credible failure modes are known
- Behavior under failure condition is understood
- Adequate evidence/assurance available - see 2.4

What is missing is the data justifying the claims - which will be developed along the way during the life-cycle for hardware, software and other technologies notably the later includes procedures to be adhered to by operators while not further addressed in IEC 61508 Ed 2

2.4 Re-use or Invent

In general an evolutionary path - starting out from a known and understood element and evolving it by modification to fit to the specific novel target - has clear advantages when building safety related systems. To harvest this advantage one can employ pre-existing elements. The main benefit of the same is that they - if widely deployed in the non-safety domain - come with readily available information on structure, interfaces and most notably historic defect reports. In the case of free-libre-open-source software FLOSS - they also come with significant DLC data from the open development. Further access to intent via querying of authors and/or organizations that had contributed specific code-elements/drivers is a valuable input to gaining in depth understanding of behavioral aspects.

In bespoke development we would enter into the requirements phase for hardware and software based on the full understanding of the systems intent though with the details open. Essentially this understanding is - ideally - technology agnostic in the sense that it would be conceived without any specific solution space in mind. In practice this is of course rarely the case simply due to the generally evolutionary nature of engineering. For low complexity elements this top-down approach has significant advantages as the underlying implementation space can be viewed as understood and engineers are able to design such systems with a significant implicit knowledge base regarding language, implementation structure (e.g. OO design). Thus this reliance on ad-hoc cognitive perception of potential solutions to guide the de-composition of the design is crucial to building robust and safe systems.

The alternative of now entering into a bespoke development is to re-use existing elements. The down-side being that these elements have properties that were not derived from the needs of the specific system and thus - by definition - do not match the specific needs. This mismatch need not be dramatic but the match is at least not complete given the element "as-is" without additional constraints. While pre-existing elements come with the advantage of (credible) failure mode and experience based behavioral knowledge they incur the disadvantage of limited fit respectively inclusion of unnecessary, possibly undesired, functionality.

If elements are highly complex though then the advantage of behavioral information and failure data respectively the ability to inspect actual implementation to guide design will over-top the disadvantages. We assume that engineering staff is not able to envision an element as complex as the Linux kernel or even glibc "from scratch" and by doing provide a significant contribution to understanding of the systems behavior under fault condition. Thus at some point building on pre-existing complex elements out-performs any top-down engineering from scratch. Exactly where this boundary lies is of course hard to pin-point - that this boundary is crossed in the case of a full featured security aware multi-core/multi-user/multi-platform OS is intuitively "obvious" to me. That this intuition is not a safety relevant artifact is though clear - we must bolster it by demonstration of the advantage rather than by simply claiming it.

From a safety perspective a key aspect is that the element, to some extent, is being used in the wrong, that is, not the initially envisioned, context. The effect that this can have is that requirements do not fit precisely. This is true all the more when the documentation of these initially assumed requirements is limited due to the non-safety related nature of the development which sometimes does imply a limited rigor. This can be compensated in the case of open-source elements though if these provide not only the freedom of inspection but also provide a significant development history. This is the case with many open-source elements of interest that may be traced back to there infancy

in many cases. Thus while the top-down nature of the intent for a new system does not change - we regain the ability to envision behavioral aspects of the complex system if built on pre-existing elements notably if these are open-source. Again at some point this freedom to inspect will be overwhelming compared to the ability to envision - I would claim that this point has been long past though with no formal criteria to offer.

2.4.1 Re-introducing elimination

A major criticism regarding re-use of pre-existing elements is the perceived inability to eliminate hazards rather than needing to mitigate them. For a given pre-existing element this does seem to be the case [5]. Taking a broader view of selecting from a set of element candidates though the picture changes. One regains the ability to select not only the best fit from a functional perspective but also to select the best-fit with respect to available data. Note that while IEC 61508-2 Ed 2 7.4.4.1.2 is typically read in relation to hardware and calling for *c) there is sufficient dependable failure data to show that the claimed rates of failure for detected and undetected dangerous failures are met* is generally seen as restricted to hardware faults that exhibit random failure behavior. In the context of pre-existing elements undergoing an evolutionary development life-cycle we extend this to include historic defect data on complex software. The rational being that we are not implying random faults in the software it self but random faults in the engineering process that generated the software. There is not good reason to assume that faults would be systematically inserted in the development of pre-existing elements (...not at least related to safety properties - security has a depressing track record of precisely this malicious misuse).

Selection of elements is thus an avenue to re-introduce hazard elimination. To foster this adequate analytical rigor and depth is needed to allow risk informed decisions to be made when selecting elements. The context that comes with a pre-existing element allows judgment - to some extent at least - of high-level properties be it code-quality, consistency or freedom from some undesired properties. This judgment will not be at a line-by-line level but rather at the level of the process that emitted the pre-existing element. Again FLOSS elements bear the advantage of inspection beyond the pure functional level - if appropriate selection criteria for element candidates are chosen. The pre-condition for this ability though is the achievement of adequate analytical depth of the systems intent so that the available data can be utilized when making risk based decisions on pre-existing elements. The criteria for such selections are readily available for software elements in the form of properties per DLC-phase provided in IEC 61508-3 Ed 2 Annex-C. So while the system requirements and architecture design for a new system even more so for a novel system, will stay top-down bespoke development, answering risk related questions at the element level prior to allocation of safety requirements, is doable when utilizing pre-existing elements. Thus the top-down process of system evolution can be supported in the earliest phases of design by credible bottom-up information on the candidate elements. This dramatically reduces the need to envision future element behaviors as is the case for bespoke development.

It is crucial to point out that the argument for hazard driven decomposition and system development based on pre-existing elements is not an economic argument it is at it roots a technical and even more so a safety argument. Complex safety related systems - we are convinced - mandate the use of

[5] Note though that IEC 61508 Ed 2 7.4.2.13,f does not the ability to "remove" undesired features - albeit at the cost of no longer actually using the pre-existing element

pre-existing elements simply to ensure that adequate understanding of the system **can** be achieved. The goal after all is

- All failure modes known, and;
- Behavior under failure condition understood, and;
- Adequate data to justify claims.

which can not be achieved through delayed establishment of element behavioral knowledge in complex systems - we actually need behavioral insight during the early requirements and design phase.

2.5 The goal of layered analysis

Satisfying the analytical needs of safety related development is what we consider the main challenge for complex systems. As argued above, we see pre-existing elements, and most notably FLOSS elements, as a major remedy for the imponderability of complex safety related systems. The information is, at least potentially, available what is left to do is to structure the digestion of this vast body of information in a form that allows not only justifying an initial claim of safety properties, but maintaining the safety claim over the highly dynamic modification cycles of these elements.

The standard approach to large problems is partitioning. In the case of highly interrelated problems, layering of the applied abstractions can refine our ability to reach a consistent and complete view of the problem. With respect to safety related systems our focus is initially on requirements and design faults. A not to underestimate issue is selection of adequate technologies. This becomes e-Vern more critical in the case of novel systems where our engineering experience and backup expert knowledge is limited.

The abstract partitioning we propose to use - that is that partitioning that is system independent - will be outlined briefly. Roughly the partitioning happens in two parallel strains of decomposition:

- layer specific: essentially the framework or abstraction regime used to look at different levels for the specific functional and non-functional aspects being developed.
- problem specific: quite traditional divide and conquer at the functional and non-functional level driven by the overall intent of the system under development.

Up to here the discussion is mostly quite independent of the HD^3 method being developed. We are convinced that the principle approach stays valid even if the process/tool level approach we have been following should show to be incomplete or inconsistent or simply economically not acceptable. The principle behind the approach is simply that the high-level goal of IEC 61508 Ed 2 can be summarized as "manage/control complexity" which is the basis to give any assurance. Of course if complexity is managed by utilizing low-complexity system the goal clearly is achieved - Our implicit assumption behind this effort is thought that future safety related systems will be — necessarily — complex. Thus a suitable approach to regain analytical control of the systems will be needed so as to be able to give adequate assurance for future socio-technical system.

The sections following are specific to our proposed process/tooling and should be viewed as "one-of" the possible solutions, not as **the** correct solution. It is our assumption that if the proposed method

works at all it will not be long until a better, more efficient and notably more effective method will be found. The goal is simply to enable the utilization of highly complex technologies to enhance safety related systems and to avoid de-coupling safety from the overall technological development.

2.6 Introducing interaction analysis

One aspect raised a few times above is the presence of interaction faults. Elements may be fully specified and actually operate fault free - never the less the system may fail due to element interaction not having been considered. Basically there are two strategies to address this.

- Grow the scope of success if analysis from component to element to system. This way at every step the interaction of the components/elements could be discovered. The pre-requisite here though is that adequate information about the internal behavior is available (and understood !) so that the corner cases (and in general it is corner cases that lead to failures at system level) can be correctly analyzed and mitigated.
- Introduce a defined interaction scope for components/elements and analyze the components/elements in context of the potential interaction events. This would assume that analysis goes clearly beyond the formal interfaces. And in practice this has often been done to some extent (e.g. thermal interaction of hardware elements, side-channels in threat analysis). For software this to our knowledge is generally not done at component level (maybe with the exception of coded processes that actually do mitigate all potential interaction faults at the hardware level - though not covering requirements and design faults).

One can now combine these strategies - but the question is - do they scale to complex systems ? Our assumption is that its not just about scoping issues, the issue is - at what level of abstraction is the analysis done ? Our assumption is that for highly complex systems managing of interaction faults needs to de-couple from the specifics of an implementation. This de-coupling is necessary to "allocate the blame" properly with other words, is it the elements implementation, the design or requirements or the incompatibility of the same with the systems expected requirements, design and implementation (behavior) of elements/components ? We assume that it is the later that is actually the limiting case for understanding system behavior.

2.7 Putting it together

The big picture in context of significant use of pre-existing elements - and this is now very SIL2LinuxMP specific and by no means the only way to approach it - is to have three distinct columns of actions. These columns are strongly intertwined and need to evolve in concert.

With this big-picture we are now ready to look at the HD^3 process it self.

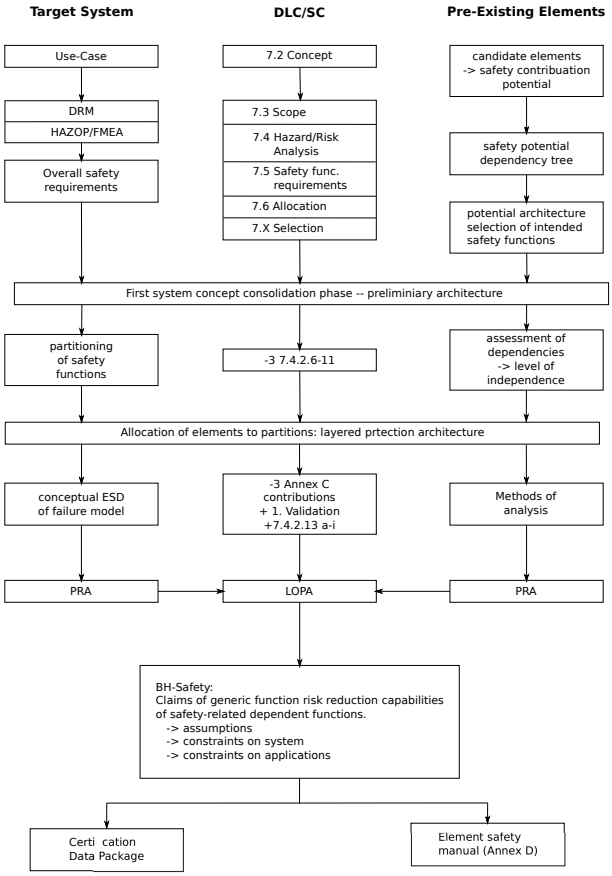

Figure 2: Pre-existing element focused DLC for complex systems

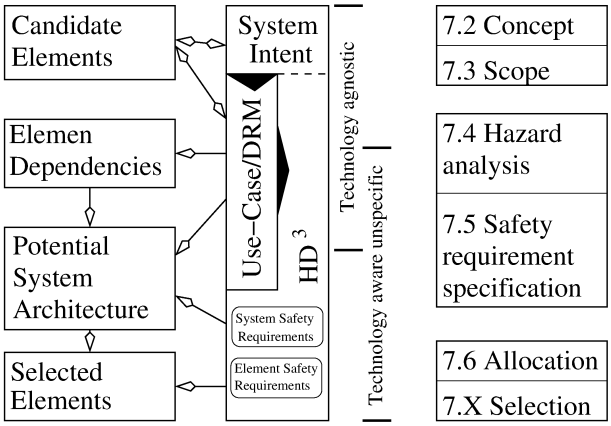

Figure 3: Early DLC phase - corresponding to IEC 61508-1 Ed 2 - HD^3 view

The three columns are the Use-Case and its decomposition, the standard(s) and mapping/interpretation and finally the element candidates that are iteratively joined into a the system architecture and then refined. The big picture of this life-cycle model is depicted in 2 whereby the main effort to gain adequate understanding is in the top part - basically prior to extracting the initial preliminary architecture. To emphasis the relationship from the analytical perspective and also roughly allocate analytical digression from the intent down to the technology aware unspecific analysis 3.5 we re-state the above phase model with the emphasis on the role of HD^3 3. Note that the intent is an informal statement describing the overall motivation and functionality of the planed system. In the trial cases studied up to now this was generally a two or three line statement only - the rest should ideally emerge from hazard driven analysis.

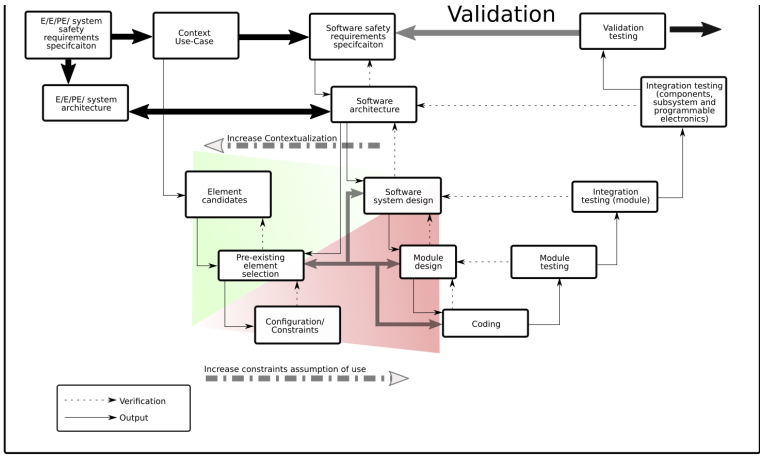

Figure 4: DLC adjustment for systems utilizing pre-existing elements

With respect to the sequential, or rather iterative, dependency of bespoke and pre-existing elements, the contextualization vs. constraints/AoUs is depicted in the adjusted V-model 4.
The key point to note is that the DLC adjustment for the pre-existing elements impact the bespoke development and vice versa.

3 HD^3 layered analysis - high-level introduction

In this section we will give a high-level introduction to HD^3 in context of the above safety principle outlined. The layering model, focus and the relation to the overall safety awaere development model of SIL2LinuxMP. As noted this is an attempt to translate the principle into a concrete solution. Weaknesses in this section are expected and some actually known - this after all is still work-in-progress. For the process level details and work/information-flow see [10] : HD3-SIL2-OSADL.

While this layering model was developed primarily to allow contextualization of complex pre-existing software elements it also relies on accessibility of significant hardware specific information. Essentially all analysis is in system context - the goal is not to remove or abstract the sofware elements but rather to structure context in a manageable and notably maintainable way.

The overall structure is summarized in Figure 5

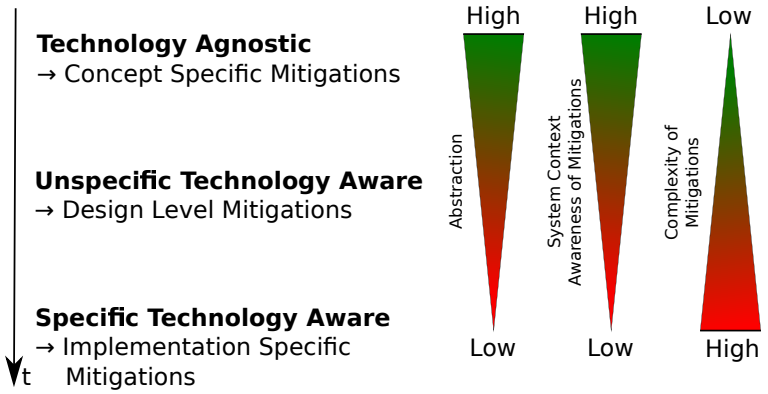

Figure 5: Layering and complexity in the HD^3 analysis process

3.1 Intent

The starting point is vague. Essentially it is a one-liner statement about what the overall intent of the novel system is. The rationale for this is that one needs a mental picture as guidance on the one hand, on the other hand this guidance should not dicate the technological or functional solution or narow the general solution space.

For the concept of intent we start at the Hazard and Operability Study (HAZOP) standard [9] definition of design intent:

> *design intent*
> *designers desired, or specified range of behaviour for elements and characteristics*
>
> [IEC 61882 Clause 3.2]

For a concrete example - a water monitoring system - this might translate to:

Measure and report E.Coli -specific enzymatic activity per volume of sample (indicating the level of fecal contamination) in at most 15 minutes, with defined and verifiable level of assurance.

[Colimínder Design Intent]

The boiling down of the solution space to the specific solution should be driven by continuous layered hazard analysis and hazard derived safety application conditions. These will generally be technical in nature but also may include constraints or procedural demands e.g. on the organization.

The intent statement is decompose into "obvious" functional units as the starting point for the formal analysis method. Again this is vague - what constitutes "obvious" is obviously... not that clear. It is though not that important as the effect of a bad initial decomposition is - if we devised the process properly - still safe, albeit just not efficient from a functional or economic perspective. After an initial review of an ad-hoc decomopsition for the above shown intnet 3.1 the refined ad-hoc decomposition was:

- Take "average" water sample
- Put 5.5ml sample in a clean bottle
- Dosage puffer (400ul) and substrate (20ul)
- Stir sample with controlled speed to get homogeneous mix up and heat controlled at reaction temperature (so as not to kill the bacteria → No thermal hot-spots)
- Measure fluorescence response of enzymatic activity over time (while stirring and maintaining reaction temperature) → read off slope
- Calculate mMFU
- Time supervision (ensuring $t \leq 15min$)
- Runtime verification of correctness (assurance)
- Report contamination

For a more detailed discussion see the HD^3 documentation [10] : HD3-SIL2-OSADL. section on design intent. What should be clear from the above example though is that this is in no way generic - this is totally system specific. So the context of the entire analysis is derived from the specific system and generic comopnents viewed from this specific perspective. Quite clearly this will not be possible without an adequately trained domain expert in the analysis team. In this sense HD^3 team members competency is quite comparable to traditional HAZOP [8].

This first decomposition is then used as the basis for entering into a guided systematic explorative analysis to derive the full context of all constituent elements/components/functions (as well as processes in some cases). This layered approach shall ensure that the allocation of hazards is correctly done. A hazardous technology is hazardous no matter how you implement it - e.g. there is no safe

Nuclear Power Plant (NPP) - it would not be reasonable to focus ontechnicalities and "fixes" if the underlying technology it self is the culprit. By degression through a set of semi-defined layers we intend to improve the allocation of hazards to it orign. This is a pre-requisite to allow effective trade-offs. A further crucial aspect of a hierarchical approach is to support incident analysis as well as change request impact.

3.2 Concept of Guidewords

Explorational analysis based on guidewords is not new. The use of HAZOP is quite well establshied [8] and has found its formalization in the in the International Electrotechnical Commission (a non-profit, non-governmental international standards organization) (IEC) 61882 standard [9]. Guidewords are formally defined in IEC 61882 as:

> *guide word*
> *word or phrase which expresses and defines a specific type of deviation from an elements design intent*
>
> [IEC 61882 Clause 3.5]

The idea behind the use of guidewords is twofold:

- Guide the analyst with respect to what deviations to focus on
- Ensure abstract completness of the overall guideword set.

The validity of analysis hinges not only on correctness of the individual analysis step but critically on the completness of the analytical scope with respect to the systems hazardous potentials. This is not a done set that can be applied "brains-off" thought there are systems of guidewords that have gained some level of credibility and in general one should only extend these guidword systems if there is good reason to do so.

Standard guidwords from IEC 61882 are shown in Table 1.

Guide word	Meaning
NO OR NOT	Complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative modification/increase
PART OF	Qualitative modification/decrease
REVERSE	Logical opposite of the design intent
OTHER THAN	Complete substitution

Table 1: IEC 61882 Table 1 Basic guide words and their generic meanings

These guidewods cover functional deviations but do not address some aspects which are crucial in complex systems. Further we decided to drop "reverse" as a guideword as it is essentially a special case of other than and to our understanding does not need a special treatment.

For the exact guildword list and their meaning proposed in HD^3 see the HD^3 process manual [10].

- Basic guidewords: As listed above without reverse
- Temporal: before, after, early, late
- Interaction/concurrency: interrupted, terminated
- Security: Spoof, Elevate Privileges, Listen, Corrupt, Tamper, Denial of Service, Malware, Wrong Connection [11]

The temporal and teraction extensions are crucial to allowing to explore complex systems in general and the security guidewords are necessary to address any threats that could translate into hazards in connected systems.

3.3 Evolution via SACs

HD^3 is an explorative analysis methodology which means it is not making any specific assumptions on what to ask. There is no "catalogue" as in other methods like FMEA/FMEDA. The explorative nature is encoded in guide-words as "input stimulus" and Safety Application Condition (SAC)s as output. A SAC can have two directions - up and down. A SAC that goes up targets an entity out side of the specific system context - typlically the organisation but it could go all the way up to the legal level (e.g. calling for a regulation of ...). A SAC that goes down is a form of extending the initial intent and the ad-hoc decomposition. These SACs are thus design refinements. These can be uncovered functional ommissions (not that often) or safety related constraints (the common case).

The down SACs emerge during the analysis of each guide-word being applied to a specific item. The list of SACs can be quite lengthy, from our current, and still quite limited experience, the technology agnostic analysis will emit multiple SACs per item. These SACs generally cluster around certain issues and by iterating over the guide-words - hopfully - are complete with respect to those issues (e.g. corruption or temporal property violation). These clusters are the the initial requirements to form a new item in the respective derived layers.

Analysis of derived layers then follows which naturally will emit further SACs if needed. This may lead to the creation of further derived items which are though integrated into the derived layer rather than creating a completely new layer. If a deeper layering is needed is not yet known - this is something we will learn from experience only - at present a single derived layer per main analysis layer seems sufficient.

3.4 Technology agnostic analysis

As with all system the key to achieving required system level properties is the design phase (provided the requiremtns phase issound of course). Quite a few safety related properties we strive for are decided not at the technological level but at the technology agnostic level. It often depends on what opetions at the technical level are actually possible given the design statement. If technology aware

and technology agnostic design is not separated then we loose relevant options to eliminate hazards. Further if we do notconciously employ a technology agnostic analysis we also loose the ability to exchange a problematic technology by a more suitable one if the problems/faults are allocated to the implementation and not the technology decision it self.

In complex safety related systems we believe that this is a crucial aspect of systems design. Sequences of use, interdependencies/independence and fault propagation and most notably evolving change of behavior across multiple elements is increasingly becomming the sournce of accidents. As far back as the mid 80s Johnston [**?**] already noted that dependent failures may dominate reliability assessment and that this is only doable based on an in depth system analysis. Likewise Leaveson [5] has shown in numeourse studies that it is not element failures but rather complex interaction and incorrectly selected technologies that play a larger role than element level failures.

Focus:

- System requirements faults, and;
- Incomplete/inconsistent system knowledge, and;
- System design and safety architecture faults, and;
- Incomplete basis for risk aware technology selection.

The technology agnostic analysis roughly covers the early phases as outlined in IEC 61508-1 Ed 2 Clause 7.2-7.6 see 2.3.

3.5 Technology aware unsepecific analysis

Once the intent has been evolved to a consolidated and complete set of functions along with their safety properties these need to be mapped to technologies. This includes allocation to hardware and software (and other technologies) as well as selection of classes of candidates (e.g. operating-system, MCU/CPU, hypervisors, etc.). These technologies have there specific sets of advantages and disadvantages. Explicating these properties is a key issue to allow not only selecting adequate technologies but also to select the best suited candidate. The technology aware unspecific analysis thus allows to eliminate technology related hazards by selecting the cadidate that shows the minimal residual hazards or allows for the best mitigation of the same (e.g. low complexity or "obvious" solutions)

Focus:

- technology suitability isses, and;
- technology requirements faults, and;
- functional allocation, and;
- integrity allocation faults

With the technology agnostic as well as the technology aware unspecific analysis in place a first priliminary system architecture can be defined. This is then the basis (actually the specific context) for the next phase.

3.6 Technology aware sepecific analysis

In the technology aware specific analysis we begin to directly address the needs of pre-existing elements. The analysis focusses on the specific candidate - so we know at this point that we would be using linux-5.4.3/glibc-2.28 on an ARM Cortex A53 quad-core.

This phase is already closly related to the formal requirements of IEC 61508-3 Ed 2 for pre-existing software though we are not claiming conformance simply by applying HD^3 rather the goal is to ensure that the technology aware specific phase extracts the necessary information/knowledge to address the related clasuses of route 3_S. Specifically we focuss on:

> *The software safety requirements specification for the element in its new application shall be documented to the same degree of precision as would be required by this standard for any safety related element of the same systematic capability. The software safety requirements specification shall cover the functional and safety behaviour as applicable to the element in its new application and as specified in 7.2 "Software Safety Requirements Specification".*
>
> [61508-3 Ed 2 7.4.2.13 a)]

The key observation is that this is defining "full contextualization" of the pre-existing element. This calls for functional and safety related **behaviors** to be covered in the safety requirements specification which we interpret as analyzing the pre-existing element in context. This is not calling for conformance to a specification of the pre-existing element, which may or may not exist, it is calling for this specification to be re-contextualized so that the behavior (and potential misbehaviors) of the pre-existing elemnt are fully understood. This is well in line with N Leaveson definition of hazard analysis **??**.

> *The elements design shall be documented to a degree of precision, sufficient to provide evidence of compliance with the requirement specification and the required systematic capability.*
>
> [61508-3 Ed 2 7.4.2.13 c)]

The element design could also be pre-existing, even then though an analysis in context of the specific deployment scenario would be needed as sublcuase c points back to the requirements specification that would have resulted from a). Notably addressing the systematic capabilities of pre-existing elements is only feasible if one also records the criticality of deviations from the specified behaviors that can occure. From this in-context criticality one then can judge the suitability and sufficience of the mitigations (again in the form of SACs).

> *There shall be evidence that all credible failure mechanisms of the software element have been identified and that appropriate mitigation measures have been implemented.*
>
> [61508-3 Ed 2 7.4.2.13 g)]

g) also clearly constraints the analysis to the specific context - credibility makes little sense "out-of-context". While not explicidly mandated we read g) as only satisfiable by an explorative analysis methodology. Much of the issues with pre-exisitng elements at this phase we belive, is going to focus on discrepencies of requirements/design of these elements - which were not anticipated in a safety context most of the time - with the specific needs of the system under analysis and any contribution to hazards that the pre-existing element can develop. The definition of hazard analysis by N. Leaveson:

> *Hazard analysis*
> *Accumulating information on how the behavioral safety constraints, which are derived from the system hazards, can be violated.*
>
> [Leaveson Engineering a Safeer World p212]

asks for the potentials of violating safety constraints - and having elements that were initially specified for a generic and thus generally broader, context, carry the inherent risk of only achieving partial or deviating behiors in corner cases, while fitting well in the generic case. An example of this is the failure behviors of standard c-library functions.

Focus:

- element requirements faults, and;
- function level suitability faults, and;
- unexpected behaviors, and;
- violations of constraints.

3.7 API level analysis

The final analysis knows that the requirements assigned to specific functionality in form of an API set, satisfies the intent if the derived constraints are adhered to. All of the previous analytical efforts did though not consider the specific of the implementation and the constraints that come with this implementation. Notably dependent behavioral aspects - be it fault propagation or implicid fault mitigation. Once we are at the specific API level we can now harvest the implementation specifics to further boild down the hazard potentials. Note that we do assume that CCFs can be identified at this level of detail while being manageable with respect to effort as the focus is narrow enough already.

Note the quite general issu that non-safety related specifications in general only address true positives (desired behavior) and true negatives (repoted and thus controlable failures) but neglect to address false positives and false negatives which are left as an exercise to the safety engineer intending to reuse such a pre-existing element. This limitation also will more often than not apply to existing test-cases and documentation.

Focus:

- local faults, and;
- induced faults, and;
- propagation potentials, and;
- residual unexpected behaviors, and;
- completness and consistency of constraints

3.8 Connecting and tracing

TODO

4 Contextualized Safety Functions

At the lowest level of safety assessment for pre-existing software elements we are operating at the API level. To achieve the demanded functionality we can utilize pre-existing software elements that provide the generic part of the specific function set. Typlically this addresses things like resource allocation, process/thread creation, timer/time handling, low-level I/O, IPC and signals. Additionally some more specific generic functions at a somewhat higher level e.g. string-processing may also be addressed by generic elements. The actuall specific application logic is not generic and will need to be handled by an appropriate fully conformant process (route 1_S - so "sipmly" applying all of IEC 61508 Ed 2 directly).

The suitability of the generic elements depends heavily on the way they were created - from requirements to maintenance. Here we are mostly going to focus on the functional aspects and the possible side-effects they could have in context of the target ssytem.

A more fundamental issue of traditional functional focus is that functional driven development (and its tools) which focuses on true positive/negatives:

- True positives: execution of intended behavior, and;
- True negatives: reporting of expected failures that then are typically handled with defensive structures

We are well equipped for this, sane APIs communicate true positives/negatives via return values or state of modified parameters. Appropriate structures, aka defensive programming, are well entrenched in the everyday programming work-flow. Functional safety standards quite naturally also mandate such defensive structures to be present. Most coding styles also focus on the correctness aspect of false negatives. Correctly detected success and failures also form the most common cases. For most of the function driven development everything else is taken to be negligible or at least tolerable.

Safety related development needs to take into account the two further cases:

- False positives: return indicates success while the announced resources are unavailable or the reported action did not take place.
- False negatives: return indicates failure while the resources or action request has been satisfied

At the formal level this is hinted at in the definition of Type-A systems again. IEC 61508 Ed 2 7.1.4.1.2 c) states *there is sufficient dependable failure data to show that the claimed rates of failure for detected and undetected dangerous failures are met.*. The "undetected dangerous failures" are a subset of the false positives/negatives that a function exhibits.

The consequence of a false positive is self-evident I guess. What might not be that clear is that some false positives are not safety related (e.g. crash-stop failures often are "ok" in fail-safe systems). In other cases they only appear safety related if viewed in isolation. A more subtle effect of contextualization is that many false positives in software show inherent protection by followup actions. With other words many false positives would manifest them selves as true negatives in dependent functions [6] conceptually as shown in Figure 6 As a more formal statement it simply means false positives often

[6] Think of open() silently returning an invalid file-descriptor and the following read trying to access the same - with high probability this read() would fail correctly

times violate the pre-requisites of dependent functions and thus reliably trigger true negatives which in turn are handled [7].

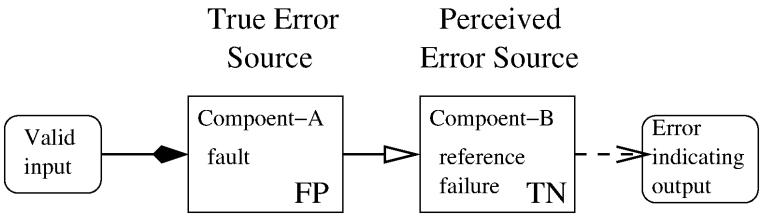

Figure 6: Followup failure protection of fault

From a analytical perspective the main hindrance of utilizing this relation is that it is generally implicit only - if one is conscious of the issue though the available documentation some times does point out the issues. As an example a brief excerpt of the read() man-page (Linux man-pages version 3.74 read 2)

```
DESCRIPTION
    read()  attempts to read up to count bytes from file descriptor
    fd into the buffer starting at buf.


    EINVAL fd is attached to  an object which is unsuitable for
           reading; or the file  was  opened  with the O_DIRECT
   flag, and either the address specified  in  buf, the
   value specified in count, or the current file offset
   is not suitably aligned.
```

Thus some of the potential false positives of open() are caught in read(). Why are they not caught in open() ? This is where context comes in - at the time of open() one only has limited context available - specifically the flags/modes passed to open() - if these are consistent with the actual use is not something that the code path for open() could actually determine. So there are checks - in this case in read() - to address possible inconsistencies of use - with other words violations of assumed context. but these checks are not complete and in general (all cases I know of) they only addressed common human error (i.e. opening a file for reading and then writing to it) but were not targeting corruption or other unexpected behavior. Appropriate analysis - in context - can uncover a number of further covered cases without actually changing any of the pre-existing code.

The consequence of a false negative is generally not so clear cut. In many cases it may seem like "just a false alarm" and entering safe state "unnecessarily" [8]. False negatives can lead to elements not being reset, e.g. file-handlers expected closed staying open or initializations taking place again

[7] Conveniently assuming a single fault hypothesis for the moment

[8] actually it is necessary as the system was in an inappropriate or uncontrolled mode of operation - but we generally respond with reset-and-move-on Bhopal was a tragic cumulation of this approach

since proper shutdown was skipped, leaving the element in an unexpected (unanalyzed and thus potentially hazardous) state.

This transition from function driven view to hazard driven view is one of the key differences in working on safety related rather than functional development. Establishing this will not happen just by a two day IEC/ISO training for staff - this is where safety culture has to take effect and this transition needs to be supported by measures/techniques and processes. HD^3, introduced in [**?**], is not in any way **the** solution - it is hopefully one possible part of an overall solution to hazard aware development.

For pre-existing elements this also defines the main analysis targets. While one can - after adequate review of the DLC - assume some level of coverage of true positive/negatives one generally can not assume any coverage of false positive/negatives in pre-existing elements (even though as noted earlier there may be some implied coverage). Some of the issues can be mitigated at the architectural level (e.g. redundancies) provided the mitigation target is random faults at hardware level or hardware non-determinism which results in software non-determinism (think of race conditions) but the general class of false positives/negatives can not be mitigated by any generic approach. Thus the only avenue we know of that can address these cases is adequate analysis in context of the specific Use-Case. Note though the inherent weakness of this as the prime goal shall be elimination and not mitigation. In the context of pre-existing software this places the onus on adequate processes for selection of these pre-existing elements.

4.1 Addressing false positives/negatives

Where are false positive/negatives mitigated in current software ? In general (non safety related) software they are mostly unhandled. In complex systems it is though not uncommon that they are indirectly protected against by follow up failures of related calls (say call_X() returned an invalid value used in a call_Y() as parameter - if call_Y() manages true negatives properly the call_X() false positive would most likely trigger a failure and appropriately indicate albeit with potentially confusing error reporting). But for general software elements this can not be claimed in any systematic manner given the function driven developed. One reason for this is simply that for non-safety related systems this class of rare events is considered tolerable/negligible.

The goal now for safety related elements is not to plaster software or systems with mitigations for each and every possible false positive/negative but rather to harvest the inherent protection of the system. One of our motivating assumptions we refer to as "capitalizing on complexity" is that complex systems inherently have relatively lengthy dependency chains which makes undetected false positives/negatives increasingly unlikely. This harvesting comes in three basic variants

- Reactive safety mechanisms: itemize
- Explicit mitigation: that could be testing of status at some other level e.g. monitoring of open files at FCU level.
- Implicit mitigation: Mitigation of potential consequences in followup calls

Redundant safety mechanism itemize

Architectural protection: For those false positives/negatives that are not systematic faults but system state related or random HW fault induced failures architectural protection covers these faults.

Umbrella protection mechanisms itemize

Assessment of components contribution potentials

Assessment of components development process maturity

Examples of such inherent potentials are:

1. Implicit follow-up call protection mechanisms - dependent failures
2. Local state expectation checking - known side-effects checking
3. Global state sanity checking - expected global behavior checking
4. Assessment based on element history/properties - extraction of credible faults and appropriate de-rating as well as probability estimates

4.2 Pre-existing elements

For non-safety related pre-existing elements the situation is a bit different. While the formal interfaces based on functional development, specify the true positive/true negative - we do not have the assurance that these specifications are complete and correct respectively that the underlying implementation adhere to the interface. Thus these three issues must be addressed explicitly:

- completeness of specification: all failure modes known, behaviors under failure condition understood == specified.
- correctness of specification: all known failure modes are credible failure modes, behavior under failure condition understandable.
- correctness of implementation: all known failure modes properly handled, behavior under non-failure and failure condition correctly reflected in reported status/data/action.

With adequate assurance based on "assessment of non-compliant development" regarding the above failure modes and success modes - effectively covering true positives and true negatives. We are left with the handling of false positives and false negatives.

False positive/negative analysis can only be in-context. The level of detail to allow maintaining these rare cases is considerable. Specifically it calls for context at different levels of abstraction. The target is not only enumeration but understanding - thats why IEC 61508 Ed2 calls for. Beyond that we also need the ability to quantify or at least qualitatively assess the severities and probabilities involved. There is a considerable difference if a false positive is possible in a safety function or "only" in a error-detection function. The later does not actually induce a hazard - it does immediately elevate the residual risk to a probably intolerable level (or why would the error detection function be there in the first place ?). Thus the severity is critically different. Even more so in complex systems where it is common that a protective or detection function serves not a dedicated hazard scenario but may impact multiple such scenarios (think of a watchdog timer which is the last line of defense for potentially many independent safety functions). In complex systems where dependent failures become frequent this aspect becomes more critical.

The proposed HD^3 contributes to adequate context - allowing to systematically gain sufficient understanding (or detect that the same is **not** established). It also contributes to detecting the false positives and false negatives by guide-word driven analysis. Finally it contributes to detection potential inherent protection mechanisms by guide-words covering the sequential nature (before/after) as well as the asynchronous impact (interrupted/terminate).

4.3 Transition to a probabilistic view of software faults

TODO

4.4 Function level safety - Contextualized Safety Functions

The above development of context and mapping to TP/FP respectively TN/FN serves the purpose to allow qualification of fully contenxtualized functions. The analysis in HD^3 includes elements that can be directly used to address the behavior categorization TP/TN/FP/FN, the analysis though does **not** build on these behavioral categories but rather uses "standard" exploratory guide-words and some extensions to allow complete coverage of the behavioral range in context of the Use-Case.

The TP/TN/FP/FN is then a structuring that is extracted at the API level allowing to document/assess the completeness of the behavioral space that a concrete function provides.

image: GSN for function level completeness argument Constraints - system context

- General Prerequisites:
  - Fully defined Use-Case (Intent and behavioral expectations)
  - Minimized context from Use-Case specific analysis -¿ decomposition
  - Adequate process environment (org, standards and qualification)
  - Adequate traceability (CMS and defined review process)
  - Known dependencies of functional units (via hierarchical decomposition)
    - semi-quantified dependencies of mitigations (at least multiplicity and severity of mitigated hazards).

The pre-requisites are quite obviously exactly those that we anticipate HD^3 to provide - though in principle it may well be possible to use other methods to achieve the necessary contextualization to then assess the function level safety properties of functions.

TP: match expected Use-Case

- Specification valid (consistent and complete)
- Implementation correct (satisfies subset of applicable spec)
- Constraints maximized (minimum scope) and strict subset of spec. (man page, SUSv7, POSIX 1003.1/1003.13)

- Evidence of implementation matching specification available
- Verification: analysis and testing

TN: match engineering expectation and rules for proper handling in place

- Error condition specification valid (consistent and complete)
- Error handling implementation correct (satisfies subset of applicable spec)
- Failure modes documented - behavior under failure condition understood (man page, SUSv7, POSIX 1003.1/1003.13)
- Rules(e.g. coding-style) for handling of TN in place
- Verification: analysis, static code analysis and testing

FP: are managed

- Excluded/minimized up-front (selection/arch-protection)
- Residuals handled *independently*, or; dependencies understood (known CCFs)
- Verification: analysis

FN: are managed

- Excluded/minimized up-front (selection)
- Behavioral impact understood
  - No impact cases (other than false alarm) identified
  - Side-effects identified, understood and handled *independently* (e.g. device opened but not closed -¿ invalid state after reboot)
- Verification: analysis

4.5 Bottom-up vs Top-down

One confusion I probably have caused by now is that of emphasizing the API level analysis. Some might have gotten the impression that one can now simply take the target system API subset and iterate over the true positive/negatives and then add in the false positive/negatives - done - safe API extracted -> build up safe system from "safe API".

The problem here is complexity. APIs are general purpose mostly - and the more general they are the more flexible they need to be. The consequence is that the flexibility, together with a more potent environment, introduces a veritable explosion of the potential false positive/negative possibilities. Thus why one in principle could do an exhaustive analysis at the API level to cover all possible side effects — context free — the enumeration for a single library function would be

mind-boggling. In traditional functionally simple generic elements this did not happen because the context of their development was already quite constraint (often times initially purpose driven and then generalized to a domain or product family only). Not so with pre-existing general purpose elements like a C-library or a full featured OS/RTOS. To re-introduce this constraint now we need the top-down analysis results which then allow to:

- Filter out the credible failure modes - in context.
- Analyze the behavior of the system under fault condition.
- Determine the inherent resilience to specific faults due to the context in which they are to be used.
- Categorize the severity of the deviation from intent.
- Estimate the occurrence rate of such events.

This build-down is the key contribution of top-down hazard driven analysis - and without it we are, I believe, unable to manage complex safety related systems in principle.

So while some might have been getting the impression that there is a convenient bottom-up approach - there unfortunately is not - and never was. We need to combine the top-down analysis results with the bottom-up implementation (from generic APIs to system specific functional and safety behaviors). The top-down analysis is what is iteratively refining the context information which increasingly constraints the potential fault-space and then we can analyze the specific API **in context** and argue inherent resilience, or where needed, amend the capabilities of the selected elements.

The effective life-cylce for a complex system utilizing pre-existing elements for providing basic services/functions (OS, libraries, drivers) thus is a V-model with the top part being bespoke (its a new system so that part can not be reused "as is") and a lower part of the DLC that depends on the origin of the element. In practice this means that the DLC is split into a part handling the pre-existing element and a second part managing the bespoke element. Of course these two parts are not independent of each other but are intimately intertwined.

FIGURE: de-composition of DLC utlizing pre-existing elements.

5 Conclusions

Acronyms

API Application Programming Interface. 3

CAST Causal Analysis based on System Theory. 3

CI Configuration Index. 3, 4

E/E/PE Electric/Electronic/Programmable Electronic. 3

FMEA Failure Mode and Effects Analysis. 3

GSN Goal Structuring Notation. 3

HAZOP Hazard and Operability Study. 17–19

IEC International Electrotechnical Commission (a non-profit, non-governmental international standards organization). 19

NPP Nuclear Power Plant. 19

SAC Safety Application Condition. 20, 22

STPA Sytem Theoretic Process Analysis. 3

References

IEC 61508-1: 2010, Functional safety of electrical/electronic/programmable electronic safety-related systems Part 1: General requirements

IEC 61508-3: 2010, Functional safety of electrical/electronic/programmable electronic safety-related systems Part 3: Software requirements

IEC Guide 104:1997, The preparation of safety publications and the use of basic safety publications and group safety publications

IEC/ISO Guide 51:1999, Safety aspects Guidelines for their inclusion in standards

Nanacy G. Leveson, *Engineering a Safer World*, MIT Press, 2012

Tim Kelly, *Using Software Architecture Techniques to Support the Modular Certification of Safety-Critical Systems*. 11th Australian Workshop on Safety-Related Programmable Systems (SCS'06), 2006

HSE, UK, *HSG 238 - Out of control - Why control systems go wrong and how to prevent failure*, HSE, 2003

Felix Redmill, Morris Chudleigh, James Catmur, *System Safety: HAZOP and Software HAZOP*, 1999

IEC 61882, *Hazard and operability studies (HAZOP studies) Application guide* IEC, 2001

Nicholas Mc Guire, SIL2 Hazard driven decomopsition, design and development CI: HD3-SIL2-OSADL, SIL2LinuxMP OSADL, 2017

Andreas Platschek, *A Harmonized Threat/Hazard Modeling Method for Safety Critical Industrial Systems*, TU Wien, IEC, 2006

SIL2LinuxMP, *Coliminder HD^3 anylsis report* , HL-HAZOP-SIL2-OSADL, OSADL, 2017