Incorporating software failure in risk analysis — Part 2: Risk modeling process and case study

Christoph A. Thieme[1,2*], Ali Mosleh[3,2], Ingrid B. Utne[1,2], and Jeevith Hegde[1]

[1]Norwegian University of Science and Technology (NTNU) Centre for Autonomous Marine Operations and Systems (AMOS), NTNU, Otto Nielsens Veg 10, 7491 Trondheim, Norway; [2] Department of Marine Technology, NTNU, Otto Nielsens Veg 10, 7491 Trondheim, Norway; [3] B. John Garrick Institute for the Risk Sciences, University of California, Los Angeles, 404 Westwood Plaza, Los Angeles CA90095, USA
*Corresponding author E-mail: Christoph.Thieme@ntnu.no

Abstract

The advent of autonomous cars, drones, and ships, the complexity of these systems is increasing, challenging risk analysis and risk mitigation, since the incorporation of software failures intro traditional risk analysis currently is difficult. Current methods that attempt software risk analysis, consider the interaction with hardware and software only superficially. These methods are often inconsistent regarding the level of analysis and cover often only selected software failures.

This paper is a follow-up article of Thieme et al. [1] and presents a process for the analysis of functional software failures, their propagation, and incorporation of the results in traditional risk analysis methods, such as fault trees, and event trees. A functional view on software is taken, that allows for integration of software failure modes into risk analysis of the events and effects, and a common foundation for communication between risk analysts and domain experts. The proposed process can be applied during system development and operation in order to analyses the risk level and identify measures for system improvement. A case study focusing on a decision support system for an autonomous remotely operated vehicle working on a subsea oil and gas production system demonstrates the applicability of the proposed process.

Keywords: Software failure; risk analysis; propagating effects; autonomy

Acronyms

3D	Three Dimensional	FTA	Fault Tree Analysis
AM	Active mode	MOOS	Mission Oriented Operating Suite
AROV	Autonomous Remotely Operated Vehicle	MDb	Mission Oriented Operating Suite Database
DFM	Dynamic Flowgraph Methodology	OEV	Operational Envelope Visualizer
ET	Event Tree	PM	Passive Mode
F	Function (in the case study)	SM	Sporadic mode
FFIP	Functional failure identification and propagation methods	SRS	Software Requirements Specification
FM	Failure Mode (in the case study)	STPA	System-Theoretic Process Analysis

FMEA	Failure Mode and Effects Analysis	UI	User Input
FSPA	Failure propagation and simulation approach	US	User Screen
FT	Fault Tree	XT	Christmas Tree

1 Introduction

Software is part of advanced technological systems. In the future, autonomous vehicles and vessels may be an essential part of the transportation system [2]. Such systems will not be accepted by the public or approved by the authorities, if they are not safe. This means that risk analysis focusing on hardware, software, human and organizational factors, is necessary.

Several challenges arise when attempting to analyze the risk contribution from software and interactions with hardware and humans. These need to be considered to cover the whole spectrum of possible failures [3, 4]. Current methods applied in risk analysis, such as fault trees (FT) and event trees (ET), cannot reflect the interaction of complex software intensive systems sufficiently [5].Software might be reliable in the sense that it is executing the programmed actions correctly. Software behaves deterministically (i.e., software failures will always manifest under the same circumstances). However, the software might act reliably in a situation where the action might be considered unsafe [6].The objective of this article and the accompanying article [1] is to propose a process that may be used to identify hazardous events from software and analyze potential propagating effects on the overall system, including the hardware. The results from the proposed process in this article may be incorporated into a holistic risk analysis during and design and system operation.

Based on the analysis' findings necessary modifications and requirements for the software system can be identified, during the design, development, use, and modification phases in the software life cycle. In addition, it is possible to analyze how the software handles propagating failures caused by other components of the system, such as sensors and human operators. The case study in this article demonstrates the usability of the process. The case study is an example of a software developed through rapid prototyping, to achieve a working solution for demonstration purposes.

This process may be used to support the analysis efforts prescribed in IEC 61508 [7], or the industry specific system safety related standards. Specific parts in some international standards address software development requirements, for example, IEC 61508 [7] in part 3 [8], ISO 26262-6, [9], EN 61511-1, [10], and EN 50128, [11]. These standards highlight the importance of software failure assessment, for the identification of software safety requirements and the software development process. The suggested process can be used to analyze the risk and identify potential software and system improvements during the

development. Similarly, for the development of systems that contain software that is not considered safety related or that are developed to achieve a working solution, the assurance processes in [7, 11, 12] may be too time consuming and resource intensive.

This article builds on the background and results from the accompanying article [1], which provides a taxonomy for functional failure modes of software and the necessary foundations for the process proposed in this article. The process described in this article is qualitative; a quantification of risk is not attempted, this is subject to further work.

A review of the relevant literature for software risk analysis and modeling approaches is presented in Section 2. This is followed by the developed and adapted process in Section 3. Section 4 exemplifies each step of the process. Sections 5 concludes this article.

2 Requirements to a Process Incorporating Software in Risk Analysis

A brief overview of current state-of-the-art methods to incorporate software systems into risk analysis is given in the accompanying article [1]. The review outlines weaknesses in the current methods. The current methods focus either on only specific software failures, e.g., through injection in simulations, or they do consider software only superficially, i.e., not consider software failure mechanisms in detail, or analyze the effect of a software failure on hardware or software.

The term “software system” is used to describe the whole software program with its algorithms and implementation in the hardware. This section presents a proposed set of requirements that were used as a guideline for developing the process presented in this article.

Garrett and Apostolakis [6] identified error forcing contexts, which will lead to software failure. They defined three abilities that risk assessment should have: (i) *represent all those states of the system that are deemed to be hazardous*, (ii) *model the functional and dynamic behavior of the software in terms of transitions between states of the system*, and (iii) *given a system failure event, identify the system states that preceded it*.

Hewett and Seker [13] identified four properties of a risk analysis including software:

1. *Represent structures and (temporal) behaviors of the whole system (together with its interactions with external environments);*
2. *Support the evolution of software*;
3. *Provide modularity and building-block capabilities to cope with scalability issues;*
4. *Offer systematic mechanisms to facilitate automated deduction and inference reasoning for risk analysis*.

Chu et al. [14] collected information from an expert panel on risk analysis of software systems. They agreed that a method incorporating software risk should account for different types of bugs and consider fault tolerant mechanisms and all available information on the software. Dependencies between hardware and software need to be included in the analysis.

In general, risk analysis shall answer three questions: *(i) What can go wrong? (ii) How likely is it that this will happen? (iii) If it does happen, what are the consequences?* [15]. Risk analysis is the *process to comprehend the nature of risk and to determine the level of risk*. These definitions and the considerations above give input to the requirements for the process incorporating software in risk analyses in Table 1.

Table 1 Requirements for a process incorporating software in risk analysis, based on input from [6, 13, 14].

Requirement		Description
R1	Identify failure modes	The process shall enable the analyst to identify failure modes that might lead to unwanted consequences in the context.
R2	Identify possible failure causes	The risk model developed in the process shall assist in the identification of possible failure causes and sources in case of a failure.
R3	Identify consequences of failure modes	The process shall enable the analyst to trace software failure modes through risk scenarios leading to adverse consequences.
R4	Represent functional behavior	The risk model developed in the process shall reflect the functional behavior and constraints of the software including different states and transition between the states.
R5	Represent temporal behavior	The risk model developed in the process shall reflect time-related behavior, requirements, limitations, and states.
R6	Represent context of use	The risk model developed in the process shall include required contextual and overall constraints, hardware, software, and human interactions.
R7	Be modular	The model developed in the process as well as the process shall be modular, such that changes in software modules can be reflected.
R8	Be scalable	The risk model developed in the process shall be scalable, such that different levels of detail can be addressed and that software systems of different sizes can be analyzed.
R9	Make use of all available information	The process shall use all available information to build and analyze the risk model developed in the process.
R10	Be applicable throughout the software life cycle	The process shall be appropriate throughout the lifecycle of the software and aide in decision making.

Requirements R6, R7, and R8 address features that a risk analysis process ~~incorporating software~~incorporating software should exhibit. Requirement R9 refers to the use of information for the process, while R10 shall assure that the process is applicable during the life of the software.

The requirements may be addressed using a functional perspective on the software, which makes it scalable and suitable for failure mode analysis [14, 16]. The discussion, Section 5, uses these requirements to highlight the features of the proposed process in comparison to existing methods and processes.

3 Process for Incorporating Software in Risk Analysis

Figure 1 summarizes the steps in the proposed process in this article and sets it in the context of the generic risk management framework presented by ISO 31000. Steps 2 to 4 are the main novel contributions from this article and the accompanying article [1]. The sections detail each of the steps, as indicated in the figure. Communication between different stakeholders, especially between software engineers and risk analysts, is of utmost importance to apply the proposed process successfully.

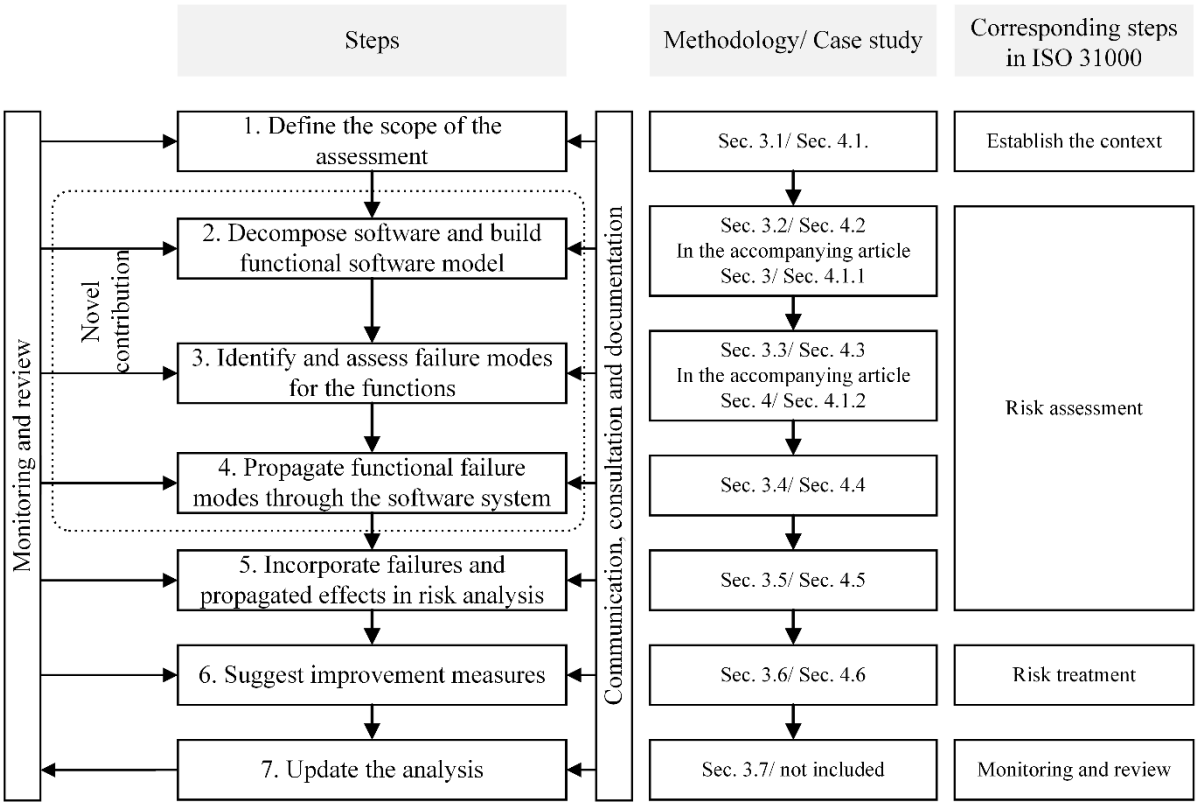

Figure 1 Steps in the proposed process to incorporate software failure in risk analysis and the corresponding steps in the ISO 31000 risk management framework. Abbreviation: Sec. – Section.

Figure 2 shows the proposed process in relation to a generic system life cycle. The outcome of the process is updated and refined through the life cycle phases. The lifecycle is not specific for software or software development, since the process is applicable through the lifetime of the system and useable during the operational phase of the system. Analysis results are fed back in the life cycle phase activities, providing input to the software and system development process. This input may then be used to support the activities outlined in IEC 61508 [7] or the industry specific standards. The analysis is then updated with new information from the engineering process through Step 7. The software development activities in the lifecycle

phases may be following the waterfall model, an agile development, or any other structured development process.

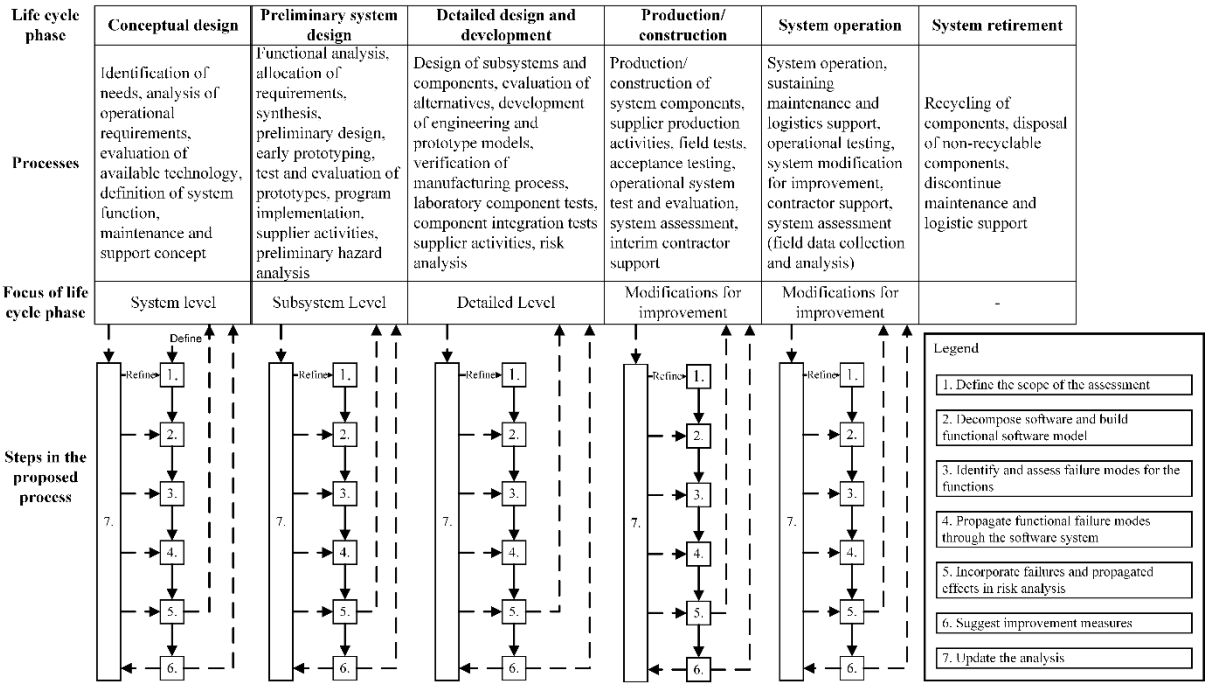

Figure 2 The steps in the proposed process to incorporate software failure in risk analysis in a systems lifecycle. The system life cycle was adapted from [17]. Broken lines with arrows indicate transfer of input.

3.1 Step 1: Define the Scope of the Analysis

The definition of the scope includes an overall description of the software, its purpose, application area, and operational context. Risk analysis is system and context specific, and the analysis should reflect this. The operational context describes which interactions the program has with its environment, such as other software programs, servers, humans, or sensors. Every interaction or output that is different from the expected interaction or output is a failure of the software. Only with the context, it is possible to analyze which failures will cause negative consequences.

The phase of the software in its life cycle determines the level of detail of the risk analysis. The level of detail of the risk analysis needs to be defined. Available documentation for the software, such as the software requirements specification (SRS) (according to IEEE 830 [18]), the system requirements specification, the software development documentation, or the verification and validation documentation, needs to be identified and used in the analysis process. Software engineers should be involved in the process and development of the functional software model to avoid ambiguity and increase understanding of the software system.

3.2 Step 2: Decompose Software and Build Functional Software Model

A functional decomposition of the software system is the first step towards building the functional software model. The functional decomposition and the description of functions is necessary in order to collect and arrange the necessary information for the next steps. The functional analysis standard EN14514 [16] may assist in the decomposition. The accompanying article (Thieme et al. [1]) provides more information on functional decomposition and the description of the functions.

The functional decomposition is used to build the functional software model, which graphically represents the collected information. The functional software model visualizes the interactions between the functions and assists the analysts in maintaining an overview of the functions and their relationships. The connections between the functional elements are constructed according to the information on inputs, outputs, and associated conditions.

Figure 3 summarizes the symbols used for building the model. Two types of connectors are used in the functional software model. *Transfer of information* refers to the connection of functions through common data (i.e., the input and output). The second type, *functional dependency*, describes the influence of functions on other functions that are not related through the exchange of data. This could be functional calls or prerequisite functions. The *software boundary* is used as a visual cue to differentiate the external interfaces from the software functions.

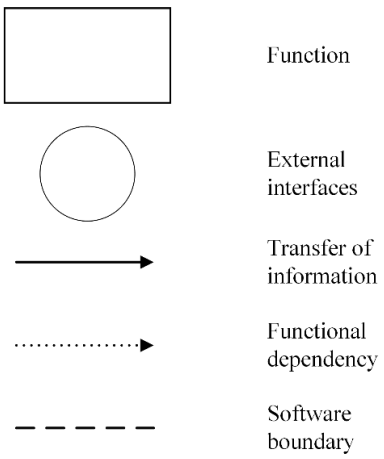

Figure 3 Modeling elements to represent the software functionality.

The information collected in the functional software model and the associated information on the functions assist in the analysis of the interaction failure modes (Step 3) and the analysis of the propagation behavior (Step 4). The description within the blocks needs to be coherent throughout the model to facilitate these steps. An example of a model is shown in the case study in Section 4.

3.3 Step 3: Identify and Assess Failure Modes for the Functions

This step is central to the proposed process since potential failure modes are identified for each function. These function-specific failure modes are propagated in the next step to analyze the effect of each individual failure mode.

The accompanying article [1] presents the failure mode taxonomy used in this present article. In general, a failure mode is the manner in which an item fails [19], and they are context specific [20]. There are four categories in the taxonomy: functional, interaction, timing-related, and value-related failure modes. The failure mode taxonomy in [1] [1] suits the functional view of software adopted in this article.

Functional failure modes are the failure modes that relate to a failure of functionality, e.g., operations are not executed, or extra, unintended functionalities are executed. Interaction failure modes between software functions reflect a failure of transition between software functions, for example, a faulty order of function executions. Timing related failure modes describe the execution of a function at the wrong point of time, with respect to the requirements. The value related failure modes similarly refer to the failure of an output value. Value here may be a numerical value, a character or a symbol.

The analysts need to assess which failure modes are applicable to the software functions. Each identified failure mode needs to have a unique identifier to make it traceable in further analysis. Each failure mode should be described according to the chosen level of analysis. The analysis should consider the complete information to give meaning to the failure modes. Especially functional and non-functional requirements and constraints need to be included in these considerations.

3.4 Step 4: Propagate Functional Failure Modes through the Software System

The output and hence the effect of each failure mode on the external interfaces needs to be analyzed with respect to the overall system functionality and the context. The critical aspect in this step is how the failure modes interact with the external interface through the propagation behavior. The analyst needs to assign an effect in a meaningful manner to the propagated failures. The failure modes are propagated until all reachable interfaces are affected. The importance of considering failure propagation is explained in the accompanying article [1].

Generally, the propagation of the effects resulting from the failure modes highly depends on the software functions and its overall function. The effect of control loops and reiterations within the software should be considered. The propagation should be reiterated at least once for feedback loops, such that the effect of these becomes visible. Faults in the feedback may not be apparent upon their occurrence, since the failure may occur after the feedback is used.

Hence, the influence on the software functions that use the feedback, needs to be assessed. Additional iterations may be necessary. Fault detection and correction mechanisms need to be considered while analyzing the failure propagation behavior.

Table 2 summarizes the propagation behaviors of the failure modes through a software system. The first column summarizes the failure modes. The second column is labeled *refined failure mode*. Refined failure modes describe the failure mode in more detail and reflect a higher level of detail of the analysis. The third column describes the propagation behavior of the failure mode. The column *Ref.* describes the source from which the propagation behavior was derived. In this case, *1* refers to Wei [21] and *2* refers to the authors' identified propagation behavior.

Value-related failures affect subsequent functions by providing an incorrect value. The effect depends on the functionality and the process in the subsequent functions. In most cases, the value failure will lead to an incorrect value failure. Effectively, decisions and output to the external interfaces will be affected by these incorrect values and/or dependent function calls. The propagation and hence the overall effect on the external interfaces ~~is~~are highly dependent on the software purpose.

Functional failure modes mainly propagate similarly to value-related failure modes. Propagation of interaction failure modes depends on the function process and interactions. Not calling or skipping functions will mostly propagate as the failure modes *no value* or *output provided too late*. In most cases, the failure modes related to external files will propagate as the *no value* failure mode.

For timing-related failures modes, three cases are differentiated [21]: no fault tolerance mechanisms with respect to timing (T1), watchdog timers or similar (T2), and failure recovery mechanisms with respect to timing (T3). In the case of T1, these failures will propagate directly through the software functions. In the case of T2, the software will either abort or exhibit a safe behavior. Safe behavior refers to a standard functional call or usage of a safe standard value. Moreover, in the case of a detected failure, T3 refers to software that will execute actions that will reduce the negative effects of the failure mode [21].

If data-rate failures are considered, then the design of the data transfer system becomes relevant [21]. In the sporadic mode (SM), the data receiving function is activated by the available data. Data can be transferred in a passive mode (PM), and the data receiving software functions check all events and data available in the associated buffer. In active mode (AM), the buffer pushes out old data when it is full, and the software function has yet not handled the data. A polling system specifically requests data as soon as the software function requires input [21].

Table 2 Propagation behaviors for each failure mode. Propagation behaviors for timing and value-related failure modes were adapted from Wei [21] (marked with 1 in the Ref. column). Other failure mode propagation mechanisms are based on the authors' assessment (marked with 2). Refined failure modes refer to special cases of failure modes. Reference 3 refers to Wei et al. [22] who in particular analyzed the propagation behavior of timing related failure modes.

Failure mode	Refined failure mode	Propagation behavior	Ref.
Function failure modes			
Omission of a function/missing operation		Propagates as incorrect value failure mode or no value failure mode.	2
Incorrect functionality		Propagates as incorrect value failure mode.	2
Additional functionality		Propagates as incorrect value failure mode (e.g., for outputs that shall not be manipulated). Can also propagate as output provided spuriously failure mode.	2
No voting		Propagates as no value failure mode.	2
Incorrect voting		Propagates as incorrect value failure mode (for the voting result).	2
Failure in failure handling		Detected failure are not handled, and the failure propagates as no value failure mode.	2
Interaction failure modes			
Diverted/incorrect functional call		Failure mode propagates in different ways, depending on the function (i.e., no value, incorrect value, or output provided spuriously).	2
No call of next function		The program stalls, it propagates as failure modes: no value, or output not provided in time.	2
No priority for concurrent functions		Output is propagated with output provided too late failure mode.	2
Incorrect priority for concurrent functions		Output is propagated with output provided too late failure mode.	2
Communication protocol dependent failure modes		These failure modes include the generic failure modes and propagate accordingly.	2
Unexpected interaction with input-output boards		Propagates as output provided spuriously.	2
Failure of interaction with external files or databases	Wrong name	Propagates as no value failure mode.	2
	Invalid name/extension	Propagates as no value failure mode.	2
	File/ database does not exist	Propagates as no value failure mode.	2
	File/ database is open	Writing: propagates as no value failure mode Reading: might not propagate or propagates as no value failure mode.	2
	Wrong/invalid file format	Propagates as no value failure mode.	2
	File head contains error	Propagates as no value failure mode.	2
	File ending contains error	Propagates as no value failure mode.	2
	Wrong file length	Propagates as no value failure mode.	2
	File/database is empty	Propagates like too many elements in data array/structure.	2
	Wrong file/database contents	Propagates as no value failure mode.	2
Timing-related failure modes			

Failure mode	Refined failure mode	Propagation behavior	Ref.
Output provided	Too early	T1[1]: No output is registered, propagates as no value failure mode.	1, 3
		T2[2]: Failure is detected, and the software aborts the operation.	1, 3
		T3[3]: Failure is masked; the premature value is stored in a buffer and available for further operation.	1, 3
	Too late	T1: Fault propagates as delayed output by the same time as the initial delay.	1, 3
		T2: Delay is detected if it is longer than the programmed interval, software aborts operation.	1, 3
		T3: If the delay is longer than the specified interval a standard value/behavior is used that is propagated as incorrect value failure mode.	1, 3
	Spuriously	No output is registered, propagates as *no value* failure mode. Alternatively, a spurious action is triggered that will propagate as *too early* failure mode.	2
	Out of sequence	No output is registered, propagates as *no value* failure mode.	2
	Not in time	See output provided too late, where, for T1, the output is not provided in time.	2
Output rate failure	Too fast	SM[4]: Propagates as *too early* failure mode.	1
		AM[5] (drop new data) or PM[6], within affordable rate: Propagates as *too early* failure mode.	1
		AM (drop new data) or PM, faster than affordable rate: Buffer fills too fast, loss of data propagates as *incorrect value* failure mode. If buffer handles events, these events are lost, and the system does not react accordingly.	1
		AM: push out old data: The output propagates as *incorrect value* failure mode, since the value that is assumed to be read is different from the assumed value.	1
	Too slow	PM: Output rate is the input rate. The *too slow* failure is propagated.	1
		AM: Old values stored in the buffer are used, propagates as *incorrect value* failure mode.	1
		PS: Output rate is the input rate. *Too slow* failure mode is propagated.	1
	Inconsistent	Propagates as *incorrect value* failure mode, pairing values from different times.	2
	Desynchronized	Propagates as *incorrect value* failure mode, taking the value from the synchronization delay.	1
Duration	Too long	Duration of a measurement: Output is propagated as *too high value* failure mode.	1, 3
		Duration of detecting a presence: Signal is recognized multiple times, propagates as *too high failure*.	1, 3
	Too short	Duration of a measurement: Output is propagated as *too low value* failure mode.	1, 3
		Duration of detecting a presence: Signal is not recognized, program does not execute the command, propagates as *output not provided in time* failure mode.	1,3

[1]No failure detection mechanism with respect to timing
[2]Failure detection mechanism,
[3]Failure detection and recovery mechanisms,
[4]Sporadic mode.
[5]Active mode, PM – passive mode
[6]Passive mode

Failure mode	Refined failure mode	Propagation behavior	Ref.
Recurrent functions scheduled incorrectly		Propagates as *output provided spuriously or output not provided in time* failure modes.	2
Value-related failure modes			
No value		Either the next function waits for the value, propagating as *too late* failure mode, or a predefined value is used, propagating as *incorrect value*.	2
Incorrect value	Too high Too low Opposite/inverse value Value is 0 (zero)	The value failure propagates through the software, assuming that the value is correct. The value will lead to wrong computational results and this wrong information will be used during further evaluation. If the computed result falls out of the expected or allowable range, the value will propagate as *out of range* failure mode.	1, 2
Value out of range	Datatype allowable range	Value is adjusted to fit in the range and will propagate as *incorrect value* failure mode.	1
	Application allowable range	Value is adjusted to the closest allowable value of the range and propagates as *incorrect value* failure mode with this value.	1
Redundant/frozen value		Value propagates with the value as incorrect value.	2
Noisy value/precision error		Depending on magnitude, will lead to an *incorrect value* failure mode and propagate as such.	2
Value with wrong datatype		Depending on the type of the conversion, different propagation mechanisms were identified. The failure mode might not influence the value and be masked, leading to a loss of precision or incorrect value failure modes. If failure detection mechanisms can detect the failure, the operation will be aborted, and the software will continue as specified. For a detailed list of datatype errors, see Wei [21].	1
Non-numerical value	Not a number (NaN)	Corresponds to an undefined value conversion; hence, it will propagate according to the propagation mechanisms for value with wrong datatype.	2
	Infinite Negative infinite	Will propagate as *incorrect value out of range* failure mode.	2
Elements in a data array/structure	Too many	Elements come from different components. Error not propagated, additional input neglected.	1
		Elements come from one component, are read in fixed format, and are added to the end. Error not propagated, additional input neglected.	1
		Elements come from one component, are read in fixed format, and are inserted in the data array/structure. *Incorrect value* failure mode propagation from the element of insertion.	1
		Elements come from one component, are read in unfixed format, and are added at the end of the data array/structure. *Incorrect value* failure mode propagation of the last element.	1
	Too few	Elements come from one component, are read in unfixed format, and are inserted the data array/structure. *Incorrect value* failure mode propagation of the remaining elements.	1
		Elements come from different components. Propagates as *too late* failure mode.	1
		Elements come from one component. Propagation as *no value* failure mode.	1
	Data in wrong order	For the elements that are wrongly ordered the failure mode will propagate as *incorrect value* failure mode. Value with wrong datatype failure modes might also be relevant.	2
	Data in reversed order	The failure mode will propagate as *incorrect value* failure mode, with the correct reversed values. Value with wrong datatype failure modes might also be relevant.	2

Failure mode	Refined failure mode	Propagation behavior	Ref.
	Enumerated value incorrect	If the value lies within the range of the array/structure, it will be propagated as *incorrect value* failure mode. If falls out of the range it will lead to a program crash or will be handled by the failure detection mechanism.	2
Correct value is validated as incorrect		Correct values are rejected. Propagated as *too late*, or *output not provided in time* (c.f. timing failure modes).	2
Incorrect value is validated as correct		Incorrect value is propagated as *incorrect value* failure mode.	2
Data is not validated		Propagated as *too late*, or *output not provided in time* (c.f. timing failure modes or software aborts).	2

3.5 Step 5: Incorporate Failures and Propagated Effects in Risk Analysis

This step incorporates the propagating effects that were identified in Step 4. These effects may be implemented, for example in FTs, or ETs.

Steps 4 and 5 are closely connected. Some iterations may be necessary to identify the relevant effects on the external interfaces that need to be incorporated in the risk analysis.

The software failure mode effect on the external interfaces needs to be viewed in the context of use with other technical sub-systems and/or operator actions [23, 24]. Human operator actions may lead to software failures, but they may also recover the system from software failure.

In addition to failures in the software, failures might arise in the interfaces of the software [3]. This might be faulty measurements from sensors, incorrectly entered data from human operators, or incorrectly implemented database queries. Applying the failure mode propagation behavior may be used to analyze the effect of an interface failure on the software system and consequently on the other external interfaces. This is not discussed further here and is subject to further work.

Quantification could be derived through expert judgment or software reliability models. However, the quantification of the identified failure modes and the propagated failure effects on the external interfaces is out of scope of this article and will not be discussed further.

3.6 Step 6: Suggest Improvement Measures

Risk analysis is used to determine the risk level of an activity and propose mitigating measures in case of high levels of risk. Measures to improve the software system are (among others) to specify additional software functionality, redesign the software system, or specify additional safety and functional requirements for the software system. Risk analysis may also reveal the need for changes to hardware and to external interfaces with the human operator (supervisor).

In general, in risk analysis should be used in the design phase of systems, so that necessary changes can be specified and implemented in an early phase of development. The same applies to the process proposed in this article; software failures need to be included along with hardware failures and human error as early as possible in the system development phase.

The process may be applied to existing technological systems to estimate and include the risk contribution from the software system to the overall risk level. In contrast to hardware systems, software failure modes that are successfully removed from the software system will not reoccur under the same circumstances. Software updates that address identified failure modes and effects on the external interfaces need to be tested and verified.

The software system should be tested, validated, and verified before it is used in operation to demonstrate compliance with the requirements. The results from the presented process to incorporate software in risk analysis could be used to generate test cases to ensure that critical failure modes will not occur. A formal software development process as laid out in ISO/IEC/IEEE 12207:2008 [25] and ISO/IEC/IEEE 15288:2015 [26] may assist in improving the software.

3.7 Step 7: Update the Analysis

In accordance with the risk management standard ISO31000 [27], risk analyses need to be updated regularly. The identification of failure modes and the associated risk analysis might make it necessary to update the functional software model. There might be changes in the context of use, change of interfaces, implementation of new functions, or implementation of failure identification and correction mechanisms.

3.8 Discussion

One important aspect for incorporating software in risk analysis of the proposed process is the propagation of identified functional software failure modes to identify their effects on external interfaces. The propagation behavior was partly adopted from the literature [21] and extended. Wei [21] defined propagation behavior for less failure modes than the accompanying article [1] covers. Therefore, this present article defines the propagation behavior for the failure modes from [1] that have not been covered previously.

The propagation behavior allows for a consistent analysis of the software behavior if a functional software failure mode occurs. The purpose of the proposed process is to highlight possible weaknesses in the software and hardware system as a basis for improving the SRS, system specification and focus testing and verification efforts on critical aspects of the software system. This implies that a software project in an early phase should consider all failure modes and therefore will be aware of possible failure modes and associated propagated effects on the external interfaces.

Table 3 assesses the proposed process to incorporate software in risk analysis against the requirements that are presented in Table 1. All requirements are fulfilled except R5 and R7. Since the process is considering timing-related failure modes, R5 is only partly fulfilled. However, only through incorporation of the process in dynamic risk analysis is it truly possible to capture the full implications of timing-related failure modes in risk analysis [28].

Requirement 7, which is not fulfilled, addresses the quantification of the likelihood of software failure modes and their associated effects on the external interfaces. This is subject to further work. A software tool may facilitate the process of analyzing the effect of propagating failure modes, their integration, and quantification in risk analysis.

Table 3 Assessment of the proposed process to incorporate software in risk analysis against the criteria from Table 1.

Requirement		Fulfillment	Comment
R1	Identify failure modes	Yes	Individual functional failure modes are identified for each function. The first part of this article identifies a comprehensive and coherent set of functional software failure modes.
R2	Identify possible failure causes	Yes	Failure causes can be found in the interfaces in the software itself or failure in the hardware support. The accompanying article outlines possible failure causes [1].
R3	Identify consequences of failure modes	Yes	Through consequent application of the failure mode propagation behavior, the consequences of software failure modes can be identified. The effects on the external interfaces can be integrated into risk analyses.
R4	Represent functional behavior	Yes	The functional behavior of the software system is explicitly modeled and represented through the functions.
R5	Represent temporal behavior	Partly	The temporal behavior is included in the model through timing constraints, requirements, and timing-related failure modes.
R6	Represent context of use	Yes	The context of use of the software is represented by including external interfaces in the functional software model, considering the overall requirements, and using context-specific failure modes for a certain situation.
R7	Be modular	Yes	The functional software model is modular through the functional decomposition. Each function is represented as its own module.
R8	Be scalable	Yes	The process for incorporating software in risk analysis is scalable. It can be used for large and small software systems. The interactions between the functions are known and hence can be modeled. The process can focus on different levels of detail and functional decomposition.
R9	Make use of all available information	Yes	The functional software model uses and reflects all the information that is collected in the SRS and other documentation.
R10	Be applicable throughout software life cycle	Yes	Through the scalability and modularity, the process can be applied at different phases of development. Especially in the operation phase, the modularity makes it easy to adapt the model to changes.

The proposed process in this article allows for identification of functional failure modes, failure consequences (through the propagation), and failure causes, which addresses R1 to R3. The process allows for representing the context and functional behavior (R4 and R6). Failure modes are identified for the functional behavior. The effects of the functional failure modes may be integrated in risk analysis, thus integrating it in the context.

The proposed process is modular and scalable (R7 and R8), which originates from the functional approach. The functional approach also allows using the proposed process in different life cycle phases (R10). The process makes use of all available information, building the functional software model and assessing failure modes based on that information.

Generally, the proposed process requires a good understanding of the software to be analyzed and the software development process. It is necessary that the risk analyst and software

developers work together and develop a common understanding of both the software and risk analysis, such that ambiguities can be avoided.

The presented process is not the first to attempt to identify and incorporate software failures into risk analysis. Wei et al. [29] applied failure modes and identified their effects in a simulation environment. Wei et al. [29] only applied selected failure modes to some of the software functions. Their approach requires that the full software is available. However, not all the information that might be available from the software development process is incorporated. Hence, the approach by Wei et al. [29] does not completely fulfill the requirements R7, R9, and R10.

The presented process in this article differs significantly from a software failure mode and effect analysis (FMEA). In most cases, FMEA assesses the effect of a failure mode based on discussion and knowledge of the analysts, and not all available information is used (R9). Moreover, FMEA is most suitable for risk analysis in the design phases of a system (R10, [19]), more detailed analyses may require more sophisticated risk analysis techniques. The method is based on simple checklists and focuses only on components or subsystems. The bottom up analysis is does not allow for assessment of interactions.

The suggested process for incorporating software in risk analysis focuses on the software and its interactions with external interfaces and implementation of relevant failure events in risk analysis. This is different from other methods and processes, such as system-theoretic process analysis (STPA, [30-32]) or the dynamic flowgraph method (DFM, [33-35]), which focus on the identification of hazardous events. In addition, DFM does not provide mechanisms for identifying failure causes (R2).

Li et al. [20] and Li [36] take as similar approach to the analysis of functional failure as in this article. Selected identified failure modes in the software are directly implemented in FT and ET, or ESD. No failure propagation is conducted (R3), which may exclude some consequences from the considerations. Although timing failure modes are considered, the analysis is rather static with only FT and ESD (R5).

The functional failure identification and propagation methods (FFIP, [37-39]) and the failure propagation and simulation approach (FPSA, [39]) are developed to assess the system behavior in case of one or multiple faults. Software and hardware interaction are the focus. Several models, such as state space models and function flow models are used to assess the propagation. The FPSA module allows for simulation of the model to identify time dependent relationships and delayed failure effects. The model fulfills all the requirements outlined above. However, failures need to be identified by the analysts and failure propagation behavior needs to be defined specifically for each function.

4 Case Study

This section exemplifies the process to incorporate the impact of software failures in risk analysis on a software-based decision support tool with risk relevant implications. Each step of the proposed process will be addressed, except Step 7. A complete analysis of the software system would be too extensive for this article. Hence, only selected aspects of the case study object will be presented in detail. The case study is deliberately kept simple with respect to decomposition to make the process better traceable and understandable for the reader. Steps 2 and 3 are briefly presented in the case study of the accompanying article [1].

4.1 Step 1: Define the Scope of the Analysis

Hegde et al. [40, 41] presented collision avoidance rules based on envelopes for an autonomous remotely operated vehicle (AROV). They implemented the set of traffic rules in a software tool to provide decision support in AROV operations, the underwater operational envelope visualizer (OEV). Since the software provides decision support with respect to the safe operation of the AROV, it is desirable that the tool does not increase the level of risk. AROV may collide with the underwater infrastructure, the seabed and other underwater vehicles [42]. A collision may lead to loss of the AROV, damages to the subsea structure and damages to the environment caused through the damages to the subsea structure.

The aim of the underwater OEV software is to increase the situation awareness of the human operator to detect collisions and avoid collision with subsea obstacles [42]. The control system of the AROV and the physical components of the AROV are not part of the analysis, except for the qualitative fault tree analysis in Section 4.5.

Ph.D. candidates at the Norwegian University of Science and Technology developed the underwater OEV for demonstration purposes in the research project *Next Generation Inspection, Maintenance and Repair operations*. The developers adopted a rapid prototyping approach, where the software was specified, written, tested and reiterated several times. This presented process is applied at a later iteration, to identify additional improvement measures for the software. The main developer is a co-author of this article and provided necessary information and input for the analysis. The process described in this and the accompanying article was applied to use a structured process to identify risk relevant software failure modes and consequently improve the software during the next iteration. It is to be noted that the approach presented in this article is programming language independent. It is developed in an academic setting, and therefore not following the lifecycle processes of software, as laid out , for example, in ISO/IEC/IEEE 12207:2017 [25] or ISO/IEC/IEEE 15288:2015 [26].

4.1.1 Context for the analysis

The underwater OEV by Hegde et al. [40, 41] has four aims: visualize the detection of static obstacles using safety envelopes, suggest a change of course based on safe traffic rules if an obstacle is detected, provide three-dimensional (3D) orientation and position visualization, and visualize the traversed path in time and space.

The underwater OEV is designed for the operation of AROVs, which are unmanned underwater vehicles that operate mostly autonomously. The aim of the underwater OEV is to aid the human supervisor during different underwater intervention scenarios. Unlike traditional collision avoidance system, which are reactive safety system, the underwater OEV can be categorized as a decision support system. The OEV follows two main assumptions: (i) the size and position of all detected obstacles are known and (ii) the exact position of the AROV is known. To ensure easy interface with different software modules within the project, the underwater OEV was developed in Python 2.7 programming language. The required functions were identified during the early development phase. A rapid prototyping and testing approach were used to create the first working version of the underwater OEV, which was later continuously improved. The user interface was created with Qt creator and was converted to python code. The renderer of the 3D model uses the Visualization Toolkit library. The plots are realized with the Matplotlib library.

AROVs will be required to approach subsea production systems to inspect, maintain or repair it during subsea inspection and repair operations. For the analysis, a transit of the AROV from a subsea garage to a working site in an underwater oil and gas production facility is assumed. The AROV moves with velocity of 1.5 m/s. The distance from the center of the AROV to the outer envelope is 2.5 m. During the transit, the AROV passes another subsea structure.

Figure 4 summarizes the system architecture and the mission test setup. In the bottom half of the figure, the AROV flies close to a mock Christmas tree (XT, a subsea valve array). The position of the mock XT is predetermined. A collision detection algorithm utilizes the position of the obstacle and the AROV and detects the collisions areas if the AROV is on collision course with the XT. The position of the AROV is obtained by the *Qualisys* motion sensors and forwarded to the Mission Oriented Operating Suite (MOOS) middle-ware.

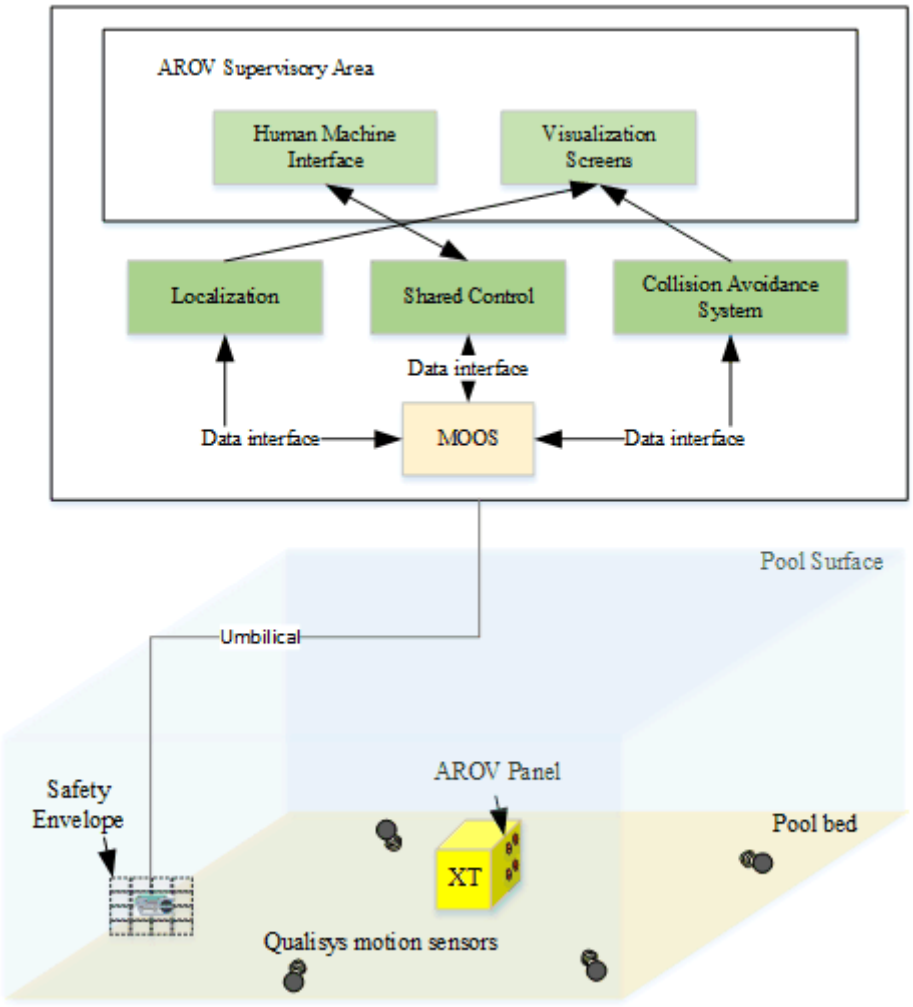

Figure 4 Test set up of the underwater OEV showing the AROV with the environment on the bottom half of the illustration and the system architecture on the top half of the illustration. Adapted from [41]. Abbreviations. XT - Christmas Tree

The upper part of Figure 4 shows the system architecture. The underwater OEV is one of the three modules accessible to AROV supervisors through visualization screens. The underwater OEV receives its input from an external interface. The MOOS database provides position, attitude, and collision data. In addition, MOOS is a middleware developed to access the mission-related parameters [43]. The MOOS database collects, and stores data produced by the AROV and associated software. The data can be requested from the AROV components that need parts of the data. The underwater OEV produces outputs. It sends requests to the MOOS database for position, orientation, and identified collision candidates, and it visualizes the 3D model, position plots, and status messages regarding recommended actions to the human operator via a screen.

Figure 5 shows the user interface of the underwater OEV. The operator and the AROV can utilize the information from the collision detection algorithm to identify the area sensitive to collisions given the position of the known obstacle. The green blocks in Figure 5 signify safe

areas and red blocks signify unsafe areas/ collision candidates. Depending on the collision candidates, the user/ AROV is suggested as an appropriate collision avoidance maneuver. The user also has access to the real-time orientation of the underwater vehicle through the visualization. For the analyzed scenario, shown in Figure 5, the expected recommendation of the underwater OEV is to execute an evasive maneuver to the left of the structure to keep a safe distance from the obstacle. The human operator could also take direct control of the AROV using the control joysticks. Although the underwater OEV is a conceptual development, it is assumed that it is part of the human-machine interface of the human operator with the AROV and hence can assist in the operation.

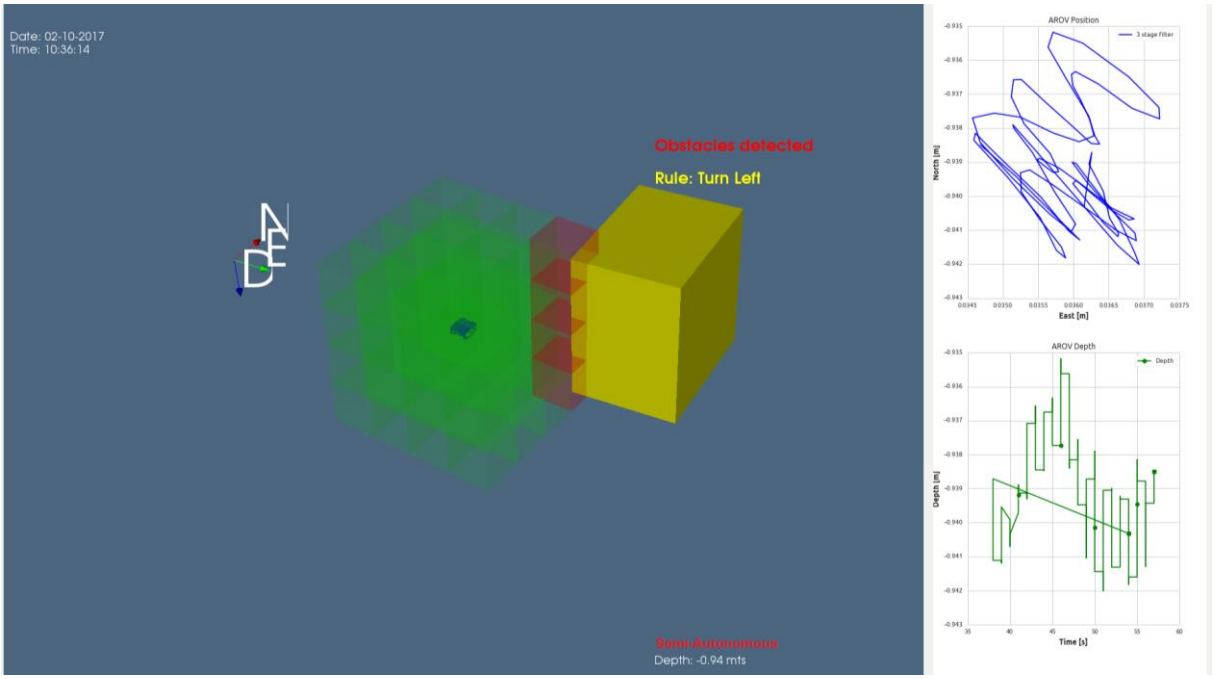

Figure 5 Situation visualization for the case study; the plots on the right hand side are a visual example, not representing the current situation. This is what the operator will see during an operation.

The implementation of the safety envelopes in the MOOS database and the AROV control has been verified and demonstrated [40, 41]. The traffic rules are assumed to be implemented correctly in the underwater OEV. It has been verified that the MOOS database gives expected datatypes and outputs in the right format.

4.1.2 Aim of the risk analysis

The analysis focuses on how the underwater OEV could contribute to a collision with the subsea structure that the AROV shall pass. Based on the above-described situation, the possible effects of the software on the external interfaces are analyzed with the failure modes

and the propagation behavior. The results of the analysis shall be implemented in qualitative FTs to analyze the effect on the overall operation.

The application of the process shall give input to potential mitigation measures and shall help to improve the software during the next update. Other mitigating measures may be to adapt the system architecture. It shall also identify additional requirements or functionalities, which are necessary to avoid or mitigate the effect of possible failures.

4.2 Step 2: Decompose Software and Build Functional Software Model

The software decomposition can be found in the accompanying article [1]. Five functions were identified: initialize underwater OEV (F1), obtain data (F2), determine suggested action (F3), prepare render information (F4), and display information (F5).

In the first function, *initialize underwater OEV*, the program starts, establishes a connection to the database, and sets up the window for visualizing the data. In F2, *obtain data*, the software polls for the necessary information that the underwater OEV uses in the subsequent functions. The underwater OEV shall poll data from the MOOS database with a frequency of 2 Hz. The function is detailed in the accompanying article [1] and shall further serve as an example for the process in this article.

In F3, *determine suggested action*, information on the collision candidates and their positions is used to determine which actions are necessary to avoid a collision and stay at a safe distance. In F4, *prepare render information*, this information and the information on the collision candidates is used to highlight the corresponding safety envelope elements and display the recommendation. In addition, the 3D model is rendered according to the orientation of the AROV to give the human operator an overview of the situation.

The last function, *display information*, updates the plots for the position and the 3D model. This information is sent to the user screen, where the human operator will see the information and use it as aid for operating and monitoring the AROV.

Figure 6 presents the functional software model for the underwater OEV. It was developed from the functional decomposition and the description of the functions. All identified interfaces have been included. The program execution loop is represented through the broken line from F5 to F2. The diagram supports the analyses of failure modes and failure effect propagation in the next two steps. It illustrates the connection of the functions, the flow of information, and the dependency of functions. Each line is labeled with the associated output. These are described in Table 4. They represent the information that was summarized above.

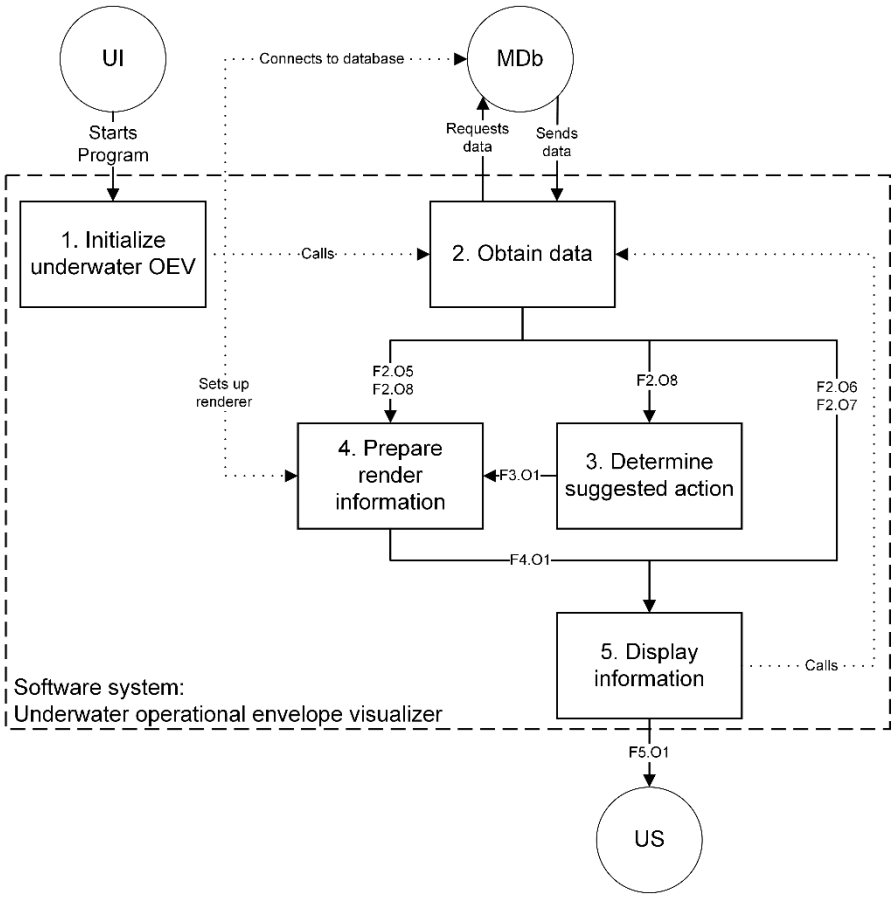

Figure 6 Functional software model of the underwater operational envelope visualizer. Abbreviations: UI – User input; MDb – MOOS Database; US - User screen, description of the outputs can be found in Table 7.

Table 4 Description of the outputs of the functions of the underwater OEV, found in Figure 7.

Abbreviation	Name	Description
F2. O5	AROV orientation	Vehicle orientation in roll, pitch, and heading of the AROV.
F2.O6	AROV operational mode	Mode of operation of the AROV (i.e., remote control, semi-autonomous, autonomous).
F2.O7	AROV position	Local position of the AROV with respect to a local reference coordinate system, described in the north, east, and down reference frame.
F2.O8	Information on identified collision candidates	Information on objects that were identified as falling within the safety envelopes of the underwater OEV.
F3.O1	Suggested action	Suggested action to the AROV operator based on the current context.
F4.O1	Render information	Information necessary to update the renderer.
F5.O1	Screen information	Visualized information containing the render model, suggested action and position plot.

4.3 Step 3: Identify and Assess Failure Modes for the Functions

As mentioned in the previous section, F2, *obtain data*, is used as the case study object. The accompanying article [1] identified 36 failure modes for that function. The set of identified failure modes is incomplete. It focuses on demonstrating how most of the generic failure modes can

be applied to the software function. The identified failure modes can be found in the first two columns of Table 5 in the next section. The failure mode identification will not be explained further here.

4.4 Step 4: Propagate Functional Failure Modes through the Software System

Table 5 summarizes the effects of applying the failure mode propagation behavior to the identified functional failure modes of F2. For the propagation of the failure modes, the information collected in Figure 6 is used. Information on the affected functions can be read directly from the functional software model (Figure 6).

In general, functions that are assessed with *no effect* do not propagated further. *No information updated or displayed* in the column *effect on user screen* can be interpreted as a crash or a hanging of the underwater OEV. The human operator will not receive any information. Some selected examples shall clarify the analysis process and provide additionally needed information in the following paragraphs.

The failure mode FM4, *incorrect functionality of storing values in the corresponding variables*, *making them unavailable,* will result in *no output* to the subsequent functions. These will not be able to produce their required output due to the missing data. Therefore, the user screen will not be updated, or any information displayed.

In FM8, *no function call to F3*, the software execution is affected in such a way that F4 will be executed directly. That means that the render information is prepared and sent further to function F5. In this case, F5 will prepare the display data without the suggested action since it was not determined. Hence, the user screen will show all information correctly, except the suggested action.

With respect to timing-related failure modes, two examples will be further explained. *Output provided too late (500 ms): request for AROV orientation* (FM14), which is a delay in the execution of the functions that succeed F2, occurs. The program will periodically run the functions in the specified order. The human operator will experience the delay since the screen is not updated in real time but with the delay of 500 ms.

Table 5 Propagation behavior applied to the failure modes identified for Function 2 obtain data. The outputs are described in Table 3.

Failure modes		Propagation through F3	Propagation through F4	Propagation through F5	Effect on user screen (the external interface)
Function failure modes					
FM1	*Omission* of "Obtain data", which is not executed.	No F3.O1	No F4.O1	No F5.O1	No information displayed or updated
FM2	*Omission* of requesting data, which means that data is not requested.	No F3.O1	No F4.O1	No F5.O1	No information displayed or updated
FM3	*Omission* of converting MDb.O1 to AROV orientation data, which means that the orientation is note executed.	No effect	F4.O1 with wrong orientation	F5.O1 prepared with wrong orientation	Rendered model displayed with wrong orientation
FM4	Incorrect functionality of storing values in the corresponding variables, making them unavailable	No F3.O1	No F4.O1	No F5.O1	No information displayed or updated
FM5	Additional functionality while converting AROV orientation (e.g., conversion of AROV position)	No effect	No effect	F5.O1 prepared with wrong position	Plots displayed with wrong position
FM6	Failure in failure handling, no detection that no value has been received	No F3.O1	No F4.O1	No F5.O1	No information displayed or updated
Interaction failure modes					
FM7	Incorrect function call, calling F4, skipping F3	F3 not executed	Updated without F3.O1	F5.O1 prepared without suggested action	Information displayed without suggested action
FM8	No function call to F3	F3 not executed	Updated without F3.O1	F5.O1 prepared without suggested action	Information displayed without suggested action
FM9	Incorrect priority for functions, call function F4 before F3	Executed after the renderer is updated	Updated with F3.O1 with old information	F5.O1 prepared with old suggested action	Information displayed with old suggested action
FM10	Unable to request information from the database (communication protocol-dependent failure)	No F3.O1	No F4.O1	No F5.O1	No information displayed or updated
FM11	Request with wrong variable name to the database for AROV position	No effect	No effect	F5.O1 prepared without position information	No update of AROV position plot
Timing-related failure modes					
FM12	Output provided too early: Request for AROV orientation	Output provided too early	Output provided too early	F5.O1 provided too early	Information on screen is updated earlier than required
FM13	Output provided too late: Request for AROV orientation	Output provided too late	Output provided too late	F5.O1 provided too late	Information on screen is updated later than required
FM14	Output provided too late (500 ms): request for AROV orientation	Output provided 500 ms late	Output provided 500 ms late	F5.O1 provided 500 ms late	Information on screen is updated 500 ms later than required

Failure modes		Propagation through F3	Propagation through F4	Propagation through F5	Effect on user screen (the external interface)
FM15	Output provided spuriously: AROV operational mode	No effect	No effect	No effect	Information on screen is incorrectly updated
FM16	Output provided out of sequence: Information on identified collision candidates provided before AROV position	No effect	No effect	No effect	Information on screen is incorrectly updated earlier than required
FM17	Output not provided in time: Information on identified collision candidates	Output provided too late (delay determined by delay in F2)	Output provided too late (delay determined by delay in F2)	F5.O1 provided too late (delay determined by delay in F2)	Information on screen is updated later than required (delay determined by delay in F2)
FM18	Output rate too fast: Requests to database send too fast (within affordable rate)	Output provided too early	Output provided too early	F5.O1 provided too early	Information on screen is updated earlier than required
	Output rate too fast: Requests to database send too fast (out of affordable rate, data dropped)	No F3.O1	No F4.O1	No F5.O1	No information displayed or updated
FM19	Output rate too slow: Requests to database send too slow	Output provided too late	Output provided too late	F5.O1 provided too late	Information on screen is updated later than required
FM20	Inconsistent rate for requests	Output provided inconsistently in time	Output provided inconsistently in time	F5.O1 provided inconsistently in time	Information on screen is updated inconsistently
Value-related failure modes					
FM21	No value for AROV position	No effect	No effect	F5.O1 is not containing an position update	No update of AROV position plot
FM22	Incorrect value for AROV position (not further defined)	No effect	No effect	F5.O1 is not containing the right position	Plots displayed with wrong position
FM23	Incorrect value, too high for AROV operational mode = 2	No effect	No effect	F5.O1 contains wrong operational mode “autonomous”	Display of information that AROV is in autonomous mode
FM24	Incorrect value, too low for AROV operational mode = 0	No effect	No effect	F5.O1contains wrong operational mode “manual”	Display of information that AROV is in manual mode
FM25	Incorrect value, too high, AROV orientation [0, 0, -15]	No effect	Render model prepared with wrong heading orientation (- 15°)	F5.O1 prepared with - 15° wrong heading	Render model displayed with -15° wrong heading
FM26	Incorrect value, too high, AROV orientation [0, 0, -30]	No effect	Render model prepared with wrong heading orientation (- 30°)	F5.O1 prepared with - 30° wrong heading	Render model displayed with -30° wrong heading

Failure modes		Propagation through F3	Propagation through F4	Propagation through F5	Effect on user screen (the external interface)
FM27	Incorrect value, zero for AROV position [0,0,0]	No effect	No effect	F5.O1 prepared with position displayed as origin of the local coordinate system	AROV position displayed as origin of the local coordinate system
FM28	Value out of application allowable range for Information on identified collision candidates includes the value 68	No F3.O1	No F4.O1	No F5.O1	No information displayed or updated
FM29	Value out of datatype range for AROV operational mode = 2,147,483,648	No effect	No effect	Operational mode adjusted to closest allowable value (0), F5.O1 prepared with wrong operational mode	Display of information that AROV is in manual mode
FM30	Frozen value for Information on identified collision candidates (no collision candidates detected)	F3.O1 = no suggested action	F4.O1 without highlighted envelope elements	F5.O1 prepared without suggested action and without highlighted envelope elements	Display of information that no action is needed, and no collision candidates were detected
FM31	Imprecise value for AROV position varying more than 1 m	No effect	No effect	F5.O1 prepared with imprecise position (+/- 1m)	AROV position imprecisely displayed (+/- 1m)
FM32	Wrong datatype for AROV operational mode, string instead of int	No effect	No effect	No effect, if the value is 1	No effect
FM33	Too many (65) elements, in information on identified collision candidates	No effect, if the value is within the range	No effect	No effect	No effect
FM34	Too few elements (two elements instead of three) in AROV orientation	No effect	No F4.O1	No F5.O1	No information displayed or updated
FM35	Data in wrong order in AROV position [z, x, y] instead of [x, y, z]	No effect	No effect	F5.O1 prepared with wrong position	Plots displayed with wrong position
FM36	Incorrect value (no value) for F2.O5-F2.O8 is validated as correct and is output	No F3.O1	No F4.O1	No F5.O1	No information displayed or updated

Abbreviation: FM – Failure mode, MDb – MOOS database, Functions: F2 – Obtain data, F3 – Determine suggested action, F4 – Prepare render information, F5 – Display information

In FM16, *output provided out of sequence: information on identified collision candidates provided before AROV position,* there is *no effect* on the output. The information is stored in dedicated variables. Unless the information is stored to the wrong variables, it will not affect the output to the external interfaces.

The failure modes FM23 and FM24, *incorrect value, too high, AROV orientation [0, 0, -15]/ [0, 0, -30]*, respectively, are a special demonstration of how similar failure modes might affect the risk level. In this case, the heading of the vehicle is shifted in the failure mode by -15° and -30°, respectively. This failure will affect the model of the AROV being displayed with a wrong heading. Incorrect orientation display might have different implications for the human operator.

Regarding FM28, *value out of application allowable range for information on identified collision candidates includes the value 68*, the failure mode will propagate as no output. The output will lead to no output in F3 since the value cannot be interpreted. No mechanisms are in place to check whether the value falls in the range. The *no output* failure mode will propagate to the screen, and the human operator will experience it as a hanging or crashing of the program.

Similarly, FM 34, *too few elements, (two elements instead of three), in AROV orientation*, will lead to no output in Function 4. Function 3 is not affected since it does not use the information in the output *AROV orientation*. In Function 4, the program will read from the array, which only has two elements and not the expected three elements. When trying to read the third element, the function will not be able to do so and cannot produce an output. The human operator again will experience this as hanging or crash.

4.5 Step 5: Incorporate Failures and Propagated Effects in Risk Analysis

This section demonstrates how the identified effects on the external interfaces and the safety-relevant effects can be implemented in the risk analysis. For that purpose, a fault tree analysis (FTA) was conducted. The top event for the FT is *collision with subsea structure during transit*. It incorporates human- and software-related events. The developed FT covers only part of the complete risk analysis.

Figure 7 and Figure 8 present the developed FT, which is split into two parts for better readability. The effects on the interfaces from the propagated failure modes that relate to the display of wrong information are presented in Figure 7.

The effects on the interfaces from the propagated failure modes that relate to the omission of displaying information can be found in Figure 8. Examples are *no information displayed or updated* or *no update of AROV position plot*. These events are only relevant if the human operator needs to rely heavily on the underwater OEV, due to visibility or technical conditions, and if the human operator decides to continue the mission, despite the degraded performance

of the underwater EOV. Two events in the FTs are undeveloped, these relate to the failure in the control system and human operator failure during waypoint planning or implementation.

The main part of the FT, Figure 7, includes some of the events that relate to a wrong display of information or delayed output of information. The AND-Gate 3, for example, contains events in which the information is provided too late with respect to the requirements. However, it might be possible that the human operator can act beforehand or that the human operator can react and avoid a collision. Effects of propagated functional software failure modes that were included are *information on screen is updated 500 ms later than required*, *information on screen is updated later than required,* and *information on screen is updated inconsistently*.

Another group of effects of propagated functional software failure modes are those that relate to wrong information being displayed, such as position in AND-Gate 4, heading in AND-Gate 2, and *AROV operational mode being displayed as autonomous operation* in AND-Gate 5. Most of the events that will lead to a collision require the human operator to be fully trusting the information provided by the software, while not using other available information.

Not all of the identified effects of propagated functional software failure modes are relevant for the context. Hence, they were not included in the FTs. For example, *information on screen is updated earlier than required* does not influence the risk in relation to a collision. On the contrary, the earlier information is available and updated (an increased update frequency is implied) the better it is for the human operator.

Similarly, *display of information that AROV is in manual mode* was not included since the human operator will act, in this case. This is disregarding the possibility that the human operator will not act due to other reasons. Such an event could be potentially found in the undeveloped event *operator failure during waypoint planning or implementation*.

The event *render model displayed with -15° wrong heading* was not included in the FT, since it is a rather limited change of heading and it falls in the normal variation of the AROV heading (e.g., to compensate for external disturbances). A deviation by more than that, in this case -30°, is assumed significant, such that the human operator will act, in this case, one that may lead to a collision.

The minimal cut sets should be identified from the fault trees to reveal the most critical events. Risk reduction and mitigation efforts should be prioritizing the most critical events from the minimal cut set analysis.

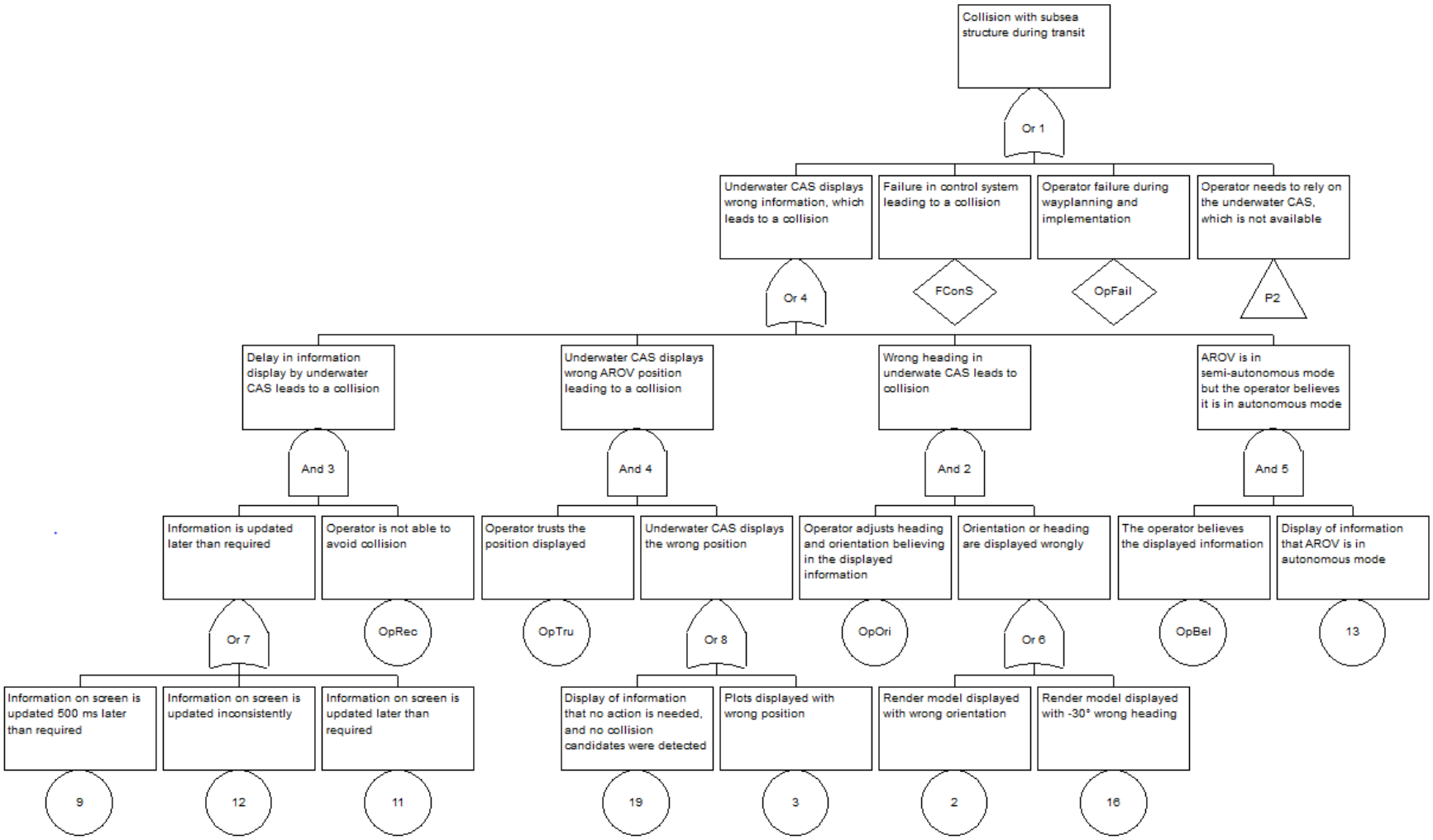

Figure 7 Main fault tree with the top event *collision with subsea structure during transit*. The fault tree was developed with the effects from the propagated functional software modes.

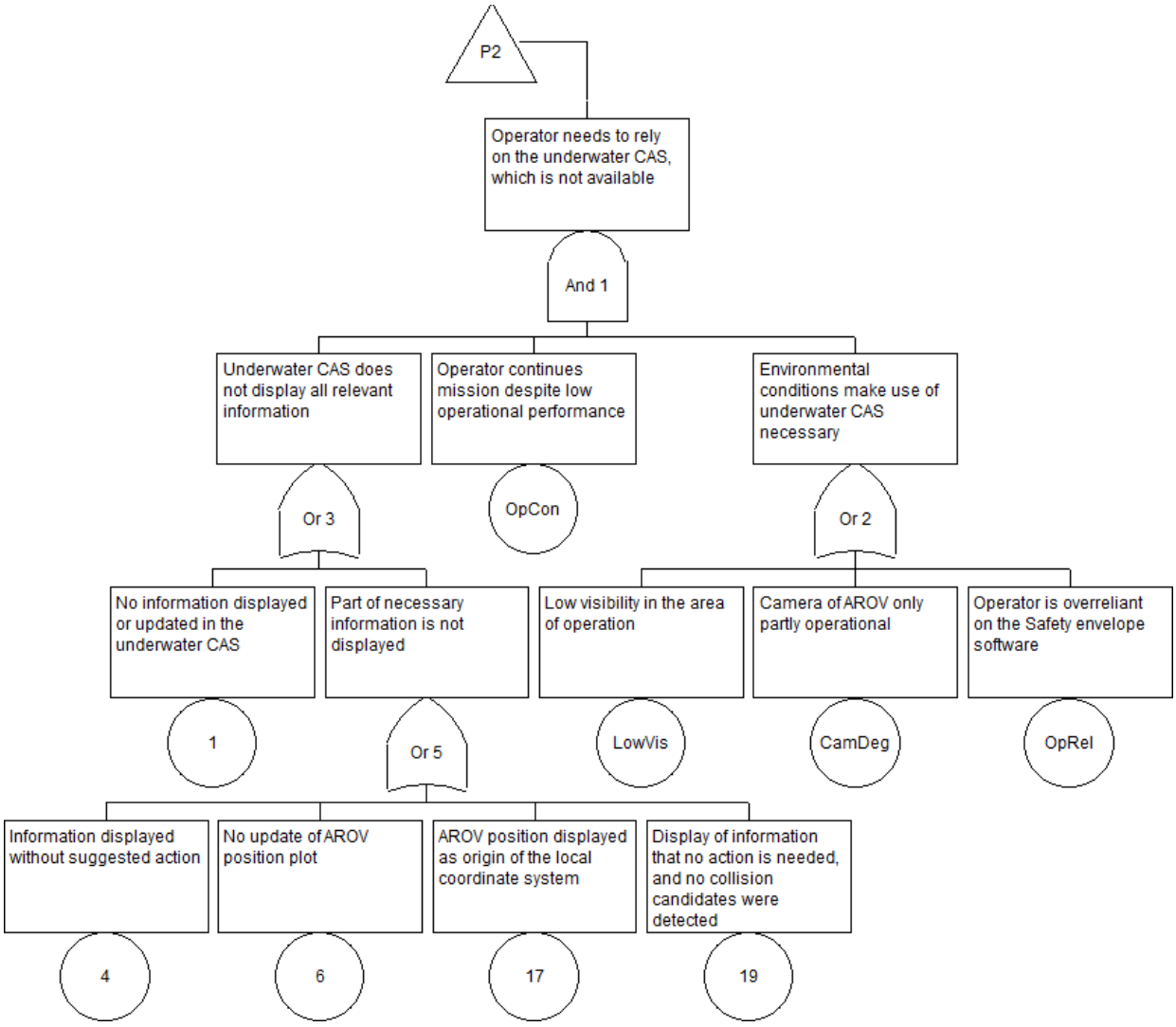

Figure 8 Sub-fault tree for the transfer gate P2 of the fault tree *collision with subsea structure during transit*.

4.6 Step 6: Suggest Improvement Measures

Most of the failure modes and their propagation effects on the interfaces of the underwater OEV that were identified could be prevented by verifying that the data received is in the correct format and expected datatype.

The most critical software failures are those that lead to a crash or hanging of the software. Most of these may be avoided. In the current version of the program, no timing watchdogs or similar mechanisms are implemented to ensure that the software will abort after a time without output. By defining such requirements and accordingly implementing them, hanging of the program can be detected and prevented. Similarly, no mechanisms for checking the validity and of inputs from the MOOS database or outputs within the software.

In general, the underwater OEV was missing an implemented failure message system to the human operators. This should be implemented to assist the human operators in failure detection and solutions. Since the case study only covers a limited set of failure modes, no more improvement measures will be discussed.

4.7 Discussion

The case study was chosen due to its relevance for operation of an AROV and the potential for software improvement. Almost all the failure modes can be applied to the case study; hence, it is well suited for demonstration. Function 2 of the underwater OEV is described in detail. The analysis of other functions of the underwater OEV can be carried out similarly. The identification and propagation of software failure modes has been demonstrated. Only a few timing requirements are defined; therefore, only a few aspects of the timing-related failure modes could be demonstrated.

Results from the case study show that software functional requirements and fault detection mechanisms can be identified to improve the software. This is addressed in the case study in Section 4.6. Analyzing the other functions and Function 2 completely could potentially identify additional relevant effects on the external interfaces, which should be implemented in the risk analysis. Consequently, this will lead to more specific recommendations for improvement of the software.

The example demonstrates that the effects of propagated failure modes on the external interfaces can be implemented in a risk analysis, in this case an FTA. The presented FTA uses a simplified FT, neglecting failures that might arise independently of the analyzed software. In a full risk analysis, these events may need to be considered. For example, the control system of the AROV should be analyzed with the proposed process.

Some challenges are associated with the application of the proposed process to the underwater OEV. The software is developed in an academic setting, which does not apply a formal development process, as it may be used in the industry. However, it is believed that the example is representative for safety-relevant and safety-related software systems and the risk analysis of these. The analyzed software is an important support system for the operation of AROV and might be implemented in future human-machine interfaces for AROV.

The programming language chosen in the case study, Python, may be seen critical. It is not a recommended programming language for safety related systems [44]. However, the underwater OEV does not perform a safety related function as defined in IEC 61508-4 [45]. The underwater OEV is a supporting tool for visualization of the systems state that may lead to accidents that may

result in severe losses and environmental consequences. In addition, the software was developed through a rapid prototyping approach to achieve a working solution. Following the assurance processes in, e.g., IEC 61508 [7], would not have been viable. Hence, this example shows also that a non-safety-related function may contribute to risk and needs to be incorporated in risk analyses.

The proposed process is cumbersome; thus, only one of the five functions of the underwater OEV was analyzed. Analyzing more complex software systems will be time-consuming. However, it will benefit the software being analyzed by deriving a comprehensive list of functional failure modes and their associated effects on external interfaces. Hence, an automated software tool should be developed and used to aid in the process.

5 Conclusion

This article presents a process for incorporating software failures in risk analysis, analyzing the effects of the propagated failure modes on external interfaces, and incorporating these into the risk analysis. The process provides a systematic way to analyze the effects of ~~f~~ailure modes on the software output and associated external interfaces. The identified effects can be implemented in risk analysis and incorporated with human operator, sensor, and computing hardware–related failure events. This is an advantage over the current methods for incorporating software in risk analyses since a structured process is applied that may produce replicable, traceable and understandable results.

The proposed process may be used in the development phase of the software. It may aid in highlighting necessary measures to improve the software and make it safe before the software code is finalized and released. The process may be applied to systems that are not per definition safety related but that may have implications for the level of risk. In addition, systems that are developed through rapid prototyping approaches may benefit from the proposed process, since the rapid changes can be easily captured and implemented in the model. The process may be applied to existing software systems, which makes it possible to improve existing software systems through updates and changes in operation.

Ten requirements were developed to assess the process for incorporating software in a risk analysis. The proposed process fulfills these requirements, except for two. The proposed process does not fully capture the dynamics of the software with respect to the context; a dynamic risk analysis is required. The proposed process does not provide an approach to quantify the likelihood of the identified effects of propagated functional software failure modes on the external interfaces.

Relevant software failure effects are context specific and can be implemented directly in a risk analysis, via methods, such as FTs and ETs. The case study in this article shows how such a venture could be conducted. It is believed that the proposed process can assist in identifying a cohesive set of software failure effects on other safety related software systems, hardware systems and/ or human users through its external interfaces. Therefore, it is possible to improve the safety performance of the overall system.

In the future, the process should be applied to more complex technical systems, such as autonomous ships, to demonstrate its applicability and feasibility. In addition to the expected increase in the complexity of each software system, such an analysis will require the analysis of different software systems that interact and communicate with each other, e.g., a navigation system and a traffic monitoring system.

The dependencies between different software systems may be analyzed with the proposed process, for example a software failure effect may propagate through another software system. Future work should also incorporate the software failure effects on the external interfaces with human and organizational factors and the complete hardware system. Including these propagation behaviors will improve the incorporation of mutual dependencies between software users, hardware, and software.

Further work includes the investigation of failure causes for the failure. Two main causes may lead to a software failure; undetected faults in the code or failures of sensor or the computing hardware. Further investigation is needed to identify a suitable quantification method. In addition, a software tool, facilitating the proposed process in this article, should be developed, to save the analyst time.

Acknowledgements

The authors want to thank three anonymous reviewers, who helped to improve this article through their valuable comments.

C. A. Thieme and I. B. Utne appreciate the support of the Research Council of Norway through the Centres of Excellence funding scheme, Project number 223254 – NTNU AMOS.

J. Hegde, has been supported by the Research Council of Norway, Statoil and TechnipFMC through the research project Next Generation Subsea Inspection, Maintenance and Repair Operations, 234108/E30 and associated with AMOS 223254.

C. A. Thieme acknowledges the financial support from Norges tekniske høgskoles fond, who supported this research with a scholarship for a research exchange with the B. John Garrick Institute for the Risk Sciences at the University of California in Los Angeles.

References

[1] Thieme CA, Mosleh A, Utne IB, Hegde J. Incorporating Software Failure in Risk Analysis – Part 1: Software Functional Failure Mode Classification. Submitted for review to Reliability Engineering and System Safety. submitted: pp. 1-29.

[2] Marr B. The Future of the Transport Industry - Iot, Big Data, Ai and Autonomous Vehicles. 2017; https://www.forbes.com/sites/bernardmarr/2017/11/06/the-future-of-the-transport-industry-iot-big-data-ai-and-autonomous-vehicles/#2b854d791137; Accessed: 21.02.2018

[3] Ozarin NW. The Role of Software Failure Modes and Effects Analysis for Interfaces in Safety- and Mission-Critical Systems. IEEE International Systems Conference Proceedings, SysCon 2008. Montreal, QC, Canada: IEEE; 2008. pp. 200-207.

[4] Ozarin NW. Applying Software Failure Modes and Effects Analysis to Interfaces. Annual Reliability and Maintainability Symposium 2009. pp. 533-538.

[5] Mosleh A. Pra: A Perspective on Strengths, Current Limitations, and Possible Improvements. Nuclear Engineering and Technology. 2014;46: pp. 1-10.

[6] Garrett CJ, Apostolakis G. Context in the Risk Assessment of Digital Systems. Risk Analysis. 1999;19: pp. 23-32.

[7] IEC. Iec 61508: Functional Safety of Electrical/Electronic/ Programmable Electronic Safety Related Systems. Geneva, Switzerland: International Electrotechnical Commission; 2010.

[8] IEC. Iec 61508-3: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. Part 3: Software requirements. Geneva, Switzerland: International Electrotechnical Comittee; 2010.

[9] ISO. Iso 26262-6: Road Vehicles-Functional Safety. Part 6: Product development at the software level. Geneva, Switzerland: International Organization for Standardization; 2018.

[10] EN. En 61511-1: Functional Safety - Safety Instrumented Systems for the Process Industry Sector. Part 1: Framework definitions, system, hardware and application programming requirements. Brussels, Belgium: European Committee for Electrotechnical Standardization; 2017.

[11] EN. En 50128: Railway Applications - Communication, Signalling, and Processing Systems. Software for railway control and protection systems. Brussels, Belgium: European Committee for Electrotechnical Standardization; 2011.

[12] ISO. Iso 26262: Road Vehicles - Functional Safety. Geneva, Switzerland: International Organization for Standardization; 2018.

[13] Hewett R, Seker R. A Risk Assessment Model of Embedded Software Systems. 2005 29th Annual IEEE/NASA Software Engineering Workshop, SEW'05. Greenbelt, MD, USA: Institute of Electrical and Electronics Engineers Computer Society; 2005. pp. 142-149.

[14] Chu T-L, Martinez-Guridi G, Yue M, Samanta P, Vinod G, Lehner J. Workshop on Philosophical Basis for Incorporating Software Failures in a Probabilistic Risk Assessment. Digital System Software PRA. Brookhaven National Laboratory; 2009. pp. 1-1-2-21.

[15] Kaplan S, Garrick BJ. On the Quantitative Definition of Risk. Risk Analysis. 1981;1: pp. 11-27.

[16] EN. Ns-En14514: Space Engineering Standards - Functional Analysis. Brussels, Belgium: European Committee for Standardization; 2004.

[17] Blanchard BS. System Engineering Management. 4th Ed. ed. Hoboken, N.J: Wiley; 2008.

[18] IEEE. Ieee 830: Recommended Practice for Software Requirements Specification. New York, NY, USA: Institute of Electrical and Electronics Engineers; 2009.

[19] IEC EN. En Iec 60812: Analysis Techniques for System Reliability – Procedure for Failure Mode and Effects Analysis (Fmea). Brussels, Belgium: International Electrotechnical Commission, European Committee for Electrotechnical Standardization; 2018.

[20] Li B, Li M, Ghose S, Smidts C. Integrating Software into Pra. Issre 2003: 14th International Symposium on Software Reliability Engineering, Proceedings. 2003. pp. 457-467.

[21] Wei YY. A Study of Software Input Failure Propagation Mechanisms. College Park, MD: University of Maryland; 2006.

[22] Wei Y, Rodriguez M, Smidts C. How Time-Related Failures Affect the Software System. In: Stamatelatos MG, Blackman HS, editors. Proceedings of the Eighth International Conference on Probabilistic Safety Assessment and Management (Psam). New York, NY: ASME; 2006.

[23] Ozarin NW. Bridging Software and Hardware Fmea in Complex Systems. 2013 Proceedings Annual Reliability and Maintainability Symposium (RAMS). 2013. pp. 1-6.

[24] Lindholm C, Notander JP, Höst M. A Case Study on Software Risk Analysis and Planning in Medical Device Development. Software Quality Journal. 2014;22: pp. 469-497.
[25] ISO/IEC/IEEE. Iso/Iec/Ieee12207: Systems and Software Engineering - Software Life Cycle Processes. Geneva, CH; New York, NY, USA: International Organization for Standardization , International Electrotechnical Commission, Institute of Electrical and Electronics Engineers; 2017. pp. 1-164.
[26] ISO/IEC/IEEE. Iso/Iec/Ieee 15288: Systems and Software Engineering - System Life Cycle Processes. Geneva, Switzerland: International Organization for Standardization , International Electrotechnical Commission, Institute of Electrical and Electronics Engineers; 2015. pp. 1-118.
[27] ISO. Iso 31000 Risk Management - Principles and Guidelines. Geneva, Switzerland: International Organization for Standardization 2018.
[28] Zhu D, Mosleh A, Smidts C. A Framework to Integrate Software Behavior into Dynamic Probabilistic Risk Assessment. Reliability Engineering & System Safety. 2007;92: pp. 1733-1755.
[29] Wei YY, Rodriguez M, Smidts CS. Probabilistic Risk Assessment Framework for Software Propagation Analysis of Failures. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability. 2010;224: pp. 113-135.
[30] Leveson NG, Fleming CH, Spencer M, Thomas J, Wilkinson C. Safety Assessment of Complex, Software-Intensive Systems. SAE International Journal of Aerospace. 2012;5: pp. 233-244.
[31] Abdulkhaleq A, Wagner S, Leveson N. A Comprehensive Safety Engineering Approach for Software-Intensive Systems Based on Stpa. Procedia Engineering. 2015. pp. 2-11.
[32] Abdulkhaleq A, Wagner S. Integrated Safety Analysis Using Systems-Theoretic Process Analysis and Software Model Checking. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2015. pp. 121-134.
[33] Garrett CJ, Guarro SB, Apostolakis GE. The Dynamic Flowgraph Methodology for Assessing the Dependability of Embedded Software Systems. IEEE Transactions on Systems, Man, and Cybernetics. 1995;25: pp. 824-840.
[34] Guarro SB, Yau MK, Dixon S. Context-Based Software Risk Modeling: A Recommended Approach for Assessment of Software Related Risk in Nasa Missions. 11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference, PSAM11, ESREL2012. 2012. pp. 1839-1848.
[35] Guarro SB, Yau MK, Ozguner U, Aldemir T, Kurt A, Hejase M, et al. Formal Framework and Models for Validation and Verification of Software-Intensive Aerospace Systems. AIAA Information Systems-Infotech At Aerospace Conference. Grapevine, TX, USA: American Institute of Aeronautics and Astronautics Inc, AIAA; 2017.
[36] Li B. Integrating Software into Pra (Probabilistic Risk Assessment) [Monograph]. College Park, Md: University of Maryland; 2004.
[37] Jensen D, Tumer IY, Kurtoglu T. Modeling the Propagation of Failures in Software Driven Hardware Systems to

Enable Risk-Informed Design. 2008 ASME International Mechanical Engineering Congress and Exposition. Boston, Massachusetts, USA: ASME; 2008.
[38] Tumer I, Smidts C. Integrated Design-Stage Failure Analysis of Software-Driven Hardware Systems. IEEE Transactions on Computers. 2011;60: pp. 1072-1084.
[39] Mutha C, Jensen D, Tumer I, Smidts C. An Integrated Multidomain Functional Failure and Propagation Analysis Approach for Safe System Design. Artificial Intelligence for Engineering Design, Analysis and Manufacturing. 2013;27: pp. 317-347.
[40] Hegde J. Tools and Methods to Manage Risk in Autonomous Subsea Inspection, Maintenance and Repair Operations. Trondheim, Norway: Norwegian University of Science and Technology (NTNU); 2018.
[41] Hegde J, Henriksen EH, Utne IB, Schjølberg I. Development of Safety Envelopes and Subsea Traffic Rules for Autonomous Remotely Operated Vehicles. Journal of Loss Prevention in the Process Industries. 2019: pp.
[42] Hegde J, Utne IB, Schjølberg I. Development of Collision Risk Indicators for Autonomous Subsea Inspection Maintenance and Repair. Journal of Loss Prevention in the Process Industries. 2016;44: pp. 440-452.
[43] Newman PM. Moos-Mission Orientated Operating Suite. Massachusetts: Department of Ocean Engineering, MIT; 2008. pp. 1-53.

[44] IEC. Iec 61508-7: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. Part 7: Overview of techniques and measures. Geneva, Switzerland: International Electrotechnical Comittee; 2010.
[45] IEC. Iec 61508-4: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. Part 4: Definitions and Abbreviations. Geneva, Switzerland: International Electrotechnical Comittee; 2010.