Incorporating software failure in risk analysis – Part 1: Software functional failure mode classification

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, CA 90095, USA

*Corresponding author E-mail: Christoph.Thieme@ntnu.no

Abstract

Advanced technological systems consist of a combination of hardware and software, and they are often operated or supervised by a human operator. Failures in software-intensive systems may be difficult to identify, analyze, and mitigate, owing to system complexity, system interactions, and cascading effects. Risk analysis of such systems is necessary to ensure safe operation.

The traditional approach to risk analysis focuses on hardware failures and, to some extent, on human and organizational factors. Software failures are often overlooked, or it is assumed that the system's software does not fail. Research and industry efforts are directed toward software reliability and safety. However, the effect of software failures on the level of risk of advanced technological systems has so far received little attention. Most analytical methods focus on selected software failures and tend to be inconsistent with respect to the level of analysis.

There is a need for risk analysis methods that are able to sufficiently take hardware, software, and human and organizational risk factors into account. Hence, this article presents a foundation that enables software failure to be included in the general framework of risk analysis. This article is the first of two articles addressing the challenges of analyzing software failures and including their potential risk contribution to a system or operation. Hence, the focus is on risks resulting from software failures, and not on software reliability, because risk and reliability are two different aspects of a system.

Using a functional perspective on software, this article distinguishes between failure mode, failure cause, and failure effects. Accordingly, 29 failure modes are identified to form a taxonomy and are demonstrated in a case study. The taxonomy assists in identifying software failure modes, which provide input to the risk analysis of software-intensive systems, presented in a subsequent article (Part 2 of 2) [1].

Keywords: Software risk; functional failure mode; hazard identification; hazard taxonomy; risk analysis

Acronyms

AROV	Autonomous remotely operated vehicle	FT	Fault tree
CCF	Common-cause failure	NASA	National Aeronautics and Space Administration
CSRM	Context-based software risk management	OECD	Organization for Economic Cooperation and Development
DFM	Dynamic-flowgraph methodology	OEV	Operational-envelope visualizer
ESD	Event-sequence diagram	SEooC	Safety element out of context
FMEA	Failure mode and effects analysis	STPA	Systems-theoretic process analysis
FPSA	Failure propagation and simulation approach		

1 Introduction

Risk analysis provides decision support in the design and operation of technological systems. One important step of risk analysis is the identification of hazardous events; however, this may be challenging for advanced systems such as autonomous vehicles and vessels. Advanced systems consist of both hardware and software, and the analysis of software failures is difficult if traditional methods of risk analysis are used [2].

For example, autonomous vehicles and vessels may become an essential part of future transportation systems [3]. Autonomous cars are presently being tested. In the maritime industry, autonomous ships are expected to operate within the next five years [4, 5]; however, concerns are being raised with respect to the safety of autonomous ships and their control systems [6]. The risk contribution from software is often treated superficially and is assumed to be safe and reliable such as in maritime transportation [7].

For the regulatory authorities and the public to approve and accept the widespread use of autonomous systems, the level of risk associated with such systems needs to be sufficiently low. Hence, the risks associated with these systems need to be analyzed for both the hardware and the software. Risk analysis attempts to answer three questions: "(i) what can go wrong, (ii) how likely is it that it will happen, and (iii) if it does happen, what are the consequences?" [8]. A risk analysis is the process that answers these questions.

The overall purpose of this article (Part 1) and the subsequent article (Part 2) [1] is to provide a process for analyzing software and hardware to incorporate into existing risk-analysis methods. Part 1 helps identify hazards and develops an associated taxonomy. Part 2, the next step of risk analysis, proposes a process for analyzing the effects of software failures on integrated and interacting software, on hardware components, or on operators of the system.

Thus, we first describe the identification of potentially hazardous events or possible failure modes, particularly of software. "A failure mode is the manner in which an item fails" [9] and is specific to the context of operation [10]. A generated numerical output that is legitimate and correct in general may be wrong and inappropriate for a specific situation (the context of operation). Some experts agree that software that is used under normal operational conditions can be analyzed from a functional point of view [11] with respect to risk, but a different process for risk analysis is necessary because software failures are not like hardware failures. They may lead to unanticipated effects that are not easily identified [12]. Software might be reliable in the sense that it executes programmed actions correctly. However, the software might act reliably in a situation where the action might be considered unsafe [13]. Software behaves deterministically (i.e., software failures will always manifest under the same circumstances). Failures of external interfaces or the computing hardware may result in cascading failures affecting the whole system. This should be considered to cover the whole risk spectrum for a system.

Some risk-analysis approaches have been introduced to cover the risk contribution of software. Examples are software failure mode and effects analysis (FMEA) [14-20]), dynamic-flowgraph methodology (DFM) [21-24], or simulations with failure mode injection [25-27]). The current methods make only limited use of available information and the context of use (i.e., the interaction with other software, hardware, or human operators). The mechanisms of how a failure in the software develops and affects the system are not treated in detail.

Existing failure mode taxonomies do not adhere consistently to one level of analysis (i.e., the software-system level, functional level, or code level). This makes analysis and decision making for mitigating measures for software risks difficult. Hence, a generic and consistent set of failure modes may improve the identification process with respect to coherence and the reduced time needed for such analyses.

The objective of this article (Part 1 of 2) is to identify and structure generic functional-software failure modes into one consistent and comprehensive taxonomy that can be further used in the risk analysis of advanced systems and operations. The generic failure modes can be combined with specific functions of a software system. The purpose is to provide a comprehensive and systematic basis for the risk analysis of software-intensive systems such as autonomous vehicles and ships. Failure modes could be used to support the efforts necessary to fulfill the requirements for safety-related systems, such as the industry standards in IEC 61508 [28], ISO 26262 [29], or EN 50128 [30].

The subsequent article (Part 2 of 2) [1] describes the process for incorporating and analyzing the effect of software failure modes in risk analysis, based on the taxonomy proposed herein. Part 2 relies on the functional representation of software. The process described in Part 2 [1] can be applied from an early conceptual stage and be refined with system development. It may be used to support analysis requirements per industry standards for safety-related systems and the development of software that is not considered part of a safety system. Therefore, it could aid in the identification of safety requirements and assist in preparing verification and validation plans. The topic of risk acceptance related to (software) system failure is not covered, as that depends highly on the system and its application.

The next section offers the necessary background on the methods of risk analysis for software systems and relevant software failure mode taxonomies. Section 3 defines and describes the concepts of software functions and the functional decomposition of software. Failure mode taxonomy application to a case study is presented in Section 4. The last section discusses and concludes the work.

2 Background

2.1 Related Standards

Software has been recognized as a key element of safety-related systems. Hence, international standards for the development of safety-related systems address software in specific parts: for example, the software requirements set forth in the generic standard for safety-related systems (IEC 61508-3 [31]). Specific industry standards for safety-related systems also address software requirements for road vehicles (ISO 26262-6 [32]), the process industry (EN 61511 [33]), and railway applications (EN 50128 [30]). Not all software is related to safety, but a software failure may result in cascading failures that may affect the level of risk.

Safety of the intended function is a concept that refers to risk resulting from functional insufficiencies (e.g., a classification algorithm that does not detect an object [34]) and is involved in some processes described herein and in Part 2 [1].

In the industry standards, some methods are thought to support life-cycle activities. For instance, IEC 61508-3 [31] recommends the use of fault-tree (FT) analysis, reliability block-diagram analysis, and event-tree analysis, among other methods. In a number of cases, these methods use the findings from an FMEA as input, but they are not able to analyze the detailed risk aspects of software.

FMEA is a bottom-up analysis, which considers the failure of individual components and their associated effect on the overall system. The standard for hardware FMEA, IEC 60812 [35], is commonly used as a basis for software FMEA, as no formal process for software FMEA is defined [36]. However, several taxonomies specifically for software FMEA have been described in some detail in Section 2.3.

2.2 Software Failure and Risk

In contrast to hardware systems, software fails mainly due to design and coding error. Software does not have a time-dependent failure rate. Software failure mechanisms, such as common-cause failure (CCF), differ from those that cause hardware failures [11].

To describe most software-related failures, generic failure modes may be used so that they can be implemented in risk analysis [37]. "A failure mode is the manner in which a failure occurs" [9]. A failure mode may be determined by a function lost, intended behavior not provided, or a state transition that occurred [9, 35, 38]. For software, this may be determined by a function lost as a result of software output, whether omitted or committed. In this regard, it is analogous to the definition of human failure modes [35].

Ensuring software reliability is a focus of the design and development of most modern systems. However, reliable software may still lead to hazardous situations and contribute to unacceptable levels of risk [13]. Several incidents show that correctly and reliably working software under some circumstances may lead to accidents (e.g., on space missions [39, 40] or in marine control systems [41]). The reliability analysis of software is not context specific and is based on the amount of bugs removed [42]. These methods help to understand failure causes and consequences in the software–hardware system only to a limited extent. Hence, reliability-assessment methods are not very useful for risk analysis.

Garrett et al. [21] and Guarro et al. [43] developed DFM to assess the dependability and safety of software systems. DFM is a two-step process: (i) build the model for the software system and (ii) analyze the model to build FTs. A model using DFM is a directed graph with functional relations (the causality network) and conditions that trigger functional relations (the conditional network). The software system is seen as a flow of information that is manipulated by different software functions. Failures are only assessed if they are identified as relevant beforehand. A timed FT is built into the second step of DFM by assessing which conditions in the DFM model lead to the undesired top event of the FT.

Al Dabbagh [44] and Al Dabbagh and Lu [45] applied DFM to a networked control system of a communication network. Special submodels have been developed to model recurring functions in such a network with a focus on the timing of functions of such a system (e.g., preprocessing times, waiting times, etc.). DFM is applied to the analysis of failures in relation to the flow of information and its timing. For example, missing operations or unanticipated function calls are not considered. The analysis does not rely on information from software documentation, as data-type failures or failures that are related to other interactions between functions might be overlooked.

The National Aeronautics and Space Administration (NASA) [46, 47] has used DFM in their context-based software risk assessment methodology (CSRM). In CSRM, critical-mission stages that include a risk-relevant software contribution are identified. These mission stages are assessed with fault and event trees. Guarro et al. [47] suggested the use of simple logic models, such as FT analysis, for simple software systems or for a high level of modeling abstraction, while recommending DFM for more complex software systems that also have time-dependent behavior.

Aldemir et al. [48, 49] used Markov cell mapping combined with DFM. The method can capture system behavior dynamically and discover event sequences that otherwise are hard to identify by an analyst. They acknowledge that design errors might not be revealed by these methods. The Markov methods are based on analyzing different combinations of states, which requires setting up new models for each analysis. Because DFM is combined with the Markov cell mapping, the limitations of DFM apply to this combined method as well.

Li et al. [26] and Li [10] decomposed software into functional units, and failures of these functions are inserted in FTs and event-sequence diagrams (ESDs) for risk analysis. Only selected failure modes are input directly into the risk analysis, combining different levels of software analysis and decomposition. The concept of functional-failure modes is not applied consistently, and certain failure modes that are not relevant to the functional level may be included.

Wei [50] and Wei et al. [27] presented a framework that includes the risk contribution of software in risk analysis. The framework comprises four steps: input-failure analyzer, operational-profile builder, software-propagation analyzer, and probabilistic risk assessment updater. The results of the analysis can be included in an ESD and FT. The method is only applicable to existing software systems and not suitable for the design phase. Moreover, the analysis does not make use of all the information that is typically available (e.g., software specifications or safety requirements), which would reveal deficiencies with respect to the requirements.

Zhu [51] and Zhu et al. [25] built on the work of Wei [50] and included software failure in dynamic risk analyses, which also consider the relative timing of events. Random software failures are *injected* into a dynamic model, and the simulation *reacts* to these failures. Associated faults and event trees are built automatically by the system. The software behavior and failures are represented in finite-state machines. In their construct, the simulation model covers only selected failure modes and their influence on dynamic behaviors.

Leveson [52] and Leveson et al. [53] stated that systems-theoretic process analysis (STPA) is a hazards-identification method that is also suitable for software. In their construct, hazards arise from insufficient control action: not providing a necessary control action; providing an unsafe control action; providing a potential control action too late, too early, or out of sequence; or providing a safe control action that is too short or too long. Abdulkhaleq and Wagner [54] and Abdulkhaleq et al. [55] extended the STPA for automated model checking of critical software applications by identifying potentially hazardous situations arising from a software model and by verifying that the already-taken control actions are safe.

Rokseth et al. [56] recommended combining FMEA and an adapted version of STPA to improve the identification, analysis, and verification of hazardous events and failure modes of dynamic positioning (DP) systems on ships and offshore oil and gas rigs. STPA and FMEA were found to be complementary and a combination well suited to complex and software-intensive systems such as DP. Positioning systems will be crucial for autonomous systems such as ships [57, 58].

Gran [59] developed an influence network to assess the quality of software processing, the resulting quality of the software, and the associated risk. However, this approach does not consider the specific purpose of the software and the influence on the risk level or hazards that arise from the software. Therefore, it is not possible to identify and incorporate hazards in risk analysis by this approach.

Hewett and Seker [60] analyzed the risk of embedded software systems with timed-decisions tables. The approach is like DFM. Decision tables represent software behavior, which is decomposed into functional modules. From the decision tables, timed FTs are built based on a predefined initiating event through backward reasoning. Similar to DFM, only failures that are related to a wrong value and the associated decisions are considered. Hence, it is not possible to identify failures and their contribution to the risk level that relate to unanticipated interaction of functions or data-type failures, for example.

Sadiq et al. [61] proposed software risk analysis using software FT analysis. The framework is intended for prioritizing testing and for software improvement but also highlights the need to consider software requirements, modeling uncertainty, and possible errors in the analysis. However, the method only addresses the software-system level; it does not identify events that might arise from within the software, such as interactions with other system components.

Functional-failure identification and propagation methods [62-64] and a failure propagation and simulation approach (FPSA) [64] have been developed to assess system behavior in cases of one or multiple hardware or software faults. Several models such as state-space models and function-flow models have been used to assess propagation. An FPSA module allows simulation to identify time-related failure effects. However, software and hardware failures need to be identified by the analysts individually, failure-propagation behavior needs to be defined specifically for each function, and failures need to be injected into the simulation.

A starting point for risk analysis is to identify hazards and hazardous events. Because taxonomies with failure modes may be used as a basis for hazard identification, we conducted a search to identify existing software failure mode taxonomies.

2.3 Software Failure Mode Taxonomies

Searches in the Institute of Electrical and Electronics Engineers (IEEE) database Xplore[1] and in Scopus[2] (using the keyword phrases *software failure mode identification*, *software FMEA*, *software FMECA*, *software failure mode effect analysis*, and *software failure mode effect criticality analysis*) were conducted.

The searches covered only publications from 2000, based on the assumption that these publications also reflect previous taxonomies. Publications that include relevant taxonomies have been closely investigated and selected for further analysis. Taxonomies that are the same in several publications have been assessed only once. Relevant taxonomies were found in publications, published between 2002 and 2014.

Li et al. [26] and Li [10] developed a failure mode taxonomy for software functions. They defined several types of failure modes, in two categories: functional-failure modes (comprising attribute and functional or

[1] http://ieeexplore.ieee.org/ (accessed Feb. 02, 2018).
[2] https://www.scopus.com/ (accessed Dec. 08, 2017).

function-set failure modes) and external failure modes (comprising timing, input/output, multiple-interaction, and support failure modes).

Input/output failure modes are those that do not originate from the software function itself [10]. An input failure will lead to an output failure. Input/output failure modes can be further divided into value-related and timing-related failure modes [10].

Multiple-interaction failure modes concern communication through a common language to exchange information [10]. Support failure modes comprise those related to hardware resources and the physical operating environment and thus do not apply directly to software functions. For the most part, they are not addressed herein, although they do fall into the category of software failure causes (see Section 2.4).

The Organization for Economic Cooperation and Development (OECD) [37] has presented a taxonomy for hardware and software failure modes that builds on research by Li and by Li et al. [10, 26, 65], Authen et al. [66], Authen and Holmberg [67, 68], and Holmberg et al. [69], among others. It addresses different levels of the system: overall system, division, instrumentation, and control levels.

Ristord and Esmenjaud [16], Huang et al. [70], Stadler and Seidl [17], Park et al. [20], and Prasanna et al. [19] presented and discussed their own adaptations of software FMEA. Although such prior reported research offers a basis for the identification of possible failure modes, clear descriptions and distinctions among the targeted levels of software abstraction (e.g., software system, software functions, and code analysis) for the taxonomies are absent. Only the taxonomy by OECD [37] attempts such distinctions. However, owing to the definitions of system, module, and submodule levels, there is some ambiguity as to the failure modes and their effects and causes with respect to the level of analysis. In addition, that report includes hardware failure modes for different system levels. However, some of the failure modes presented as software failure modes actually relate to the hardware system but can be seen as the cause for software failure. Section 4.3 elaborates on some of the ambiguous failure modes, and Section 2.4 and 2.5 specifically define the software causes and consequences to avoid such ambiguity, respectively.

2.4 Failure Causes

Each failure mode may be attributable to one or more failure causes [71]. A failure cause is “the set of circumstances that lead to a failure” [35]. The causes of software failure can be found in its specification, design, or implementation [16], and a NASA document [46] states parameter and data-entry errors and defects introduced during the removal of other defects as additional causes. Stadler and Seidl [17] mentioned infinite loops, multi-process thread/deadlock, counter rollover, numerical overflow/underflow or saturation, and finite precision errors among other potential failure causes.

Ozarin [12, 18, 36, 72] highlighted the need to consider the inherent interaction of software–hardware interfaces when analyzing software, especially with respect to causes such as bad input data or analog–digital converter failure. The computing hardware may also be the reason for failures that manifest themselves as software failures. ISO 26262-11 [73] presents failure modes and models for failure mode effects for semiconductors (i.e., the computing hardware). Such failures may be related to safety elements out of Context (SEooC), [38], which are systems that were not specifically developed for a safety function but are necessary to carry out a safety function, such as the memory-wiper system in a software operating system or in the hardware [38].

2.5 Failure Mode Propagation

Failure propagation determines how a failure mode in one function will affect the software system [50]. The failure effect of a failure mode may be defined as the “*consequence of a failure within or beyond the boundary of the failed item*” [35]. A software failure may lead to consequences for the software itself and/or for the system that uses the software. The two main categories of failure propagation are CCF and cascading failures [37, 38]. “A cascading failure is a failure of an element of an item resulting from a root cause [inside or outside of the element] and then causing a failure of another element or elements of the same or different item” [38, 73]. A CCF occurs if two or more elements of an item fail directly from a single specific event or root cause, which may be internal or external to all these elements [38]. This contrasts with common mode failures, which describe several failures that have failed in the same manner [38].

Propagation means that the failure is not discovered or is masked during the execution of the program. Masking describes the situation in which software produces the right output despite a failure during the execution. Multilayer traps might conceal a failure through several subfunctions of the software, according to Wei [50], who derived a set of propagation mechanisms for software failure modes, which should be considered in risk analysis.

Failures may also propagate by other means, such as from the computing hardware or sensors to the software system [73]. Such failures are among the causes of failure modes discussed herein but are not the focus of either this article or Part 2 [1].

3 Functional View of Software

The software system can be decomposed into its functions. IEC 61508 [28] defines a functional unit of a system as an "entity of hardware or software, or both, capable of accomplishing a specified purpose." The purpose of functional decomposition is to enable the identification of relevant failure modes associated with each software function.

Software can be analyzed and broken down into functions at different levels of detail (Figure 1). Decomposing the software further would eventually lead to the software code level in an abstracted form, represented by pseudocode. Such a low level of decomposition is not covered by the taxonomy herein.

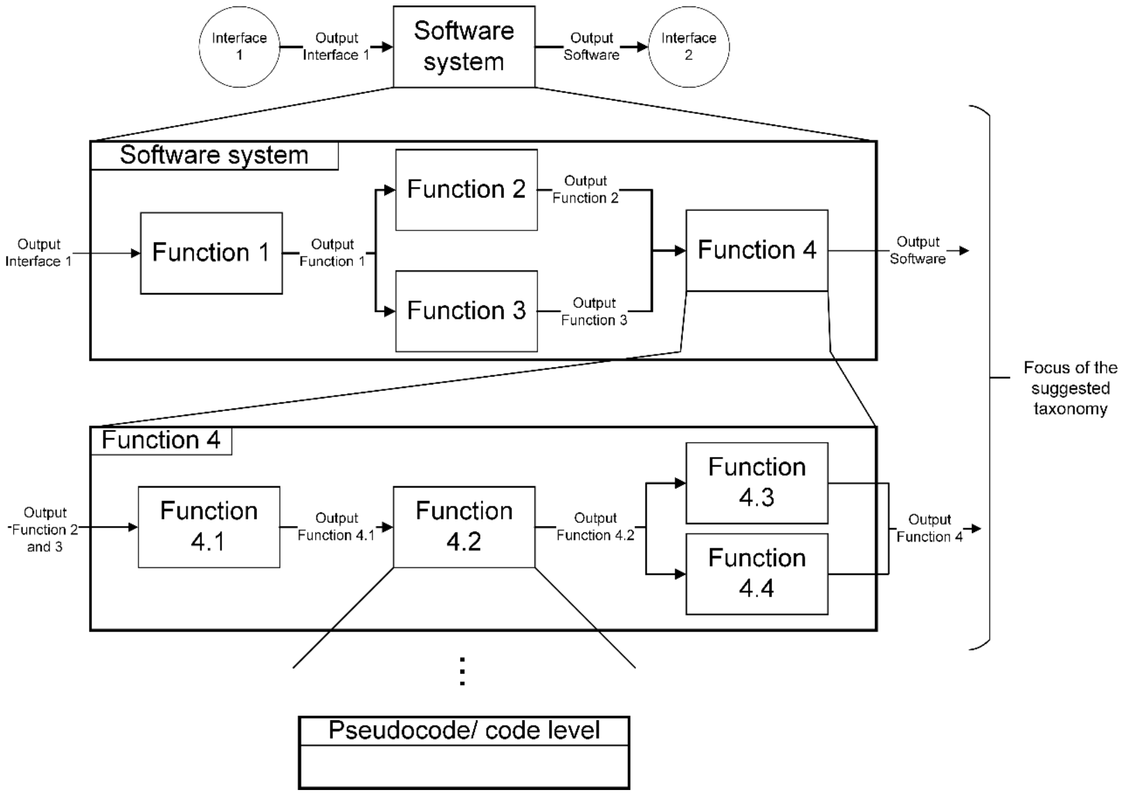

Figure 1 Different levels of software functions of a software system.

Beginning from the overall functional description, the software should be decomposed into subfunctions, which describe what the software should do, not how it is implemented. EN 14514 [74] also provides guidance for decomposition. Design maturity [74] and the depth of analysis of the software are two factors that determine the level of decomposition. Information for the decomposition can be extracted from the safety requirements specification, such as defined by IEEE 830 [75]. This functional view of software can be used in different software-development life cycles (such as the waterfall model, the V-model, or the Scrum method).

Depending on the software-development approach, different activities and requirements are needed [76]. A functional decomposition may be carried out during software-development phase. The results from this decomposition should be used.

A generic function and its main elements once the desired level of breakdown and resolution is achieved are shown in Figure 2, which also shows where the different categories of failure modes can be applied. The process section is where the functional behavior and computation are executed, turning input into output. Function failure modes are associated with this input–process–output section.

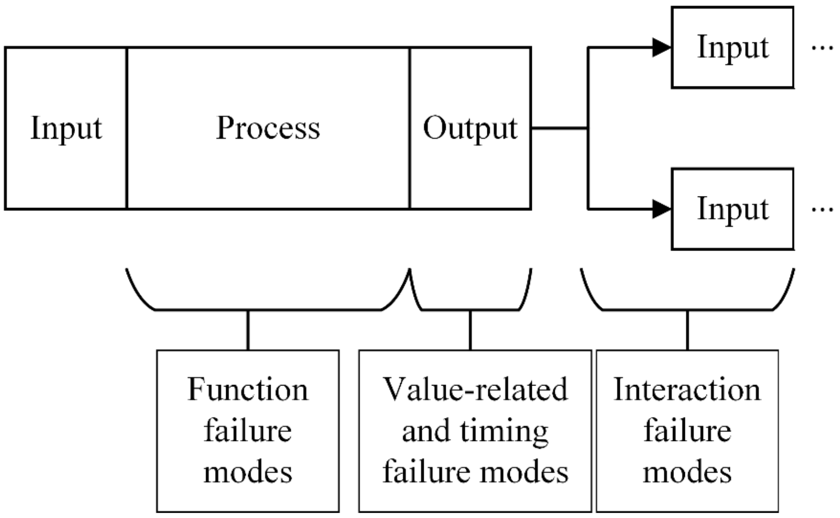
Input
Process
Output
Input
...
Input
...
Function failure modes
Value-related and timing failure modes
Interaction failure modes

Figure 2 Simplified view of software function and its components underlying all levels (developed and extended from Huang et al. [70]).

The description of each function includes the purpose, process, input–output, conditions of execution, requirements and constraints, failure detection, and correction mechanisms. An example datasheet for describing a software function is shown in Table 1. Not all the information may be available [11], but collection of as much as possible is important to determine the relevant failure modes.

Table 1 Example of software-function datasheet.

ID	**Data for function**						
Function purpose	Short description of what function is to achieve						
Inputs	List and description of inputs received by function						
	Input name	**Source**	**Data type**	**Data format**	**Data range**	**Data rate**	**Buffer**
Outputs	List and description of outputs that function produces						
	Input name	**Target**	**Data type**	**Data format**	**Data range**	**Data rate**	**Buffer**
Conditions	Trigger conditions						
	Conditions that trigger other functions						
Process	Describe behavior through input→output formulas						
	Consider dependencies and sequence of operations						
Requirements	**Functional**	Related to function itself (e.g., accuracy)					
	Nonfunctional	Related to speed, security, safety, use of resources, etc.					
Constraints	Factors that limit way function could be implemented (e.g., regulatory or hardware constraints; high-order language requirements; signal-handshake protocols; and application criticality)						
Failure detection and correction features	Measures implemented to detect, handle, and warn about software failures (e.g., control function, input-validity checks, and error-handling system)						

A function always has at least one output. This might be a numerical value, a binary value, or a specific function call. Input is the output of another function or is provided via external interfaces. An output of one function can also be the input to several other functions, and each function might have several inputs and several outputs.

Input and output have an associated data type and an acceptable range, which might be limited by the data type, acceptable values, or the set of meanings assigned to the values. If the output is part of a data array or structure, data format refers to the order of elements. The data rate (the periodic output and its characteristics) and buffer (the type used for input or output to collect data or events) only need to be described if applicable. Both value-related and timing-related failure modes are associated with this part of a software function.

Functions in a software system are executed in a specified order. They might be executed periodically or on demand, depending on the result of the operation. Each function passes on information or calls another function. These interactions (represented by arrows in Figures 1 and 2) have associated interaction failure modes. External interfaces (agents that interact with the software) include other software systems, sensors, databases, or human operators through a human–machine interface.

In the subsequent article (Part 2) [1], we propose a process to incorporate software in risk analysis. For this purpose, software-function failure modes are identified, and their effects at the software-system level are analyzed. The decomposition of software into functions is an essential part of that proposed process [1].

4 Proposed Taxonomy

4.1 Procedure

To determine whether failure modes presented in other published studies are relevant for the functional level of software and the software-function failure mode taxonomy, we asked three questions:

1. Does the presented failure mode fall into our definition of a failure mode?
2. If yes, is it within one of our failure mode categories (interaction, function, value-related, or timing-related)?
3. If yes, is the failure mode different from failure modes that were previously identified?

If all the questions were answered "yes," the failure mode is included in the failure mode taxonomy; however, if it does not fulfill our definition of a failure mode or does not fit into one of the mentioned categories, it is rejected. It is necessary to define an unambiguous and consistent failure mode taxonomy. Where necessary, distinctions between similar failure modes are mentioned to give more guidance for their use, and they are labeled as refined failure modes. To retain the experiences, practices, and findings from others' research, our generic failure mode taxonomy includes mainly published failure modes.

Contributions from relevant research were identified by screening past publications (Table 2). They contain types of failure modes that are relevant with respect to the software-function level. All reviewed taxonomies cover value-related failure modes. All the screened publications except Prasanna et al. [19] cover timing-related failure modes, and all except Wei [50] consider interaction failure modes. Function failure modes are covered by only five of the publications.

Table 2 Publications that form basis for identification of software-function failure mode taxonomy.

Publication	**Number of failure modes**		**Failure mode coverage**			
	Presented	**Relevant**	**Function**	**Interaction**	**Timing-related**	**Value-related**
Ristord and Esmenjaud [16]	12	5	Yes	Yes	Yes	Yes
Li [10]	31	19	Yes	Yes	Yes	Yes
Wei [50]	12	12	No	No	Yes	Yes
Huang et al. [70]	25	22	No	Yes	Yes	Yes
Stadler and Seidl [17]	21	17	Yes	Yes	Yes	Yes
OECD [37]	37	22	Yes	Yes	Yes	Yes
Park et al. [20]	21	14	Yes	Yes	Yes	Yes
Prasanna et al. [19]	11	10	No	Yes	No	Yes

Each of the publications has a different focus and therefore presents a different number of failure modes in each category, with different levels of detail. Most failure modes are presented in the OECD [37] study.

Several failure modes have been rejected for our proposed taxonomy. For instance, Stadler and Seidl [17] included *memory-address errors* in their taxonomy; however, these are not relevant from a functional point of view, although they are causes of functional failure. Similarly, the failure modes *central-processing-unit failure* [26], *memory failure* [26], *deadlock* [26], and *stop of operating system* [16] cause but do not represent function failures. These are failures of the computing hardware and are considered failure causes for the software-function failure modes.

Interrupt [17] or *interrupt-induced failures* [37] and *raised execution* already imply that they are failure causes, not failure modes, and hence were excluded. The failure mode *wrong task scheduling* [19] is a very general description representing several interaction failure modes.

The failure modes *software aborts* [38], *hang/crash* [both in 37], *program stop with/without clear message* [16], and *fail to return/complete* [17] were rejected because they represent effects of failure modes at the software-system level.

4.2 Taxonomy

The resulting taxonomy for software-function failure modes is shown in Table 3. Six identified failure modes address only the processing part of a function.

Table 3 Taxonomy for software-function failure modes.

Failure mode	**Additional description**
Omission of function or missing operation	Function (or part of it) is not executed.
Incorrect functionality	Function is not executing intended actions.
Additional functionality	Extra unspecified operation in function is executed by function.
No voting	Voting on input is not carried out within function.
Incorrect voting	Voting on input is not carried out according to specification within function, and therefore voting result is incorrect.
Failure in failure handling	Detected failures are not handled appropriately.

Table 4– Table 6 include a column for *refined failure mode*, which represents a more detailed case of the failure mode. This was done to retain the knowledge presented in prior publications while still classifying the failure modes generically.

Table 4 summarizes the interaction failure modes between software functions. These failure modes reflect a failure of interaction between software functions. Seven failure modes were identified for the interaction between functions. Ten refined failure modes were identified for the interaction between software functions and external files or databases.

Table 4 Taxonomy for interaction failure modes of software functions.

Failure mode	Refined failure mode	Additional description
Diverted or incorrect functional call		Wrong function is called after current function is finished.
No call of next function		No further functions are called after current function is finished.
No priority for concurrent functions		Functional calls for functions that need to be executed concurrently are given no priority.
Incorrect priority for concurrent functions		Functions needed to be executed concurrently are given incorrect priority.
Communication protocol-dependent failure modes		Failure modes specific to particular communication protocol used to interchange information between parts of software system.
Unexpected interaction with input–output (IO) boards		Failure mode related to interaction (possibly spurious) with input–output board or interface.
Failure of interaction with external files or databases	Wrong name	Name of file or database is incorrect.
	Invalid name/extension	Name entered for file or database contains invalid symbols.
	File/ database does not exist	File or database name appears to be specified correctly but file or database does not exist.
	File/ database is open	File or database is open in another program and cannot be reopened.
	Wrong/invalid file format	File format is different from expected file format.
	File head contains error	File-header information contains different information from that required.
	File ending contains error	File-ending information contains different information from that required.
	Wrong file length	Length of file is different from required or expected length.
	File/database is empty	
	Wrong file/database contents	Information in file or database is different from expected or required information.

Four timing-related failure modes are summarized in Table 5. Five refined failure modes were identified for the timing of the provided output, and four were identified for output-rate failures.

Table 5 Taxonomy for timing-related failure modes of software functions.

Failure mode	Refined failure mode	Additional description
Output provided	Too early	
	Too late	
	Spurious	Output provided when not requested or not needed.
	Out of sequence	
	Not in time	No output is provided from function.
Output-rate failure	Too fast	
	Too slow	
	Inconsistent	
	Desynchronized	
Duration	Too long	Length of time output is available.
	Too short	
Recurrent functions scheduled incorrectly		Periodically required output not delivered at expected time.

Table 6 summarizes the 11 software-function failure modes related to value, which here refers to the content assigned to a variable. This may be a numerical value, or it may be a character or symbol in string format. Four refined failure modes relate to the failure mode *incorrect value*, five to data arrays or structures, and three to data validation.

Table 6 Taxonomy for value-related failure modes of software functions.

Failure mode	Refined failure mode	Additional description
No value		No value is provided.
Incorrect value	Too high	Value is higher than expected or required value. This might be 1, maximal allowed, or higher increment of value.
	Too low	Value is lower than expected or required value.
	Opposite/ inverse value	Value is opposite or inverse of expected value.
	Value is zero	Value is zero instead of expected value.
Value out of range	Data-type allowable range	
	Application allowable range	
Redundant/ frozen value		Same value is produced constantly.
Noisy value/ precision error		Values that are transferred are not precise enough.
Value with wrong data type		
Nonnumerical value	Not a number (NaN)	Values are transferred that are not interpretable by software.
	Infinite	
	Negative infinite	
Elements in data array or structure	Too many	
	Too few	
	Data in wrong order	
	Data in reversed order	
	Enumerated value incorrect	Wrong element in data array or structure is addressed.
Correct value is validated as incorrect		
Incorrect value is validated as correct		
Data are not validated		Validity check is not executed.

4.3 Discussion

A clear distinction between failure mode, failure cause, and failure effect is difficult to achieve. The taxonomy proposed herein (Tables 4–6) attempts to clearly separate failure effects and causes from the failure modes for the functional level. Hence, failure modes such as *incorrect realization of an attribute or function* [26] or *incorrect realization of a function* [37] were not included in the proposed taxonomy as they are considered to be failure causes, originating from the software realization process.

Some failure modes have refined failure modes. For example, *interaction with external files or databases* is refined by several subordinate (refined) failure modes. To identify a failure mode as just a *wrong interaction with external files or databases* is not useful for further analysis; therefore, it is necessary to specify how it is interacting wrongly.

Similarly, for timing-related failure modes, *output-rate failure* is rather vague. Hence, the refined failure modes *too slow, too fast, inconsistent*, and *desynchronized* were retained from published research. Especially in the category *value-related failure modes*, several distinctions were made. The failure mode *incorrect value* would cover most of the failure modes but is too generic in a number of cases. Therefore, refined failure modes for *incorrect value* were introduced. In addition, *nonnumerical values* were differentiated, as they have a different effect on the software function from that of an incorrect numerical value. This adds more meaning to the failure modes and allows application-specific failure mode analysis.

The chosen perspective on software is challenging in terms of the identification of a sufficiently low level of decomposition. The level of detail of software decomposition depends on the maturity of software development and the purpose of the analysis, such as a detailed risk study. A functional view of the software allows analysis of the software in an early development stage while it is still independent of the implementation. Especially during the early stage of development, software documentation is apt to be immature, and decomposition may only be possible at a higher level. Decomposition down to the code level is not recommended because even medium-sized software projects have several tens of thousands of lines of code.

Failures of SEooC may be captured with the presented failure mode taxonomy. To be included in the analysis, the SEooC needs to be modeled and represented explicitly. Because the presented taxonomy focuses only on software-intensive systems, it cannot be used to assess hardware SEooC.

5 Case Study

To demonstrate possible applications of the proposed taxonomy, a case study is included in this section. Hegde [79] and Hegde et al. [77, 78] presented an underwater operational-envelope visualizer (OEV) that combines safety envelopes and subsea traffic rules for an autonomous remotely operated vehicle (AROV). According to Hegde et al. [79], an AROV can collide with underwater infrastructures, the seabed, and other underwater vehicles. The OEV supports the human operator in detecting hazardous situations that can lead to collision with subsea obstacles [79]. The system is a decision-support system but is not safety critical. The AROV still has a conventional collision-avoidance system. A collision may lead to loss of the AROV, damage to the subsea structure, and damage to the environment due to leakage.

The software was developed by Ph.D. students at the Norwegian University of Science and Technology for demonstration purposes in the research project Next Generation Inspection, Maintenance, and Repair [80]. The underwater OEV was developed in Python to ensure compatibility with other software components. (Notably, the approach presented herein is programming-language independent.) The developers used a rapid prototyping approach, whereby the software was tested and improved iteratively several times. The process described herein (Part 1) and in the subsequent article (Part 2) [1] was applied as a structured process to identify relevant software failure modes and consequently improve the software in the next iteration.

AROVs are tethered underwater robots that have a higher level of autonomy in their operation than conventional remotely operated vehicles but have more human-operator interaction than autonomous underwater vehicles. The underwater OEV provides decision support with respect to safe operation of the AROV. It is necessary to ensure that the underwater OEV does not increase the level of risk. The underwater OEV receives data from a database and provides information for operational decision making. This “Case Study” section focuses on demonstrating the individual failure modes in an application setting, and the corresponding processing steps are further discussed in the subsequent article (Part 2) [1].

5.1 Functional Decomposition

The functional hierarchy (Figure 3) identifies five sequential functions of the software in the case study on the first level of decomposition: initialize underwater OEV, obtain data, determine suggested action, prepare rendering information, and display information. The software for the underwater OEV has only about 1,000 lines of code. Data collection and control of the AROV are executed by other dedicated software systems, which are not part of this analysis.

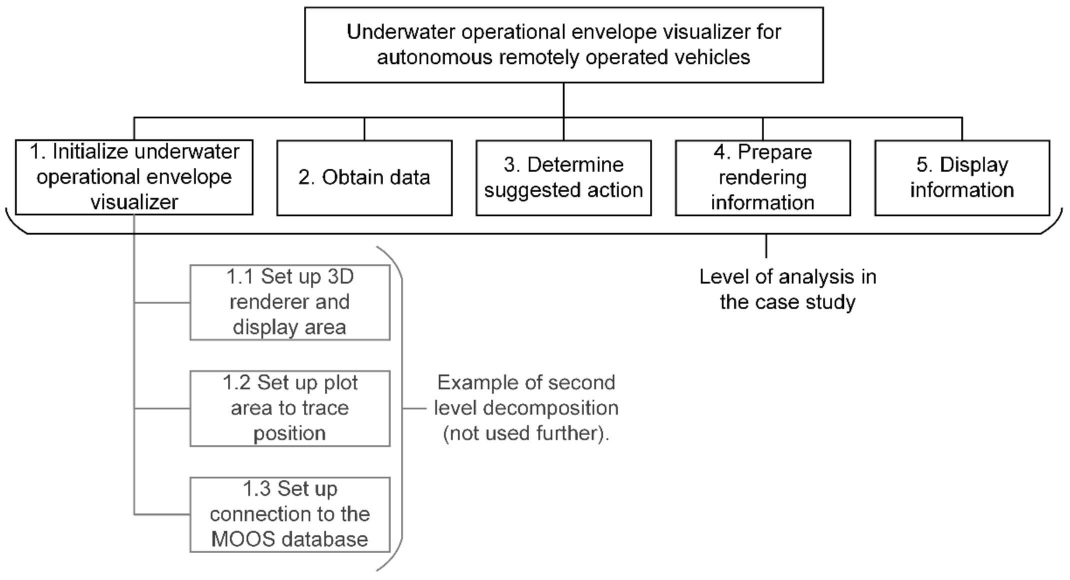
Underwater operational envelope visualizer for autonomous remotely operated vehicles
1. Initialize underwater operational envelope visualizer
2. Obtain data
3. Determine suggested action
4. Prepare rendering information
5. Display information
Level of analysis in the case study
1.1 Set up 3D renderer and display area
1.2 Set up plot area to trace position
1.3 Set up connection to the MOOS database
Example of second level decomposition (not used further).

Figure 3 Functional decomposition of underwater operational-envelope visualizer (OEV).

Hence, it was decided that a decomposition to the first level is enough. As an example, *initialize underwater OEV* was decomposed to the second level. This software was chosen as a demonstrator for two reasons: (i) all failure modes in the proposed taxonomy can be demonstrated and (ii) access to the software developers aids the understanding and analysis of possible software failure modes. The functions on the second level are already close to pseudocode; therefore, decomposing the function further would lead to code instructions.

Function 2, *obtain data,* serves as a suitable example because it covers a variety of output types and functional behaviors (Table 7). The function polls the database at a frequency of 2 Hz for data on AROV position, operational mode, and orientation (in radians, to be converted to degrees by the function) and for information on identified collision candidates. The database returns the requested values, and the *obtain data* function makes them available to subsequent functions.

Table 7 Datasheet for Function 2, *obtain data*. [AROV, autonomous remotely operated vehicle; :f, float; :i, integer; :s, string; MDb, Mission-Oriented Operating Suite database; N.A., not applicable]

ID: F2	**Function 2: Obtain data**							**ID**
Function purpose	Send request for updated information on parameters (AROV position, AROV orientation, operational-envelope information, AROV operation mode) and make it available for subsequent functions.							
Inputs	**Input name**	**Source**	**Data type**	**Data format**	**Range**	**Rate**	**Buffer**	**ID**
	AROV orientation from database	MDb	float	:f, :f, :f	0 to 2*π	2 Hz	N.A.	MDb.O1
	AROV operational mode from database	MDb	integer	:i	0 to 2	2 Hz	N.A.	MDb.O2
	AROV position from database	MDb	float	:f, :f, :f		2 Hz	N.A.	MDb.O3
	Information on identified collision candidates	MDb	string	:s, max. 64 elements	00-07, 10-17, 20-27, 30-37, 40-47, 50-57, 60-67, 70-77	2 Hz	N.A.	MDb.O4
Outputs	**Output name**	**Target**	**Data type**	**Data format**	**Range**	**Rate**	**Buffer**	**ID**
	Request for AROV orientation	MDb	string			2 Hz	N.A.	F2.O1
	Request for AROV operational mode	MDb	string			2 Hz	N.A.	F2.O2
	Request for AROV position	MDb	string			2 Hz	N.A.	F2.O3
	Request for information on identified collision candidates	MDb	string			2 Hz	N.A.	F2.O4
	AROV orientation	F4	float	:f, :f ,:f		N.A.	N.A.	F2.O5
	AROV operational mode	F4	integer	:i	0 to 2	N.A.	N.A.	F2.O6
	AROV position	F4	float	:f, :f, :f		N.A.	N.A.	F2.O7
	Information on identified collision candidates	F3	string	:s, max. 64 elements	00-07, 10-17, 20-27, 30-37, 40-47, 50-57, 60-67, 70-77	N.A.	N.A.	F2.O8
Conditions	Initiated by F1							F2.C1
	Initiated by F4.2 after first iteration							F2.C2
	Initiate F3							F2.C3
Function behavior	Send request for AROV position, AROV orientation, AROV operational mode, and information on identified collision candidates.							F2.B1
	Convert AROV orientation from radians to degrees.							F2.B2
	Store values of AROV position, orientation, and operational mode and store information on identified collision candidates in corresponding variables.							F2.B3
Requirements	**Functional**	None						
	Nonfunctional	Poll MDb with 2 Hz						F2.NF1
Constraints	Successful connection to MDb in F1							F2.Ct1
Failure detection and correction features	Request to database for nonexistent data returns error message and does not return value.							F2.D1

5.2 Application of the failure mode taxonomy

Table 8 presents the identified failure modes with the taxonomy for Function 2. One developer of the underwater OEV (one co-author) and a risk analyst (the first author) carried out the analysis. The table does not present all value-related failure modes. More value-related failure modes could be identified similarly to the ones identified in the table. A detailed list of all failure modes would add to the length of the table but not more insight on the identification of failure modes.

The top of Table 8 defines the expected input and output for the example function *obtain data*. This sets the context for the failure mode identification. The failure modes are applied based on the information found in the datasheet in Table 7. The information on the inputs and outputs is necessary for the analysis of value-related failure modes. Conditions describe the functional interactions and dependencies with other functions. Functional and non-functional requirements set the context for the analysis, such as acceptable timing delays or value inaccuracies.

The top part of Table 8shows that it is not always possible to define expected values. They might be unknown due to the complexity of the function or the behaviour of the function over time. In other cases, the expected values are known due to the context. In the case of the function *obtain data*, the expected values are assumed to be known. The AROV is traveling in semi-autonomous mode, Mode 1, from the south to the north without any pitch or roll angle, corresponding to [0, 0, 0]. An object has been detected to the left of the AROV, corresponding to the envelope elements [66, 67, 76, 77]. The exact location of the case study is not relevant, only its accuracy.

The first column in Table 8 is labeled ID for identifier. Each recognized failure mode needs to have an identifier to be able to trace the failure modes. The second column summarizes the element that is affected by the failure mode: the variable, the execution timing, part of the function block, or a functional transfer. In the third column, the failure mode is described and specified.

Table 8 Failure mode identification for *obtain data* function of underwater operational-envelope visualizer. [AROV, autonomous remotely operated vehicle; MDb, Mission-Oriented Operating Suite database]

Expected input		
ID	**Name**	**Expected value**
MDb.O1	AROV orientation from database	[0,0,0]
MDb.O2	AROV operational mode from database	1
MDb.O3	AROV position from database	Correct (not further specified)
MDb.O4	Information on identified collision candidates	[66, 67, 76, 77]
Expected output		
ID	**Name**	**Expected value**
F2.O1	Request for AROV orientation	Correct request
F2.O2	Request for AROV operational mode	Correct request
F2.O3	Request for AROV position	Correct request
F2.O4	Request for information on identified collision candidates	Correct request
F2.O5	AROV orientation	[0,0,0]
F2.O6	AROV operational mode	1
F2.O7	AROV position	Correct (not further specified)
F2.O8	Information on identified collision candidates	[66, 67, 76, 77]
F2.C3	Initiate F3	-

ID	**Associated element**	**Failure mode**
Function failure modes		
FM1	F2	*Omission* of "Obtain data," which is not executed
FM2	F2.B1	*Omission* of requesting data, which means that data are not requested
FM3	F2.B2	*Omission* of converting MDb.O1 to AROV orientation data, which means that orientation is not executed
FM4	F2.B3	*Incorrect functionality* of storing values in corresponding variables, making them unavailable
FM5	F2.B2	*Additional functionality* while converting AROV orientation (e.g., conversion of AROV position)
FM6	F2.D1	*Failure in failure handling*, not detected that no value has been received
Interaction failure modes		
FM7	F2.C3	*Incorrect function call*, calling Function 4 "Prepare rendering information," skipping Function 3 "Determine suggested action"
FM8	F2.C3	*No function call* to F3
FM9	F2.C3	*Incorrect priority for functions*, call Function F4 "Prepare rendering information," followed by Function 3 "Determine suggested action"
FM10	F2.B1	Unable to request information from database (*communication protocol-dependent failure*)
FM11	F2.B1	*Request with wrong variable name* to database for AROV position
Timing-related failure modes		
FM12	F2.O1	*Output provided too early*: Request for AROV orientation
FM13	F2.O1	*Output provided too late:* Request for AROV orientation
FM14	F2.O1	*Output provided too late (500 ms):* Request for AROV orientation
FM15	F2.O7	*Output provided spuriously:* AROV operational mode
FM16	F2.O8	*Output provided out of sequence:* F2.O8 provided before F2.O7
FM17	F2.O8	*Output not provided in time*: Information on identified collision candidates
FM18	F2.O1–F2.O4	*Output rate too fast: Requests to database* sent too fast

ID	Associated element	Failure mode
FM19	F2.O1–F2.O4	*Output rate too slow: Requests to database* sent too slow
FM20	F2.O1–F2.O4	*Inconsistent* rate for requests
Value-related failure modes		
FM21	F2.O7	*No value* for AROV position
FM22	F2.O7	*Incorrect value* for AROV position (not further defined)
FM23	F2.O6	*Incorrect value*, too high for AROV operational mode = 2
FM24	F2.O6	*Incorrect value*, too low for AROV operational mode = 0
FM25	F2.O5	*Incorrect* value, too high, AROV orientation [0,0,−15]
FM26	F2.O5	*Incorrect* value, too high, AROV orientation [0,0,−30]
FM27	F2.O7	*Incorrect* value, *zero* for AROV position [0,0,0]
FM28	F2.O8	*Value out of application allowable range* for information on identified collision candidates includes value 68
FM29	F2.O6	*Value out of data-type range* for AROV operational mode = 2,147,483,648
FM30	F2.O8	*Frozen value* for Information on identified collision candidates (no collision candidates detected)
FM31	F2.O7	*Imprecise value* for AROV position (varying more than 1 m)
FM32	F2.O6	*Wrong data type* for AROV operational mode, string instead of integer
FM33	F2.O8	*Too many elements*, 65, in information on identified collision candidates
FM34	F2.O5	*Too few elements* (two instead of three) in AROV orientation
FM35	F2.O7	*Data in wrong orde*r in AROV position [z,x,y] instead of [x,y,z]
FM36	F2.O5–F2.O8	*Incorrect value (no value) is validated as correct* and is output

The applied failure modes from the presented taxonomy are marked explicitly in italics in Table 8. The case study demonstrates that different levels of detail can be applied to the identified failure modes, such as FM22, FM23, FM25, and FM26. FM22 indicates that the value is generally incorrect. With the background information and level of detail available, it is enough to describe it as incorrect. For FM23, because the expected value is known, a definite value can be associated. Both FM25 and FM26 are special cases of values that are too high. It is occasionally necessary to differentiate in incremental steps, as different values imply different interpretations of the failure mode and may lead to different risk contributions. Similarly, for timing, different levels of detail can be applied (e.g., FM13 and FM14). With a too-late value of 500 ms, FM14 is a refined version of FM13.

5.3 Discussion

The case study demonstrates how failure modes can be identified for different elements of a function and how several failure modes can be applied to the functional level of a software system. The case study is relevant both because the OEV may contribute to the risk level and because the OEV could benefit from the improvement measures suggested. Not all failure modes could be applied and demonstrated, because not all failure modes were relevant for the case study and because there would have been some amount of repetition of similar failure modes. However, application of the other failure modes would be like the example laid out. The risk analysts along with software developers should be able to apply the failure modes in a manner that is relevant to the context. This is only possible if the analysts have a common understanding of the software system and the associated terminology.

It was demonstrated how different levels of detail can be integrated into the identification and application of failure modes. For a further example, value-related failure modes can be described very generically as incorrect, or in relation to a specific value, or within a specific range of values. This implies that the taxonomy is applicable during different project phases such as at the preliminary-design or detailed-design stage.

The programming language chosen in the case study, Python, may be seen critical. Python is not a recommended programming language for safety critical systems [81]. However, the underwater OEV does not perform a safety-related function as defined in IEC 61508-4 [82]. It is a supporting tool for visualization of the state of the systems. However, a software failure in the OEV may lead to accidents that may result in severe losses and environmental consequences, as is demonstrated through the case study herein (Part 1) and in the subsequent article (Part 2) [1].

One shortcoming of the case study is that the underwater OEV was not developed according to a software-development standard. Hence, the amount of information documented was limited. However, the main developer of the program is one of the authors of this article and who provided additional information when necessary.

6 Conclusion

This article presents a functional-failure mode taxonomy for software functions of a software system. Although no clear definition of the functional-software level and the associated description of generic failure modes for that level exist yet, we have defined and clarified herein the concept of software functions and the associated software-system failure modes for risk analysis purposes. The taxonomy was synthesized from prior published research and suits the functional view that we have taken.

A functional view makes the analysis scalable and modular, and it is appropriate for risk analysis. The system can be broken down into the desired level of detail and based on the availability of information at a given phase in the software life cycle. This implies that the analysis is transparent to the level of decomposition. Lower functional levels of the software, treated as “black boxes,” are not analyzed further. Because the immediate effect on software output might not be derived directly from the functional-failure modes, failure-effect propagation is needed.

Having a generic failure mode taxonomy that can be applied to software functions through the provided guidance may facilitate the identification process. It may contribute to an improved identification of software-function failure modes and contribute to a systematic and thorough software failure mode identification process. In addition, the proposed taxonomy may add to traceability and hence efficiency, given the comparable structure and wording since different analysts and developers have a comparable basis. Therefore, it is believed that the generic failure modes and the functional analysis presented herein may lead to improved software risk analysis.

Because a functional analysis can be carried out at an early development stage, the failure modes can be identified and used from early on. For example, this could help support activities related to the development of safety-related systems according to IEC 61508 or its industry-specific standards. Although the failure mode taxonomy was developed for the context of analysis processing [1], it may be used in and to improve software FMEA or design FMEA.

The application of failure mode taxonomy was tested on an actual software program. Although the process is time consuming. It may nonetheless be more efficient, as the analysts receive guidance through the generic failure modes. A computer-aided tool could be used for the identification process to reduce the associated workload and documentation.

The subsequent article (Part 2) [1] presents a process for incorporating software in risk analysis. This process uses the failure mode taxonomy and analysis of the effect of the software failure modes on the external interfaces. These identified effects may be included in risk analysis.

The proposed taxonomy only considers the functional level of software. In the future, it might be useful to identify failure modes on levels such as the software-system level or the code level and clearly define these, building on and extending previous work. Considerations such as those presented in 26262-11 [73] should also be included. As discussed in Section 2.4, there may be several causes for a software failure. Identifying potential causes is subject to further work.

Acknowledgments

The authors thank four anonymous reviewers whose comments helped to improve this article.

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 was 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 NTNU 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, Los Angeles.

The authors would like to thank Enago for the English language review.

References

[1] Thieme CA, Mosleh A, Utne IB, Hegde J. Incorporating Software Failure in Risk Analysis – Part 2: Risk Modeling Process and Case Study. Submitted for review to Reliability Engineering and System Safety. submitted: pp. 1-33.
[2] Mosleh A. Pra: A Perspective on Strengths, Current Limitations, and Possible Improvements. Nuclear Engineering and Technology. 2014;46: pp. 1-10.
[3] 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
[4] Kongsberg Maritime. Yara and Kongsberg Enter into Partnership to Build World's First Autonomous and Zero Emissions Ship. 2017; https://www.km.kongsberg.com/ks/web/nokbg0238.nsf/AllWeb/98A8C576AEFC85AFC125811A0037F6C4?OpenDocument; Accessed: 27.07.2017
[5] Kongsberg Maritime. Automated Ships Ltd and Kongsberg to Build First Unmanned and Fully Autonomous Ship for Offshore Operations. 2016; https://www.km.kongsberg.com/ks/web/nokbg0238.nsf/AllWeb/65865972888D25FAC125805E00281D50?OpenDocument; Accessed: 24.04.2018

[6] Nautilus Federation. Report of a Survey on What Maritime Professionals Think About Autonomous Shipping. Regulatory scoping exercise for the use of maritime autonomous surface ships (MASS). London, UK: International Maritime Organization, Maritime Safety Committee; 2018. pp. 4-17.
[7] Thieme CA, Utne IB, Haugen S. Assessing Ship Risk Model Applicability to Marine Autonomous Surface Ships. Ocean Engineering. 2018;165: pp. 140 - 154.
[8] Kaplan S, Garrick BJ. On the Quantitative Definition of Risk. Risk Analysis. 1981;1: pp. 11-27.
[9] ISO/IEC. Iso/Iec Guide 51: Safety Aspects - Guidelines for Their Inclusion in Standards. Geneva, Switzerland: International Organization for Standardization , International Electrotechnical Commission; 2014. pp. 1-22.
[10] Li B. Integrating Software into Pra (Probabilistic Risk Assessment) [Monograph]. College Park, MD: University of Maryland; 2004.
[11] 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.
[12] 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.
[13] Garrett CJ, Apostolakis G. Context in the Risk Assessment of Digital Systems. Risk Analysis. 1999;19: pp. 23-32.
[14] Reifer DJ. Software Failure Modes and Effects Analysis. IEEE Transactions on Reliability. 1979;R-28: pp. 247-249.
[15] Goddard PL. Software Fmea Techniques. Proceedings of the Annual Reliability and Maintainability Symposium. 2000: pp. 118-123.
[16] Ristord L, Esmenjaud C. Fmea Performed on the Spinline3 Operational System Software as Part of the Tihange 1 Nis Refurbishment Safety Case. Cnra/Csni Workshop on Licensing and Operating Experience of Computer-Based I&C Systems. Hluboka nad Vltavou, Czech Republic: NEA/CSN/OECD; 2002. pp. 37-50.
[17] Stadler JJ, Seidl NJ. Software Failure Modes and Effects Analysis. 59th Annual Reliability and Maintainability Symposium, RAMS 2013. Orlando, FL, United States: Institute of Electrical and Electronics Engineers Inc.; 2013. pp. 1-5.
[18] Ozarin NW. Bridging Software and Hardware Fmea in Complex Systems. 2013 Proceedings Annual Reliability and Maintainability Symposium (RAMS). 2013. pp. 1-6.
[19] Prasanna KN, Gokhale SA, Agarwal R, Chetwani RR, Ravindra M, Bharadwaj KM. Application of Software Failure Mode and Effect Analysis for on-Board Software. 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI). 2014. pp. 684-688.
[20] Park GY, Kim DH, Lee DY. Software Fmea Analysis for Safety-Related Application Software. Annals of Nuclear Energy. 2014;70: pp. 96-102.
[21] 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.
[22] Yau MK, Dixon S, Guarro SB. Applications of the Dynamic Flowgraph Methodology to Dynamic Modeling and Analysis. 11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference, PSAM11, ESREL2012. Helsinki, Finland: Probablistic Safety Assessment and Management (IAPSAM); 2012. pp. 606-615.
[23] Karanta I. Implementing Dynamic Flowgraph Methodology Models with Logic Programs. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability. 2013;227: pp. 302-314.
[24] 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.
[25] 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.
[26] 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.
[27] 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.
[28] IEC. Iec 61508: Functional Safety of Electrical/Electronic/ Programmable Electronic Safety Related Systems. Geneva, Switzerland: International Electrotechnical Commission; 2010.

[29] ISO. Iso 26262: Road Vehicles - Functional Safety. Geneva, Switzerland: International Organization for Standardization; 2018.
[30] 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.
[31] IEC. Iec 61508-3: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. Part 3: Software requirements. Geneva, Switzerland: International Electrotechnical Commission; 2010.
[32] ISO. Iso 26262-6: Road Vehicles-Functional Safety. Part 6: Product development at the software level. Geneva, Switzerland: International Organization for Standardization; 2018.
[33] 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.
[34] ISO. Iso 21448: Road Vehicles - Safety of the Intended Functionality. Geneva, Switzerland: International Organization for Standardization; 2019.
[35] 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.
[36] Ozarin NW. Developing Rules for Failure Modes and Effects Analysis of Computer Software. SAE Advances in Aviation Safety Conference - 2003 Aerospace Congress and Exhibition. Montreal, QC, Canada: SAE International; 2003.
[37] OECD. Failure Modes Taxonomy for Reliability Assessment of Digital I&C Systems for Pra. Paris, France: Organisation for Economic Co-operation and Development, Nuclear Energy Agency; 2014. pp. 1-135.
[38] ISO. Iso 26262-1: Road Vehicles - Functional Safety. Part 1: Vocabulary. Geneva, Switzerland: International Organization for Standardization; 2018.
[39] Albee A, Battel S, Brace R, Burdick G, Burr P, Casani J, et al. Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions. Pasadena, CA, USA: JPL Special Review Board; 2000.
[40] Tolker-Nielsen T. Exomars 2016 - Schiaparelli Anomaly Inquiry. Paris, France2017.
[41] Marine Accident Investigation Branch. Sbs Typhoon Contact in Aberdeen Harbour, 26 February 2011. 2011.
[42] IEEE. Ieee Std 1633-2016: Ieee Recommended Practice on Software Reliability. New York, NY, USA: Institute of Electrical and Electronics Engineers Reliability Society; 2016. pp. 1-261.
[43] Guarro SB, Yau MK, Motamed M. Development of Tools for Safety Analysis of Control Software in Advanced Reactors. Washington DC: ASCA Inc.; 1996. pp. 1-115.
[44] Al-Dabbagh AW. Dynamic Flowgraph Methodology for Reliability Modelling of Networked Control Systems. Oshawa, ON, Canada: University of Ontario Institute of Technology; 2009.
[45] Al-Dabbagh AW, Lu L. Reliability Modeling of Networked Control Systems Using Dynamic Flowgraph Methodology. Reliability Engineering & System Safety. 2010;95: pp. 1202-1209.
[46] Stamatelatos M, Dezfuli H, Apostolakis G, Everline CJ, Guarro SB, Mathias D, et al. Probabilistic Risk Assessment Procedures Guide for Nasa Managers and Practitioners. Washington D.C.: National Aeronautics and Space Administration; 2011. pp. 1-431.
[47] Guarro SB, Yau MK, Dixon S. Context-Based Software Risk Model (Csrm) Application Guide. 1st Ed. ed. Washington, D.C. 20546: ASCA Inc.; 2013. pp. 1-73.
[48] Aldemir T, Guarro SB, Kirschenbaum J, Mandellil D, Mangan LA, Bucci P, et al. A Benchmark Implementation of Two Dynamic Methodologies for the Reliability Modeling of Digital Instrumentation and Control Systems. Washington, DC, USA: Office of Nuclear Regulatory Research; 2009.
[49] Aldemir T, Guarro SB, Mandelli D, Kirschenbaum J, Mangan LA, Bucci P, et al. Probabilistic Risk Assessment Modeling of Digital Instrumentation and Control Systems Using Two Dynamic Methodologies. Reliability Engineering & System Safety. 2010;95: pp. 1011-1039.
[50] Wei YY. A Study of Software Input Failure Propagation Mechanisms. College Park, MD: University of Maryland; 2006.
[51] Zhu D. Integrating Software Behavior into Dynamic Probabilistic Risk Assessment. Collage Park, MD: University of Maryland; 2005.
[52] Leveson NG. A New Accident Model for Engineering Safer Systems. Safety Science. 2004;42: pp. 237-270.
[53] 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.

[54] 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.
[55] Abdulkhaleq A, Wagner S, Leveson N. A Comprehensive Safety Engineering Approach for Software-Intensive Systems Based on Stpa. Procedia Engineering. 2015. pp. 2-11.
[56] Rokseth B, Utne IB, Vinnem JE. A Systems Approach to Risk Analysis of Maritime Operations. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability. 2017: pp. 53-68.
[57] AAWA. Remote and Autonomous Ships - the Next Steps. In: Laurinen M, editor. Advanced Autonomous Waterborne Applications. London, UK 2016. pp. 56-73.
[58] Bureau Veritas. Guidelines for Autonomous Shipping. Paris, France2017. pp. 5-25.
[59] Gran BA. The Use of Bayesian Belief Networks for Combining Disparate Sources of Information in the Safety Assessment of Software Based Systems. Trondheim, Norway: NTNU - Norwegian University of Science and Technology; 2002.
[60] 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.
[61] Sadiq M, Ahmad MW, Rahmani MKI, Jung S. Software Risk Assessment and Evaluation Process (Sraep) Using Model Based Approach. ICNIT 2010 - 2010 International Conference on Networking and Information Technology 2010. pp. 171-177.
[62] 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.
[63] Tumer I, Smidts C. Integrated Design-Stage Failure Analysis of Software-Driven Hardware Systems. IEEE Transactions on Computers. 2011;60: pp. 1072-1084.
[64] 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.
[65] Li B, Li M, Smidts C. Integrating Software into Pra: A Test-Based Approach. Risk Analysis. 2005;25: pp. 1061-1077.
[66] Authen S, Björkman K, Holmberg J-E, Larsson J. Guidelines for Reliability Analysis of Digital Systems in Psa Context — Phase 1 Status Report. Roskilde, Denmark: Nordik Nuclear Safety Research (NKS); 2010. pp. 1-23.
[67] Authen S, Holmberg J-E. Reliability Analysis of Digital Systems in a Probabilistic Risk Analysis for Nuclear Power Plants. Nuclear Engineering and Technology. 2012;44: pp. 471-482.
[68] Authen S, Holmberg J-E. Guidelines for Reliability Analysis of Digital Systems in Psa Context — Phase 3 Status Report. Roskilde Denmark: Nordic nuclear safety research (NKS); 2013. pp. 3-37.
[69] Holmberg J-E, Authen S, Amri A. Development of Best Practice Guidelines on Failure Modes Taxonomy for Reliability Assessment of Digital Ic Systems for Psa. 11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference, PSAM11 ESREL 2012. Helsinki, Finland: Probablistic Safety Assessment and Management (IAPSAM); 2012. pp. 1887-1894.
[70] Huang B, Zhang H, Lu M. Software Fmea Approach Based on Failure Modes Database. 8th International Conference on Reliability, Maintainability and Safety. 2009. pp. 749-753.
[71] Ozarin NW, Siracusa M. A Process for Failure Modes and Effects Analysis of Computer Software. Proceedings of the Annual Reliability and Maintainability Symposium. 2003. pp. 365-370.
[72] Ozarin NW. Applying Software Failure Modes and Effects Analysis to Interfaces. Annual Reliability and Maintainability Symposium 2009. pp. 533-538.
[73] ISO. Iso 26262-11: Road Vehicles - Functional Safety. Part 11: Guidelines on application of ISO 26262 to semiconductors. Geneva, Switzerland: International Organization for Standardization; 2018.
[74] EN. En14514: Space Engineering Standards - Functional Analysis. Brussels, Belgium: European Committee for Standardization; 2004.
[75] IEEE. Ieee 830: Recommended Practice for Software Requirements Specification. New York, NY, USA: Institute of Electrical and Electronics Engineers; 2009.
[76] Myklebust T, Stålhane T, Hanssen GK. Important Considerations When Applying Other Models Than the Waterfall/V-Model When Developing Software According to Iec 61508 or En 50128. ISSC symposium. San Diego2015.

[77] 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.
[78] 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.
[79] 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.
[80] Next Gen IMR. Next Generation Inspection, Maintenance, and Repair. 2018; https://www.ntnu.edu/oceans/nextgenimr; Accessed: 05.04.2019
[81] 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 Commission; 2010.
[82] IEC. Iec 61508-4: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. Part 4: Definitions and Abbreviations. Geneva, Switzerland: International Electrotechnical Commission; 2010.