Journal of Marine Science and Technology (2024) 29:620–640
https://doi.org/10.1007/s00773-024-01008-0

ORIGINAL ARTICLE

Integrating scenario- and contract-based verification for automated vessels

Georg Hake[1] · **David Reiher**[2] · **Jan Mentjes**[1] · **Axel Hahn**[1,2]

Received: 5 March 2023 / Accepted: 23 May 2024 / Published online: 27 June 2024

Abstract

Scenario-based verification defines the current state of the art for examining a vessel's control systems for reliability and safety. However, software updates after release can only be covered to a limited extent. To take changes to a deployed system into account, the design and test phase must be harmonized with the operational phase. For all phases, regulatory, technical and safety requirements provide the scope to which the development process and the scenario-based tests need to be aligned and whose specifications the System under Test (SuT) must adhere to during operation. For this reason, a procedure is needed that converts the requirements into a format that can be utilized across all phases and measured in a structured manner comparing the original system to the updated version. This work does so by combining scenario-based verification methods with formal composition and monitoring techniques based on contract-based design into an integrated development approach. It is shown how safety requirements can be transferred into a Verification Descriptor that in turn provides the foundation for the division into model-based system development, contract-based virtual integration testing and a scenario-based test environment. For the entire lifecycle of the System under Test (SuT) to be included, the extended scenario and contract descriptors are carried forward up to the operational phase, such that the previously defined properties of the SuT can be monitored and validated during runtime. The approach is designed alongside a minimal-viable system and evaluated on an actual implementation of a safety-critical maritime LiDAR-based positioning system.

Keywords Scenario-based verification · Reliability · Assumption-guarantee contracts · Verification descriptor · Updates · Safety · Maritime autonomous surface ships (MASS)

Abbreviations

A/G	Assumption/Guarantee
ADAS	Advanced driver system
AI	Artificial intelligence
BAS	Berthing assistant system
BSA	Berthing support area
BVA	Boundary value analysis
CD	Continuous delivery
CI	Continuous integration
CPS	Cyber physical system
COLREG	Collision avoidance regulations
DevOps	Development and operations
EARS	Easy approach to requirements syntax
EPM	Equivalence partitioning method
GNSS	Global navigation satellite systems
GUI	Graphical user interface
HIL	Hardware in the loop
INS	Integrated navigation system
LiDAR	Light detection and ranging
MASS	Maritime autonomous surface ship
ML	Machine learning
ODD	Operational design domain
OS	Ownship
SIL	Software in the loop
SoS	System of systems

✉ Georg Hake
georg.hake@dlr.de

David Reiher
david.reiher@uol.de

Jan Mentjes
jan.mentjes@dlr.de

Axel Hahn
axel.hahn@dlr.de; axel.hahn@uol.de

[1] Institute of Systems Engineering for Future Mobility, German Aerospace Center (DLR), Oldenburg, Lower Saxony, Germany

[2] Department of Computing Science, Carl von Ossietzky University of Oldenburg (UOL), Oldenburg, Lower Saxony, Germany

SuT	System under test
TS	Target ship
VD	Verification descriptor
VIL	Vehicle in the loop
VIT	Virtual integration test

1 Introduction

The maritime domain is undergoing a transformation towards highly automated assistance systems and MASSs that take over safety-critical functions and support the operator on board as much as possible. The former Secretary-General and CEO of The Baltic and International Maritime Council (BIMCO) Angus Frew described this situation by stating that the "[...] industry has been living in a world of hardware. But software has been integrated into most physical equipment on the vessels, and the systems and procedures to manage the software have not kept up with technical developments, and it creates problems". The results were accidents on the high seas where ships suffered complete system failures due to a faulty update of software components [1].

Unlike in the traditional case, the built-in software components are subject to new requirements and at the same time offer opportunities for new test procedures that also extend into the operation phase. For this reason, supplier companies [2] and the International Association of Classification Societies (IACS) [3] developed standards and proposals for the integration and maintenance of software components on ships, which resulted in the ISO 24060 [4] standard for system monitoring in 2021 and was extended by the IACS Unified Requirements on the cyber resilience of ships in April 2022 [5, 6], which is applied to all new ship constructions after January 2024.

In summary, three key points emerge for the software-based system verification of maritime transportation systems:

- The software components used are subject to constant change and continuous further development and must also be checked again for safe behavior after an update following the initial release.
- For novel automated and autonomous navigation functions, no classical certification procedures exist yet, so that additional qualification stages [7] must be passed through before individual components can be certified and released.
- The behavior of Artificial Intelligence (AI) and Machine Learning (ML)-based components cannot be described deterministically [8]. Similarly, unexpected situations to which the target system is exposed during the operational phase cannot be fully mapped at design time. Therefore, unknown environmental factors, impacts, and encounter situations must be considered.

Various approaches for safety assessment of automated vessels take these factors into account [9], including (i) traffic simulation-based safety assessment approaches, (ii) staged introduction of autonomous vehicles, (iii) shadow modes, (iv) formal verification, (v) function-based approaches, (vi) real-world testing and (vii) scenario-based testing. In their survey on scenario-based safety assessment, Riedmaier et al. [9] conclude that a combined approach of scenario-based methods and formal verification compensates the drawbacks of individual approaches to demonstrate the safety of a SuT.

In this work an integrated approach on scenario-based methods with a formal verification approach based on Assumption/Guarantee (A/G) contracts is presented. A continuous test procedure is introduced that supports scenario-based test techniques along all phases of the Development and Operations (DevOps)-cycle as depicted in Fig. 1, such that the entire development and verification process does not have to be repeated in its entirety when changes to the SuT occur.

Continuous improvement of system components along the DevOps-cycle has so far found only limited application for safety-critical system components since Continuous Integration (CI) and Continuous Delivery (CD) pipelines conflict with the rather static safety requirements of a system. Safety standards such as ISO 26262 [13] require proof of a comprehensive safety case before deployment. Not only the safety of an updated module must be proven, but also the maintenance of the safety properties of the entire system, because safety is an overall system property that also affects the hardware and mechanical parts on board and can therefore not be considered in isolation. Nevertheless, it is argued here that software-driven continuous improvement moves increasingly into the development of safety-critical system components in the maritime as well as other transportation domains and provides means for extending safety measures with continuous monitoring and remote diagnostics of relevant security parameters and requirements during system operation. This allows for periodic reviews by approval authorities confirming the validity of novel technology. Therefore, it becomes necessary to align the requirements of existing safety standards with the development principles of the DevOps-cycle [14, 15]. Thus, the principles of the DevOps-lifecycle can and need also be applied as additional safety-measure to safety-critical systems.

As depicted in Fig. 1, the entire lifecycle of the SuT starts with an extended scenario description complemented by contracts, which is continued up to runtime, so that the previously defined properties can also be measured during operation. The goal is to transfer the simulation and

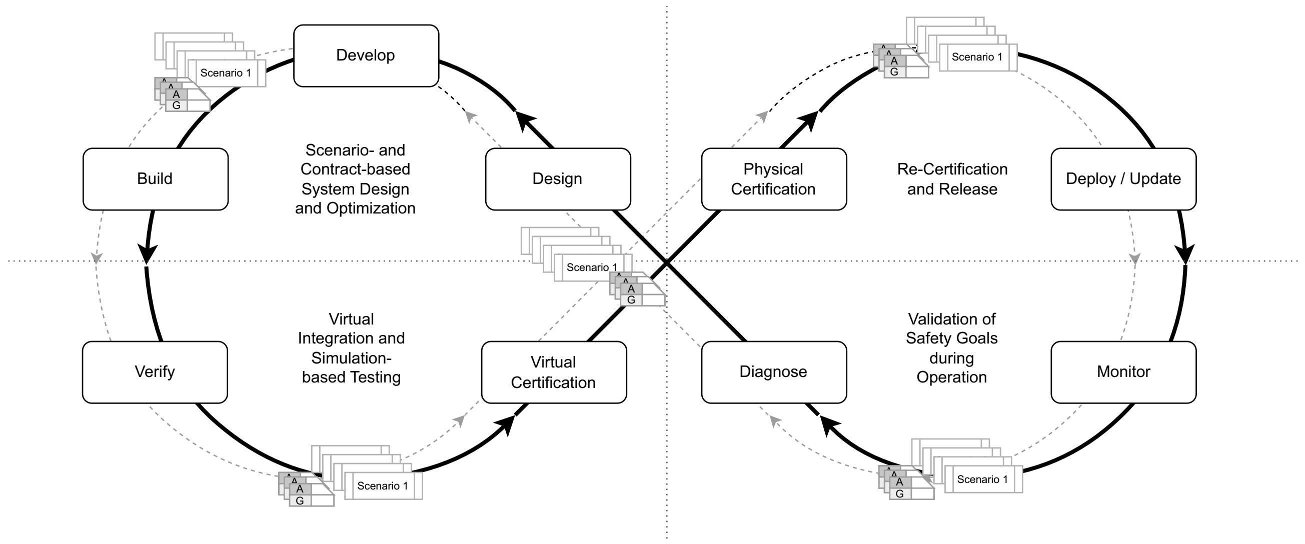
Develop
Design
Build
Verify
Virtual Certification
Physical Certification
Deploy / Update
Monitor
Diagnose
Scenario- and Contract-based System Design and Optimization
Virtual Integration and Simulation-based Testing
Re-Certification and Release
Validation of Safety Goals during Operation
Scenario 1
A
G

Fig. 1 A Verification Approach across all Phases of the Continuous Engineering DevOps-cycle (Adapted from [10–12])

scenario-based verification to the operations phase using the contracts, so that the insights from both phases can be delivered to the subsequent phase and back again. As depicted, the Continuous Engineering DevOps-cycle is extended to allow for continuous scenario-based testing at each phase of the overall process. Based on the functional and safety requirements identified during design time, test scenarios (depicted as scenario catalog at each phase) and contract-based monitors (depicted as A/G-contracts) are derived that can be carried forward to the operational and service phase and automatically applied for validation during runtime. The monitoring results form the basis for new features, requirements, and subsequently new or altered test cases used during subsequent design and development phases. In the following, related work to the process is presented and its phases are described subsequently in detail.

2 Related work

This work combines scenario- and contract-based verification approaches and demonstrates their integration into the lifecycle certification of maritime software-systems. Therefore, related work covers four areas of interest: systematic transformation and application of system requirements into a verification process, scenario-based test approaches, contract-based requirement proof, and integrated approaches of the former.

The first building block, the systematic transformation of system requirements, allows to derive a structured format of the inherently imprecise requirements and regulations written in natural language that serve as input for the framework presented. The Easy Approach to Requirements Syntax (EARS) presents a ruleset to address the ambiguous, complex, and vague requirements written in natural language [16]. Similar to the EARS ruleset, [17] developed parameterized safety description templates to structure safety requirements of embedded software systems. A further requirement management technique is presented by [18] that analyzes the activities to be performed, the tools to be applied, and the schemes used in the requirement engineering of safety-critical systems.

The second area of related work covers scenario-based test approaches. Here, the linking of safety requirements in the simulation runs of selected scenarios was presented by [19]. There, the requirements are made verifiable with respect to their safety properties based on defined environmental situations, thus making the environment statistically evaluable in relation to the SuT. Furthermore, the formal and mathematical basics of simulation runs are shown in [20]. The authors include techniques such as simulation units, behavior traces, and time testing. Finally, in [21] fundamental considerations regarding scenario-based testing, necessary to obtain a coherent safety-argumentation for the high-level safety requirements, are derived.

For the third area of interest, contract-based development, a contract-based lifecycle approach for the automotive domain is presented by [22]. Safety-critical updates are highlighted, and the demonstrator UPDATER is presented and evaluated based on an Advanced Driver System (ADAS). The subdivision of the development phase and operational deployment is also made by the authors in [23]. Here, the focus of the purpose is to negotiate the update between the system in operation and the update deployment considering multiple viewpoints. A possible way to use contracts for monitoring of ADAS is presented by [24] and integrated

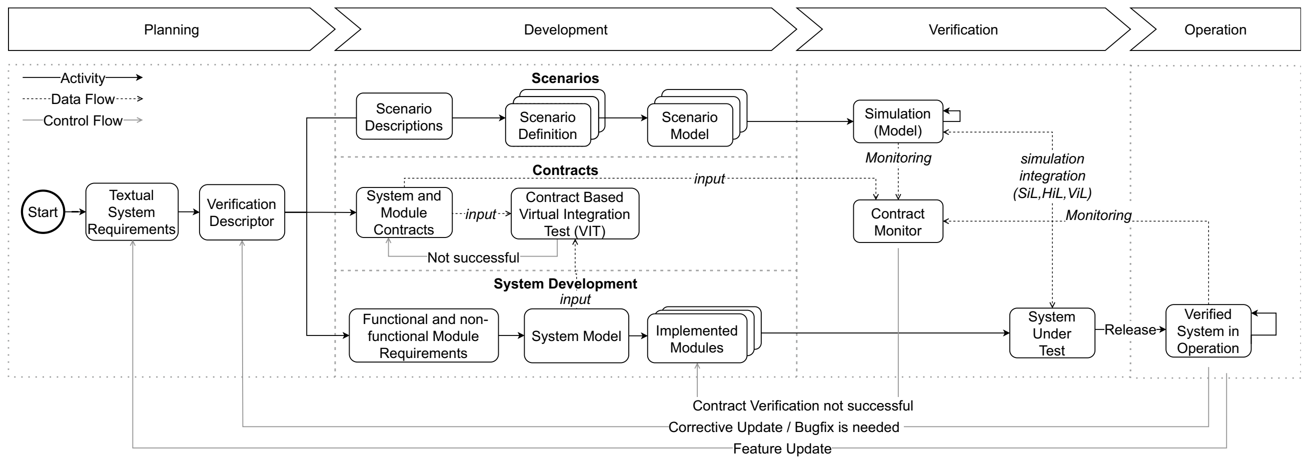

Fig. 2 The Overall Process Integrating Contracts and Scenario-based Verification

into a development process as well as illuminated from an industrial point of view.

Lastly, the integration of the previously presented building blocks of this work has been partially applied in an integrated approach of scenario-based modeling and contract-based verification in [25]. The proposed methodology is based on a sequential sequence, in which the environment and basic scenarios are modeled first, and the contracts are developed based on the functionality to be achieved within the derived scenarios. Finally, the controllers are implemented from the contracts and verified using Scenic, a domain-specific language for describing scenarios [26]. Their methodology differentiates from the approach in this work, such that it assumes the scenarios to be known before the system- and contract development, which is paralleled in this work. Nuzzo et al. present with CHASE [27] another integrated approach that transfers requirements from natural language into contracts and into a framework that enables the validation of high-level system specifications of Cyber Physical System (CPS). In their approach, requirements for legal environment configurations and scenarios are derived from the assumptions of the system components. Finally, in [28], the authors present a decomposition strategy for safety validation by introducing a simulation-based approach with error injection for requirements analysis in the form of assumption-guarantee arguments.

This work extends the approaches presented above by expanding the individual verification methods into an overarching process perspective that comprises the advantages of formal specification such as scenario space coverage, corner case identification, parameter compatibility and statement reliability, with the strengths of scenario-based approaches including black-box compatibility, system applicability, scenario representativeness and assessment transferability [9]. In the following, the integrated process is introduced and each phase is described along a minimal example.

3 Integrating scenario-based verification with contract-enhanced system development

The overarching development process as depicted in Fig. 2 is divided into the phases “Planning”, “Development”, “Verification”, and "Operation".

The process deviates from a traditional development process in that it introduces a Verification Descriptor (VD), the division of the development phase into system development, contract development, and scenario development as parallel and complementary lanes along the development phase as well as the integration of feedback loops, which return the scenarios, the model, the software artifacts, and the contracts to the previous phases. The process starts with the transformation of the textual requirements into the VD. The derivation for this is presented in the first part of the work. Based on the VD, the scenarios are extracted in the development phase, the contracts are formed and the model-based system development is started. The contracts are then transferred to a contract monitor, which uses the simulation model based on the scenario instances to check the SuT for the contractually guaranteed properties. Finally, the release of the system into the operational phase follows, for which the monitors from the verification phase are reused. If a contractual breach is detected in the verification or operational phase, a warning or safety measure is triggered by the contract monitor, which results in the implemented modules having to be revised. If this happens after the release, a revision of the VD is necessary, because the originally set requirements may no longer hold. If the system receives a new feature by means of an update after release, a revision of the original requirements is necessary. It should be noted that, as shown in Fig. 1, the existing contracts and scenarios can be reused and do not have to be completely

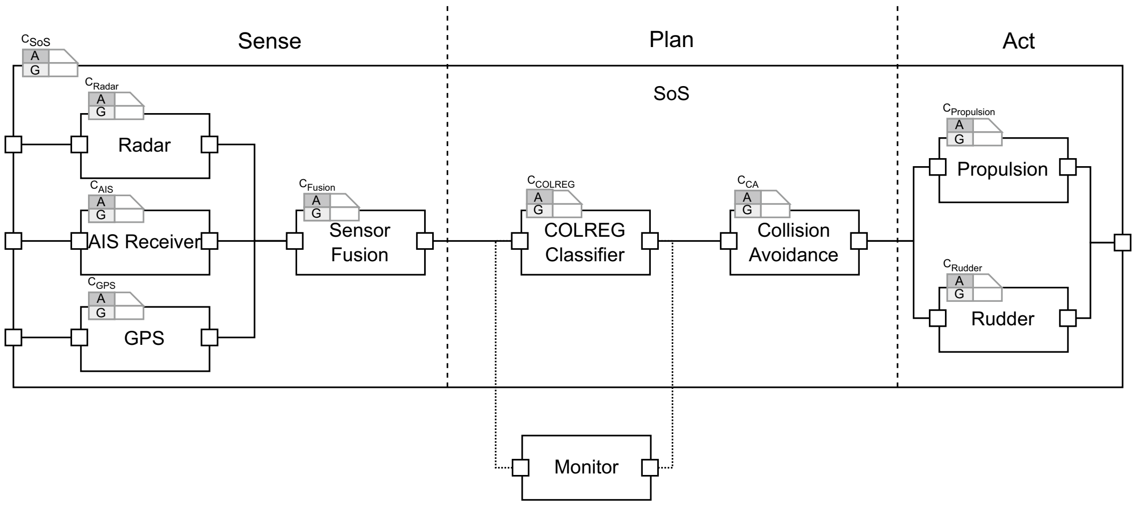

Fig. 3 COLREG-Classifier as part of a contractually enhanced System of Systems (SoS) with a Monitor attached to its Interfaces

renewed. By using the VD, it is also possible to narrow down which parts of the development phase need to be adapted and, if necessary, repeated, so that the renewed development, verification, and certification effort is significantly reduced.

This is illustrated in further detail in the following for each of the respective phases using a minimal viable example of a collision avoidance system for maritime Head-On situations. According to the Collision Avoidance Regulations (COLREG) a Head-On situation exists if two power-driven vessels meet on an opposite course [29]. If there is a risk of a collision, both ships must alter their course to starboard [30]. The SuT recognizes for an Ownship (OS) if it enters a Head-On situation and initiates the collision avoidance progress, which in turn calculates a rudder turn. We investigate the module of the SuT which classifies the encounter situation, called COLREG-Classifier. The COLREG-Classifier module is depicted in Fig. 3 as part of a System of Systems (SoS) connected to the sensors and the actuators on board. An additional module for the monitoring is attached to the interfaces of the COLREG-Classifier to validate its functionality during operation.

Furthermore, on a technical level the situation that the COLREG-Classifier is supposed to detect is depicted in Fig. 4 and defined as follows:

- An encounter situation between the OS and the Target Ship (TS) can be classified as Head-On when two or more ships are moving within a close distance (up to two nautical miles) towards each other. Among other things, collision avoidance systems on board can use this information to initiate an evasive maneuver that safely resolves the situation.

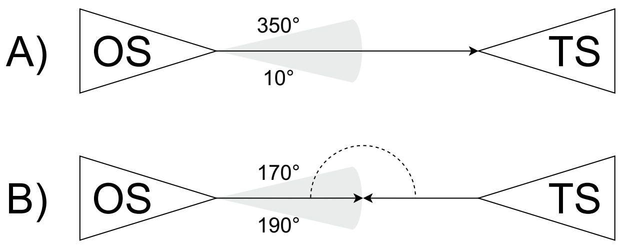

Fig. 4 Classification of Head-On situations based on bearing (a) and course difference (b)

- A head-on encounter situation exists exactly when the TS and OS have a course difference of 170° and 190° to each other and the bearing from the OS to the target ship is between 350° and 10°.

This minimal example is used below to show how the phases of the development model presented here support the development and verification of that SuT along its lifecycle.

3.1 Phase 1: planning

The overall process begins with the planning phase, this involves describing the requirements for the target system in an unorganized manner, usually in the form of long text documents and prose. To be able to further process the requirements in the following phases in a well-structured way, it is necessary as an intermediate step to transfer the textual form of the requirements into an evaluable form. The requirements are oriented at the definitions in [31] and bundled into a VD. It allows the transfer of the collected requirements to the subsequent phases and the subdivision into scenario and simulation-based verification, formal verification by means

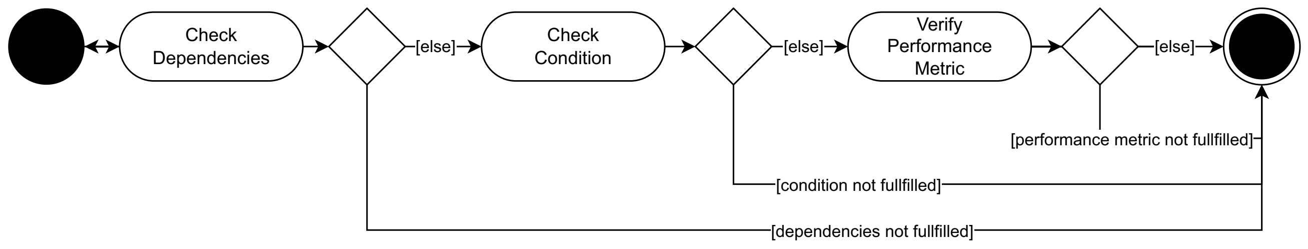
Check Dependencies
[else]
Check Condition
[else]
Verify Performance Metric
[else]
[performance metric not fullfilled]
[condition not fullfilled]
[dependencies not fullfilled]

Fig. 5 Activity diagram for checking verification descriptors

of contracts, and the model-based development of the system or update. The VD thus represents the basis for the further procedure, which is introduced in more detail in the following. It makes it possible to capture the requirements and structure them in such a way that they can be integrated by the scenario description logic and transferred to contracts of the System under Test (SuT).

3.2 Verification descriptor

The ISO/IEC/IEEE standard 29148-2018 defines the term requirement, as well as the processes needed to manage them [31]. It defines requirements as a statement that expresses a need and the associated constraints and conditions. Requirements can be formulated in natural language for that the standard states examples of relevant elements in their corresponding order. Elements include a subject (e.g., the system's name) and the action to be performed. Optionally, a measurable condition and constraints for the action can be stated. According to the standard, conditions provide attributes that permit a requirement to be verified and validated [31]. To evaluate the quality of requirements, the ISO standard defines characteristics and quality criteria. Individual requirements should be "*necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, correct and conforming*". A set of requirements should be "*complete, consistent, feasible, comprehensible and able to be validated*" [31].

Based on the requirement definition of the standard, the VD is derived. It is focused specifically on the parts of the requirements that are necessary for the verification and monitoring of a system. It can thus be defined as a subset of the requirements. A VD is formed based on the requirements by analyzing them for their attributes. The basic elements are the descriptor-id (I), requirements-id (II), parameter (III), conditions (IV), dependencies to other VDs (V), and acceptance criterion (VI). A VD is uniquely identifiable by its own id, while the requirements-id refers to the requirement it was derived from. This enables traceability during the verification process as each VD and derived test cases can be traced back to their corresponding requirements. Conditions describe under which circumstances a requirement is valid and are expressed as mathematical conditions considered the parameter. The acceptance criterion is interpreted as the output of the system and is verified if the condition is true.

In accordance with the quality criteria of the ISO standard, the criteria for the VD are defined. They must be necessary and unique. They are necessary if they contribute to the testing of a requirement and are unique if their elements do not occur in other VDs (e.g., duplications of conditions). If this is not the case, they need to be decomposed (e.g., split into atomic ones) and linked as dependencies to avoid duplications. A set of VDs must be complete, able to be validated, and comprehensible. The set is complete if all requirements and their properties have been mapped. Validity ensures that measurable conditions and no circular dependencies exist (not testable, since dependencies cannot be resolved). Comprehensibility requires stating the requirements-id to show the basis from which the VD was derived.

According to the study conducted in [32], templates are used in many cases for the specification of requirements in the industry. Requirements in natural language are easier to understand by stakeholders and therefore more accessible [33]. Thus, the basis of the VD forms textual requirements which are present by means of templates in a consistent structure. Many templates were already presented for different use cases [16, 17, 27]. However, these usually represent only one form of requirements within a requirements specification (e.g., the definition of safety requirements), whereas all system requirements should be considered for system development, scenario- and contract-generation. We base the requirements in this work on [34], in which general requirement templates are defined. A requirement essentially consists of a condition and the main clause describing the desired function. For the VD, the requirement condition is successively translated into a mathematical expression and the main clause is used to derive the acceptance criterion.

Once VDs are derived and tests are executed, they are used to verify requirements. For each requirement and derived VD, the following activities (shown in Fig. 5) are performed for the verification.

For each VD, it is first checked whether the dependencies (including conditions of the dependencies) are satisfied. Then, the condition of the VD is checked. Finally, the acceptance criterion is checked and compliance with the requirements is verified. In all other cases, the VD does not apply, so the acceptance criterion is not checked. If the conditions or dependencies are not defined within a VD, they are automatically considered satisfied.

For the exemplary detection of Head-On situations, a small set of requirements is defined, describing when an encounter situation is given and under which condition the system should classify a situation as “Head-On”:

1. The COLREG-Classifier must be able to detect encounter situations.
    a. If the distance between an OS and TS is less than two nautical miles an encounter situation exists.
2. The COLREG-Classifier must be able to classify Head-On encounter situations.
    a. If an encounter situation exists, and the course difference is between 170° and 190°, and the bearing is between 350° and 10° then the COLREG-Classifier must classify the encounter situation as "Head-On".
3. The COLREG-Classifier must be able to classify a situation in a maximum of 10ms.
4. The COLREG-Classifier must be able to process continuous data input for at least 10 s with a maximum delay of 1 s between two data sets.

For the example provided, two high-level requirements describe that the system must be able to detect encounter situations and classify them accordingly. The sub-requirements concretize these and describe conditions under which they are applied. Requirements 3 and 4 describe performance obligations, where 3 defines the maximum processing time and 4 defines the input duration and delay between datasets. Following these requirements, VDs are derived. Based on the requirements condition the parameters and conditions for the VDs are extracted. The acceptance criterion is derived based on the systems output (e.g. “Head-On“).

As depicted in Table 1, for requirement 1.a a single VD is derived, describing under which conditions an encounter situation exists. If condition (C) is true, the system’s output cannot be null (AC). For requirement 2.a two VDs are derived, as two parameters are needed to classify a situation as “Head-On”. VD 2.1 and 2.2 define the conditions for Head-On classification, based on angle difference and bearing. They are linked by the dependency between them. In addition, the dependency on VD 1.1 specifies that the classification can only be performed if an encounter situation is present. An additional non-functional requirement defines temporal constraints on the processing time. Thus, VD 3 ensures that the processing time is less or equal than 10ms if an encounter situation is present. Finally, VD 4 defines how long a continuous input stream is present and how fast data is received by the system. As this is not bound to the existence of encounter situations, the acceptance criterion is that the output cannot be empty.

3.3 Phase 2: development

The second phase is divided into three parallel processes that are interdependent and interrelated: scenario development, formalization of the requirements by means of contracts, and the actual module or system development.

In the following, a closer look at these parallel executed threads will be taken and the activities and artifacts they contain will be outlined.

3.4 Scenario development

To be able to obtain reliable results in a reasonable time frame in the verification phase, following the development phase, a targeted approach is necessary. The approach of scenario-based testing currently sets the current state of the art for examining a SuT for reliability and safety [35–38] - especially for modern non-deterministic assistance systems - and is expected to replace classical static methods in part [39]. That is to identify, model, simulate, and evaluate traffic scenarios that are relevant for the significance of simulation results with respect to a specific SuT.

When creating scenarios, a distinction is made between functional, logical and concrete scenarios [40]. Functional

Table 1 Derived verification descriptors for COLREG-Classifier requirements

Req-ID	VD-ID	Parameter (P) & Condition (C)	Dependency (D)	Acceptance Criterion (AC)
1.a	1.1	$0m \leq Distance_{OS,TS} \leq 3704m$	–	$Output \neq NULL$
2.a	2.1	$170° \leq CourseDifference_{OS,TS} \leq 190°$	–	–
2.a	2.2	$350° \leq Bearing_{OS,TS} \leq 10°$	1.1	Output = ”Head-On”
3	3	–	2.2	$CalculationTime \leq 10ms$
4	4	$InputLatency \geq 1\,s \wedge InputDuration \geq 10\,s$	–	$Output \neq NULL$

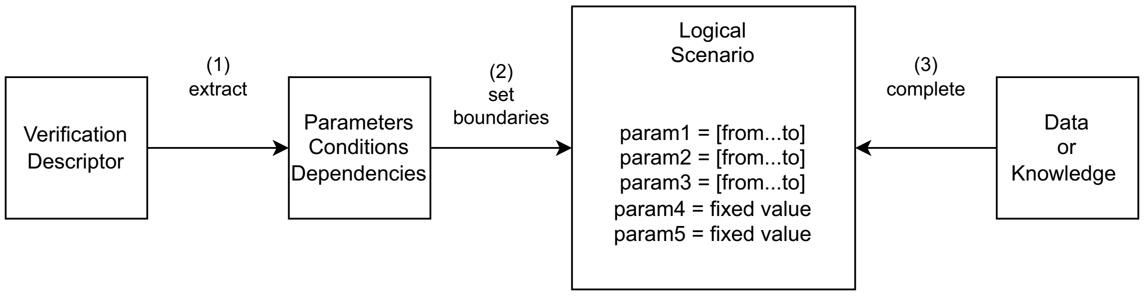

Fig. 6 Overview of the process to derive complete logical scenario space descriptions from the verification descriptors

scenarios are formulated linguistically and are typically used in the concept phase. Logical ones, on the other hand, describe a situation by parameters and boundary values for those parameters. Concrete scenarios are determined by selecting certain parameters based on the ranges defined by the limiting boundary values. As the VD is represented by several parameter conditions, the boundaries of the parameters are given, which in turn are transformed into logical scenarios. The use of those previously created VDs also leads to the creation of functional scenarios not being necessary at this point.

Since requirements describe the performance of a system, but do not necessarily describe how the test cases must be designed, a process is necessary to identify base scenarios. This can be data-driven (based on historical data sets) or expert-driven (e.g., expert knowledge, regulations). If not all necessary scenario parameters are covered by the requirements (because they are not relevant to the system), they can be filled by experts or historical data. For example, a system can be executed independently of the environment (sea area, infrastructure), but the environment must be defined for a complete and valid scenario definition.

Figure 6 outlines the process for the creation of logical scenarios. The necessary parameters, conditions, and dependencies are extracted from the VD as the first process step. In the second step, they are grouped to extract the necessary parameters for functional scenarios. These are supplemented by expert knowledge or historical data sets in the third and last process step. As a result, the scenario is represented by several parameters (e.g., speed, course, etc.) and their applicable value ranges. Since these are by then still logical scenarios described by parameter ranges and boundary values, they are not directly applicable for a simulation run and distinct parameters must be selected to obtain a set of concrete scenarios covering the given state space. Through a selection, combination, or sampling process (or a combination of those) parameters are chosen from the previously defined value ranges to create a concrete scenario for a single simulation run. Which approach is chosen here depends on the specific use case and SuT, one possible three-step process combining selection and combination is shown by Schuldt et. al [41]. The resulting set of concrete scenario descriptions is then transferred into a set of technical scenario model instances that serves as input for the used simulation system. The three steps described above form the process of moving from VDs to scenario models, which can serve as input for a simulation system. The individual steps and their respective input and output artifacts are once again listed in a structured manner in Table 2.

Similar to the example described at the beginning of this chapter - the COLREG-Classifier - the identified three steps are applied in the following to the procedure, inputs, and the outputs result.

3.4.1 Step 1: derive scenario descriptions

To obtain logical scenarios from the given verification descriptions, a method based on the process shown in Fig. 6 is necessary. For step (1), the relevant parameters can be read directly from the given VDs. Parameters are omitted that cannot be mapped in the scenario itself but are used for verification after the simulation run. This is the case here

Table 2 The three main process steps to derive concrete scenario models from a verification descriptor and their respective I/O artifacts

Step	Activity	Input	Output
1	Derive Scenario Descriptions	Verification Descriptor	Scenario Space Description (logical scenarios)
2	Parameter Value Selection	Scenario Space Description (logical scenarios)	Scenario Definitions (concrete scenarios)
3	Scenario Modeling	Scenario Definitions (concrete scenarios)	Scenario Models (simulatable scenarios)

Table 3 Relevant parameters portioned by equivalence

Parameter	Invalid	Valid	Invalid
Distance $_{OS,TS}$	–	0– 3704 m	≥ 3704 m
Course $_{OS,TS}$	≤ 170°	170° – 190°	≥ 190°
Bearing $_{OS,TS}$	≤ 350°	350° – 10°	≥ 10°

Table 4 Boundary values identified for the relevant parameters

Parameter	X (min)	X (min+)	X (nom)	X (max-)	X (max)
Distance $_{OS,TS}$	0 m	185.2 m	1852 m	3518.8 m	3704 m
Course $_{OS,TS}$	170°	171°	180°	189°	190°
Bearing $_{OS,TS}$	350°	351°	360° / 0°	9°	10°

with the *Calculation Time* parameter given by VD 3. The other three parameters can be taken as relevant: *Distance between OS and TS*, *Course Difference between OS and TS*, *Bearing of OS and TS*. Dependencies between those parameters relevant for the scenarios to simulate can then also be derived directly from the given VDs. In order to identify the relevant parameter spaces and their limit values for the identified parameters in step (2) of Fig. 2 "set boundaries", equivalent classes are first formed via the Equivalence Partitioning Method (EPM) [42] and then a Boundary Value Analysis (BVA) [43] is carried out. By that concrete parameter values can be deduced that have to be tested simulatively.

The results can be found in Table 3 and Table 4. Typically, at least three limit values are defined, X(min), X(nom), and X(max) representing the smallest, the biggest valid value, and the value lying exactly in the middle of the interval spanned by those to boundary values. In addition, values close to the limits are tested. Here, X(min+) and X(max-) were additionally defined. These lie in each case in the direct valid neighborhood of the limits of the parameter to be tested. They are each defined by an interval to be determined in advance and enable tests near the limits of a system. Here, the value that is at the 10% mark before the limit starting from X(nom) was selected in each case.

Now that the identified parameters and a set of values that are important for testing the given system have been identified, the remaining parameters that are necessary for a complete maritime traffic scenario need to be defined and filled in using expert knowledge or historical data (see Fig. 6, Step 3).

Here, for example, the location of the encounter situation, the type of ships, and the weather are important. For the minimal example considered, it is assumed that the weather does not play a role, and therefore the best possible weather could be used, i.e., no wind, infinite visibility, no waves, or currents. Thus, in the end, the weather can even be completely excluded. Since in the given minimal example only the collision risk of two ships is examined, excluding static infrastructure or land masses, the scenario is placed on a virtual open sea. Thus, this aspect can also be omitted and does not have to be explicitly parameterized or modeled. In terms of the types and dimensions of the two vessels participating in the scenario, the Evergreen A-class container ships will be used as a reference and a length of 400 m and a width of 61.5m will be assumed for both vessels. The resulting logical scenario is depicted in Table 6, the relevant parameter value selection is described in the following.

3.4.2 Step 2: parameter value selection

Since the previously created logical scenario still contains value ranges or, in the case shown here, value sets, these must be combined reasonably to obtain concrete scenarios with specific parameter assignments. The combinatorial test case generation according to Schuldt et al [41] can be used for this. There are four possible concrete methods for combining the parameters purely combinatorial and without the use of any semantic information: "each-used", "pair-wise", "t-wise" and "N-wise" [44]. For representational reasons - the methods t-wise and N-wise would require a graphical representation with 3 or more dimensions - the pair-wise method is used and shown here.

The result can be seen in Table 5. Each cell represents a combination of two parameters; the representation as a two-dimensional table is a prerequisite for the "pair-wise" combination method: Every assignment of a parameter is tested with every assignment of all other parameters. The parameters that are not specified by the cell combination must of course also be assigned. Different metrics can be used for this. At this point, the mean value X(nom) is always chosen. The cell marked with an “x” represents the concrete scenario shown in Table 6 as well as depicted in Fig. 7. The dark grey colored cells represent invalid combinations, in this case, combinations of different assignments of the same parameter. The concrete scenarios identified in this way still contain composite values such as the distance between two or more vessels to each other.

To make the scenarios easier to use and to prepare them for the next step, individual values for each traffic participant must be derived from these composite values. In the case shown here, this means that the parameter values Latitude, Longitude, Course, and Heading from the parameters Distance$_{OS,TS}$, Course Difference$_{OS,TS}$, and Bearing$_{OS,TS}$ for both TS and OS are determined. This results in the values in Table 7.

To also test the non-functional requirement 4 (see Sect. 3.2) a temporal progression must be introduced into the scenario. So far, only two ships are in a fixed position. This represents the one particular situation in which the output of the system under test is to be verified according to the

Table 5 Parameter combination table as the result of pairwise combinatorial test case generation

		Distance					Course Diff.					Bearing				
		0	182.2	1852	3518.8	3704	170	171	180	189	190	350	351	360	9	10
Distance	0	▨	▨	▨	▨	▨										
	182.2	▨	▨	▨	▨	▨										
	1852	▨	▨	▨	▨	▨										
	3518.8	▨	▨	▨	▨	▨										
	3704	▨	▨	▨	▨	▨	x									
Course Diff.	170						▨	▨	▨	▨	▨					
	171						▨	▨	▨	▨	▨					
	180						▨	▨	▨	▨	▨					
	189						▨	▨	▨	▨	▨					
	190						▨	▨	▨	▨	▨					
Bearing	350											▨	▨	▨	▨	▨
	351											▨	▨	▨	▨	▨
	360											▨	▨	▨	▨	▨
	9											▨	▨	▨	▨	▨
	10											▨	▨	▨	▨	▨

Table 6 Complete logical and derived concrete scenario

	Logical Scenario	Concrete Scenario
Name	Ownship (OS) / Targetship (TS)	Ownship (OS) / Targetship (TS)
Type	Container Ship	Container Ship
Length	400 m	400 m
Width	61.5 m	61.5 m
Distance	[0, 182.2, 1852, 3518.8, 3704]m	3704 m
Course Diff.	[170, 171, 180, 189, 190]°	170°
Bearing	[350, 351, 360, 9, 10]°	360°
Weather	Perfect (ignore)	Perfect (ignore)
Area	Open Sea (ignore)	Open Sea (ignore)

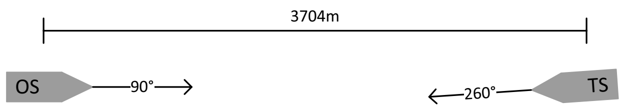

Fig. 7 The encounter situation from Table 6 illustrated graphically

identified parameter combinations to be tested. However, traffic is dynamic and usually moves continuously, which is also the aim of requirement 4. Therefore, a situation is sought which is temporally prior to the one just identified and from which it will be reached. This can be done based on recorded trajectories, average values, interpolations, or abstractions. For the minimal example shown, therefore additionally a parameter *speed* for both vessels is defined. Since the values are based on the Evergreen A-class vessels the speed will also be the average cruising speed of vessels of this class of 22.6kn [45]. For this minimal example, sailing on a straight-lined trajectory is sufficient and therefore the coordinates are shifted by 116.26m in the opposite direction to the heading so that both vessels need about 10 s to reach the situation to be tested.

3.4.3 Step 3: scenario modelling

Now that concrete scenarios have been identified and fully defined, they are to be reproduced within a simulation system. For this purpose, the traffic participants and their environment must be mapped onto the simulation model according to the values defined in the concrete scenario. How this can be achieved depends largely on the simulation system that is to be used. Some simulation systems only offer one simulation model and the situation to be simulated must therefore be defined in the simulation system itself. This is often done via a Graphical User Interface (GUI) and a subsequent run of the created scenario. Another approach is to consider the scenario model separated from the simulation model. In this case, a concrete scenario is created based on the possibilities of the scenario model. This can be an

Table 7 Concrete scenario with single concrete placement values for each ship

	Vessel 1	Vessel 2
Name	OS	TS
Type	Container Ship	Container Ship
Length	400 m	400 m
Width	61.5 m	61.5 m
Latitude	47.0	47.0
Longitude	−35.0	−34.9514208
Course	90°	260°
Heading	90°	260°
Weather	Perfect (ignore)	Perfect (ignore)
Area	Open Sea (ignore)	Open Sea (ignore)

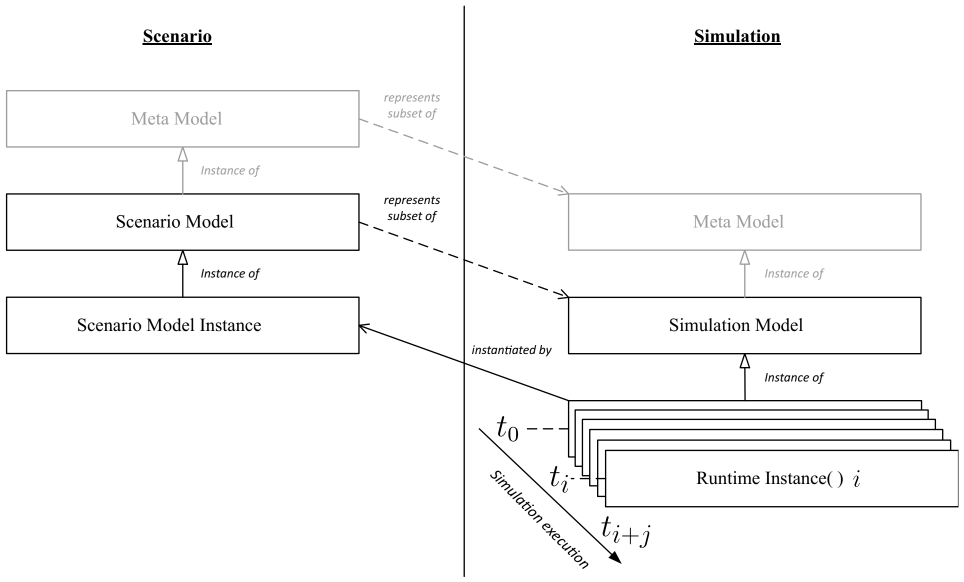

Fig. 8 From a Scenario Meta-Model to Simulation Runtime Instances (Source: [46])

XML file, for example, which describes and parametrizes concrete instances of elements from the given scenario model. These scenario instances can then usually be read in by the simulation system and serve as the basis for the simulation model. The components described in the scenario file are extracted from the possible simulation elements, instantiated, and filled with values accordingly. The result is the simulation model, i.e., a representation of the simulation contents, which can change over time during an active simulation run. These interrelationships and dependencies are shown on a high level in Fig. 8.

3.5 Contract development

The parameters in the VD are available in semi-structural form after extraction from the unstructured sets of rules and texts. To make the parameters, constraints, and dependencies contained in the VD automatically analyzable, it is necessary to convert them into a format for which formal methods of analysis exist that can be applied to the extracted values. Contract-based design is suitable for this purpose [25, 27].

Contracts are used in the design of safety and time-critical as well as component-based systems. Here, the linked embedded systems and/or software components are assigned contractual assurances about their behavior at their interfaces. This makes it possible to formally verify the interconnection on one, as well as on different hierarchy levels, and to uncover dependencies as well as temporal discrepancies in the overall system view.

Contracts consist of a pair of assumptions and guarantees. The guarantees specify the contractually assured properties that the component promises, provided that the assumptions it makes about its environment are met. In the case of INSs such as those used on modern ship bridges, for example, this means that the contracts express the behavior of the individual functions of the bridge and their interaction can be formally tested before the modules from different manufacturers are combined to form an overall system. For the COLREG-Classifier minimal example, the generated contract breaks down into assumptions and guarantees as shown in Table 8.

Table 8 The contract for the implementation and system integration of the COLREG classifier Module

ID 1.0	$Contract_{COLREG-Classifier}$					
Assumptions	Receive	Distance $_{OS,TS}$			every	$x \leq 1s$
		Course Difference $_{OS,TS}$				
		Bearing $_{OS,TS}$				
Guarantees	Whenever	Distance $_{OS,TS}$	$x \leq 3704\,m$	Output	within	$x \leq 10ms$
		Course Difference $_{OS,TS}$	$170° \leq x \leq 190°$			
		Bearing $_{OS,TS}$	$350° \leq x \leq 10°$			

Unlike in the VD, in the contract the minimal requirement for the functionality of the component that provides that function is defined. Moreover, the timing requirements for receiving the necessary parameters and the required time in which the output is generated are defined. The COLREG-Classifier as a SuT requires receiving the values for distance, course and bearing to the target ship from the ship's sensors every $x \leq 1s$ and guarantees to issue a "Head-On" signal within 10ms after receipt of the values, provided that the assumptions are fulfilled.

As a module component of a SoS system network, the COLREG-Classifier therefore expects input every second at its interface, evaluates them and outputs the result at its output interface. As can be seen in the minimal example of the COLREG-Classifier SuT shown in Fig. 3, the module receives its expected input from the sensor fusion system within the system network.

In this early stage of the design the static system verification, which is based exclusively on the contract model, referred to as Virtual Integration Test (VIT) (see Fig. 2), can be applied. For the VIT the compositional properties of the system are verified with respect to the system model developed in the design phase and the properties of the model components specified in the contract. For the model driven-development expressive modeling languages and tools, such as SysML, MathWorks Simulink, Ansys SCADE or AADL are regularly utilized [14] and the contracts are supplemented on a textual basis. Based on the extended models, tools such as MULTIC Tooling [10] or OCRA [47] enable evaluation of the modeling and the contracts.

In many cases, this is where the use of the contracts for system verification ends [48, 49]. In this work, the contractually defined properties are carried over to the simulative and operational phases, by transferring the assumptions and guarantees of the contracts to a monitor. This allows to validate the properties not only with respect to their compositional nature, but also simulatively and during operation based on the scenario space that has been identified in the simultaneously performed process steps, to determine whether the defined limits of the assumptions and guarantees are exceeded, or components lose performance or fail completely. In case of a faulty behavior the feedback loops can be triggered so that corrective action can be taken. In the following, it is described how this context is designed.

3.6 System development

The system development phase essentially follows the established procedure for the development of safety-critical applications on board of a vessel. To counteract risks, established standards are applied whose aim is to mitigate or avoid potential hazards already during system development. One of these established automotive standards is ISO 26262 [13], which revolves around the key concept of functional safety. A further standard that deals with the systematic development of electronic systems for maritime transport systems is ISO 17894 [50], which proposes a process model similar to the V-model in one of its annexes.

Due to the frequent application of this model in the maritime context, this work orientates itself on the traditional development process and adapts it to the requirements presented. As already presented in other related publications, the classical phases of the V-model are extended by stepwise V&V loops [39], so that a continuous V&V is achieved, which is in accordance with the overall model presented here and at the same time guarantees assurance of functional errors during the early phases of system development.

3.7 Phase 3: verification

One or more scenario models, a set of contracts and the realized parts of the system under test emerge from the development phase. These three building blocks enable simulation-based verification of the implemented software modules.

In the model-based simulation, the identified environmental parameters, participating and interacting systems and components as well as induced situations and error values can be checked using the output values of the simulation. Section 3.4 shows how the VDs can be used directly as a basis for a complete and valid set of scenarios. Moreover, Fig. 6 shows how the approach is applied and how the VDs serve as direct input for the scenario modelling. With this already relatively restrictive basis, the problem of elusive high-dimensional state spaces can be simplified to a large extent, as only scenarios that suit the requirements and constraints of the VDs are needed and modelled.

For the actual simulation run, the scenario model is transferred to a simulation model [51] and is merged with the SuT. The SuT is integrated into the simulated environment and takes part in the simulation execution. This is done via a communication channel between the SuT and the simulation system, sending and receiving relevant data every simulation time step. This way the SuT is a discrete part of the simulation, which makes it a sub-simulation and the simulation system a co-simulation. This way it is possible to verify that the contractually assured properties can be met. For this purpose, software as well as hardware and vehicle-in-the-loop (SIL,HIL & VIL) testing can be performed. The SuT can thus also be represented as a module of the vessel within a SoS. Furthermore, the interacting parts of the SoS do not necessarily all have to be already implemented or present on dedicated hardware but can also be present in the form of black-box models whose behavior can be simulated virtually. This is especially important to protect the intellectual property of the stakeholders and to be able to perform tests of the

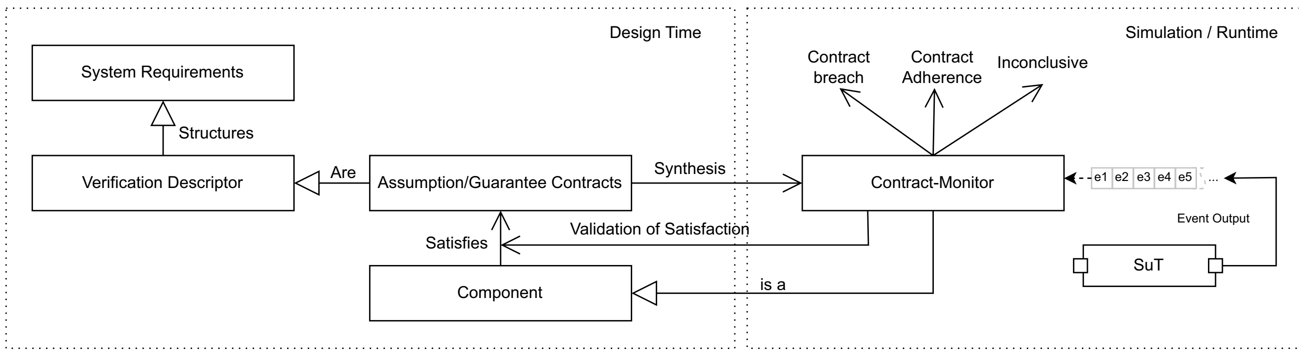

Fig. 9 From System Requirements to Contract Monitors (Adapted and modified from [52])

SuT module under investigation even before all components of a SoS to be integrated have been completed.

As shown in Fig. 9, monitors can be synthesized for the SuT's system modules based on the requirements specification at design time. These monitors then enable validation of compliance with the system specification (satisfaction), detection of ambiguous system states or results (inconclusiveness) or a violation of the specification (violation) at runtime. The system generates event chains that are picked up by the monitor and evaluated based on the requirements specification [52].

Since in this work the requirement specification is based on the VD and the contracts, the monitor synthesis results from the specifications in the contracts. The contract monitors are themselves components of the SoS within the simulation and verify compliance with the specifications of the contracts. As components within the SoS the monitors are connected to the input and output interfaces of the system modules and measure the message-based communication between the components. Thereby the contracts indicate the time between the arrival of a signal and the time of processing. In addition, value ranges can be defined by which the input and output values are allowed to fluctuate. The monitors use the assumptions and guarantees of the contracts to measure the module communication. This relationship is shown in Fig. 9 as well as Fig. 3. The verification of properties at run-time, in simulation and in actual operation, is also referred to as online monitoring and opens up possibilities for run-time verification as well as the initiation of mitigation strategies [10].

At simulation time, the simulation runs are observed with the scenarios as input from monitors resulting from the contracts. If a monitor detects a breach of the contractually secured safety properties, which were collected by the VD, a regression into the implementation phase of the involved modules takes place. The chronological sequence of the monitored contracts allows the process to be traced, thus facilitating troubleshooting.

Based on the contract for the COLREG-Classifier shown in Table 8, a monitor can be generated in the development phase. The monitor checks both the value ranges of the assumption and guarantee and their temporal processing of the incoming event and the output at the interfaces of the COLREG-Classifier module. Therefore, the predefined value ranges and time ranges of the contracts are observed when the simulation is executed. This process supplements the static VIT by additionally checking the realized software modules for their ability to implement the limit ranges that were previously checked against the system model.

3.8 Phase 4: deployment, operation and updates

Once the developed system or module has successfully passed all verification stages and has been approved, certified, and classified, it can be released and deployed. The system can enter the operational state. However, in accordance with the DevOps-cycle, verification and validation does not end here, but is continuously checked by monitoring of the properties specified in the contracts, similar to the simulation phase. Thus, the certification and classification of the transportation system on which the system is applied continues to depend on the ability to maintain in operation the properties previously promised and demonstrated in the test.

Since a test can never fully represent reality, especially in the case of novel software systems and automated or autonomous control functions, it is important to further detect deviations from specified functionality during operation. This includes the detection of trends before a threshold is exceeded, to identify load profiles, and to log individual component behavior as well as the interaction between components. The information gathered enables, in the event of a breach of contract, to pinpoint the modules involved and to reconstruct the corresponding environmental situation retrospectively.

In case of an update within a contractually defined system, three different deltas of change can be differentiated

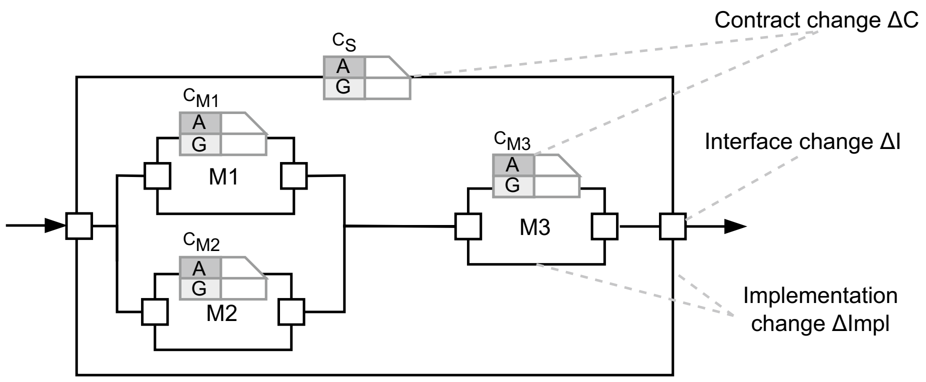

Fig. 10 Contract-based impact identification due to an update (Adapted and modified from [22])

(ΔC, ΔI, $\Delta Impl$) depending on the respective type of update (corrective, perfective and adaptive), which in turn determine the impact on the overall system. A corrective update includes bugfixes to a module, perfective updates comprise performance improvements and adaptive updates incorporate functional changes to the implementation. The different types of deltas are depicted in Fig. 10.

A contract change ΔC requires at the minimum a repetition of the VIT. An update that results in a delta of the interfaces ΔI requires a review of the modules connected to the affected module and dependent modules within the SoS, as well as an adjustment of the interface description by the contracts, if necessary. Finally, a delta of the implementation $\Delta Impl$ results in a complete revision of the functionality of the system [22].

Hence, to accommodate updates after release, two feedback loops are necessary during operation. Firstly, in case of a corrective or perfective update, it becomes necessary to return to the VD to reassess the systemized requirements. Secondly, an adaptive update would require the initial system design to be altered. Here, it is required to reevaluate whether the textual system requirements still cover the changed system functionality.

Both cases result in parts of the system being re-planned. Therefore, it is required to pass all the described development and verification steps again for the system to be re-verified. By linking requirements, scenarios and contracts with VDs the delta of the update can be narrowed down and the re-evaluation effort be minimized. Furthermore, via the IDs of the VD as well as the contracts, changes are traceable and the impact of the change can be traced bidirectionally. For each requirement, it is possible to pin down what has been derived based on it, so that the elements that are affected can be identified. Existing scenarios and contracts can be reused for a new DevOps run, which reduces the amount test runs that need to be repeated.

4 Evaluation: verification of an update of a safety critical maritime LiDAR-based berthing assistant

To evaluate the development process designed in this paper, it is applied to a safety-critical maritime Light Detection and Ranging (LiDAR)-based Berthing Assistant System (BAS). Thus, applying a concrete implementation of the developed process on a SuT. For this purpose, the SuT and its requirements are described. This forms the basis for deriving the VDs, which in turn are used to derive scenarios for the simulation as well as contracts for monitoring of runtime properties. Since requirement changes and system enhancements are part of the DevOps-cycle, it will be examined how the process behaves in the event of changes and how updates influence the scenario generation.

4.1 System under test

To evaluate the approach presented, a BAS based on LiDAR sensors, previously presented in [53], is utilized. This uses several LiDAR sensors, positioned at the harbor and connected with each other. Measurements of these are fused and made available to the nautical personal on ships in aggregated form, so that they can integrate further position data into their calculations in addition to their GPS data. The authors base their system on reference points in conjunction with a Berthing Support Area (BSA). Reference points are aligned at a quay's meter marks measuring distance, speed and acceleration values of an approaching object. The concept of reference points is based on 1D-LiDAR sensors, measuring a single distance. In contrast to physical LiDAR sensors, they can be placed virtually at any position, while working with 2D or 3D point data. Following the concept of the Operational Design Domain (ODD) of the automotive domain, the BSA defines an area in which the system offers support. It is derived based on a set of constraints, where the authors consider $P_{Control}$ (e.g., vessel angle of attack), $P_{Construction}$ (e.g. vessel hull) and $P_{Environment}$ (e.g. visibility, tide, wind) as well as the possible illumination of a target based on a LiDAR setup. Based on the concept and constraints of the BSA several requirements must be fulfilled to ensure that the system is working as expected in terms of precision and guarantees of the BSA. For the evaluation the following requirements refer to an implementation of the system for a specific setup:

1. The system must be able to offer support for a vessel within the Berthing Support Area.
    a. The system must measure the distance up to at least 120 m perpendicular to the quay wall.
    b. The system must measure the distance up to 120 m along the quay wall.
    c. The system must measure the distance for a vessel with at least 16 m length.
    d. The system must measure the distance for a vessel with at least 4.8m width.
    e. The system must provide measurements for a vessel within the BSA with a maximum angle 15°.
2. As long as a vessel is within/intersects the Berthing Support Area the reference points must provide distance measurements.
    a. If at least 5 pts are within a reference point the distance must be calculated.
    b. The system must report measurements with a delay below 200ms.
    c. The system must provide distance measurements with a precision of 0.1m.

For the BAS, two top-level requirements are derived, describing the desired functionality. Requirement 1 describes non-functional constraints for the system regarding the BSA. Sub-requirements describe more detailed constraints for these. The size of the BSA is 120 m in width and length. This is estimated by the authors based on P$_{Control}$ and P$_{Construction}$, considering the ships dimension and the maximum angle of attack. Only if a ship based on these characteristics is entering the BSA, measurements will be reliable. With requirement 2 performance conditions for processing the reference points are defined. Thus, they should only provide measurements if the vessel is within the BSA and if enough points are present within an individual reference point. As the system is based on 5HZ LiDAR sensors, reference points should provide measurements in less than 200ms. Accuracy requirements are derived based on the IMO Resolution A.915 describing minimal requirements for GNSS [54]. They define position accuracy metrics for GNSS for automatic docking maneuvers, hence the BAS must provide distance measurements with an accuracy of 0.1m.

To evaluate how the presented development process reacts to requirement changes, the update of the function is introduced. In particular, the measurement accuracy of the system gets refined by extending the condition that at least 5 points within a reference point are calculated by a further condition. A clustering of the recorded measurement points is recorded based on the distances to each other. The aim is to exclude outliers in distance measurements to enable a more precise evaluation of the measured values. This is reflected by updating the requirement 2.a to the following form:

2. a. If at least 5 pts with a maximum point to point distance of 1m are within a reference point, the distance must be calculated.

4.2 Requirements analysis

The first step is to analyze the requirements to translate requirements into VDs. For this purpose, parameters, their conditions, dependencies, and acceptance criteria are extracted from requirements, for which the results are shown in Table 9.

Seven parameters were derived from requirements 1.a to 1.e. VD 1.1 and 1.2 restrict the size of the supported area (i.e., length and width). VD 1.3 and 1.4 limit the minimum supported vessel size for the system. VD 1.5 describes that

Table 9 Derived verification descriptors based on requirements for a berthing assistant system

Req-ID	VD-ID	Parameter (P) & Condition (C)	Depen-dency (D)	Acceptance Criterion (AC)
1.a	1.1	$BSA_{Length} = 120m$	–	–
1.b	1.2	$BSA_{Width} = 100m$	–	–
1.c	1.3	$Vessel_{Length} > 16m$	–	–
1.d	1.4	$Vessel_{Width} > 4.8m$	–	–
1.e	1.5	$0.0^\circ < Vessel_{Angle} \leq 15^\circ \wedge Vessel_{Hull} \cap BSA$	1.1 ∧ 1.2 ∧1.3 ∧ 1.4	$Measurement \neq NULL$
2.a	2.1	$ReferencePoints_{PointCount} \geq 5$	1.5	$Measurement \neq NULL$
2.b	2.2	–	2.1	$0ms \leq Measurement_{Time} \leq 200ms$
2.c	2.3	–	2.1	$Measurement_{Accuracy} \leq 0.1m$

the hull must be within the BSA and that the vessel must not exceed a certain angle. One parameter was extracted from requirements 2.a to 2.c. VD 2.1 defines that if at least 5 points are within a reference point a measurement must be emitted. This is valid under the dependence that the ship is within the BSA, hence VD 1.5 is given as a dependency. VD 2.2 and 2.3 limit the calculation time and accuracy of the distance measurement. No parameters or conditions are specified for these VDs as they are always valid if a reference point provides measurements.

The introduction of an update for an existing system changes the requirements. Consequently, the VDs must be adapted to reflect the new requirements. For the SuT, the condition under which it should perform a measurement changes. Consequently, the condition for VD 2.1 must be modified. The results are shown in Table 10.

Due to the update, the condition was extended that the point-to-point distance must be smaller than 1 m within the reference point. Therefore, the parameter Reference Points $_{Point-To-PointDistance}$ and the corresponding condition is added to VD 2.1. Other VDs are not affected since they are merely linked to each other by dependencies. However, all VD 2.1 dependent descriptors must be retested due to the update, as the update could influence the measurements for the acceptance criterion.

4.3 Derivation of contracts

For the system and module contracts, the initial contractual requirement depicted in Table 11 can be derived from the VD.

The assumption of the BAS is a regular sensor input with a frequency of 5Hz within a time window of [0, 200]ms. A possible delay or offset is negligible in this case. As long as the sensor delivers values with a frequency of 5Hz, the BAS can guarantee a reaction by means of a reaction pattern. In this case, the reaction pattern looks like this: for a sensor input that identifies 5 points within the window of view, the distance to the quay wall is calculated within [0, 200]ms.

A monitor is derived from the structured assumption-guarantee contracts, which can observe the behavior of BAS in simulated or real environment. For this purpose, the monitor is connected to the interfaces of the realized implementation of the component in the operative system. As an input, sensor values must be received continuously with a frequency of 5Hz. If this input has less than 5 points the BAS should not show any reaction. In this case, however, the monitor can already report the failure of the assumption if a faulty input occurs over a long period of time, since an error on the part of the input source, i.e., the LiDAR sensor, could already be indicated here.

If the condition of an input of > 5 points is present, the reaction pattern must be monitored for time compliance. In this case, the time measurement of the calculation starts on the part of the monitor. From a timing perspective, the calculation can therefore take too long (>200ms). This can

Table 10 Changed descriptors based on an update

Req-ID	VD-ID	Parameter (P) & Condition (C)	Dependency (D)	Acceptance Criterion (AC)
2.a	2.1	$ReferencePoint_{PointCount} \geq 5 \ \wedge ReferencePoint_{Point-To-PointDistance} \leq 1m$	1.5	$Measurement \neq NULL$

Table 11 Contract of the initial configuration of the Berthing Assistant System

$C_{BerthingAssistentSystem(InitialConfiguration)}$						
Assumptions	Receive			Sensor Input	every	$[0, 200]ms\ (5HZ)$
Guarantees	Whenever	$\text{ReferencePoints}_{PointCount} \geq 5$	Provide	$\text{Distance}_{OS,QuayWall}$	within	$[0, 200]ms$

Table 12 Contract of the updated port positioning system

ID 1.0	$C_{BerthingAssistentSystem(Updated)}$					
Assumptions	Receive			Sensor Input	every	$[0, 200]ms(5HZ)$
Guarantees	Whenever	$\text{ReferencePoints}_{PointCount} \geq 5$	Provide	$\text{Distance}_{OS,QuayWall}$	within	$[0, 200]ms$
		$\text{ReferencePoints}_{Point-To-PointDistance} \leq 1$ m				

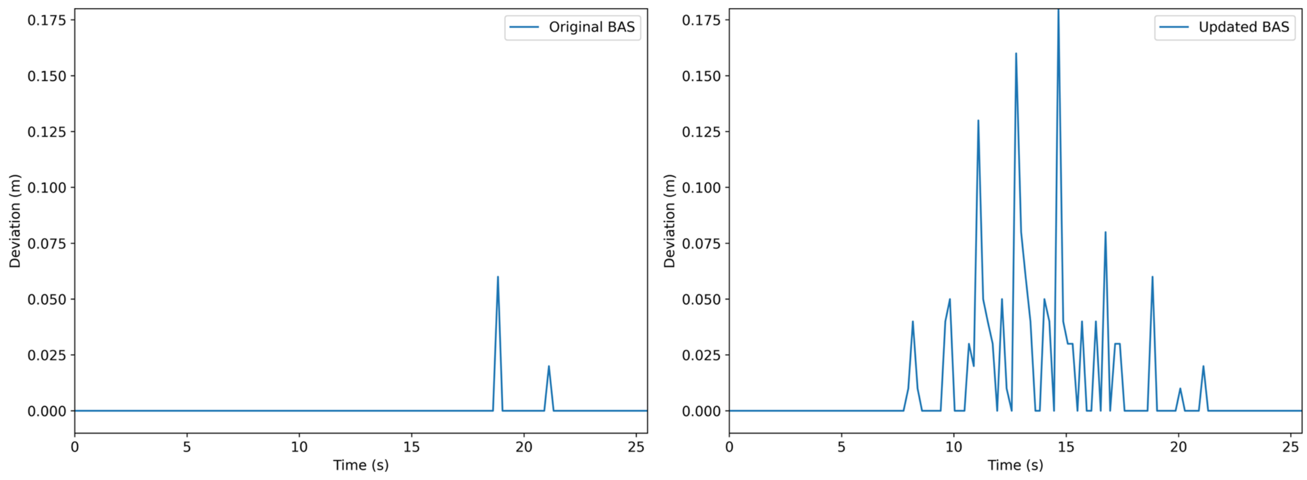

Fig. 11 Deviations in the distance calculation between both systems and the ground truth measurement during the scenario. Measurements of the original system (left) correspond to the ground truth. The system update (right) increases the deviations and exceeds the maximum acceptable deviation of 0.1m

point to faulty hardware or incorrect input from components the SuT is dependent on. Moreover, if there is a faulty clock of the component, it could lead to the malfunction of other components if dependencies exist.

For the update, the contract is extended to include the supplemented additional requirements from VD 2.1. As shown in Table 12, a further condition is added to include the clusters of point clouds as an additional condition. As explained in the VD, the point cloud may only have a maximum point-to-point distance between the reference points of ≤1 m, so that a distance calculation by the SuT can be guaranteed at its output interface.

In addition to the temporary runtime check, the value range can also be monitored for discrepancies. For example, due to the LiDAR sensors range, the possible measured distance of the identified OS is limited to a maximum range. The same applies to the maximum speed, which is limited on the one hand by the sensors accuracy, but also by contextual knowledge about the allowed ship types in the specific port. Thus, the monitor can be set up in a generalized way and supplemented by context information or expert knowledge in the concrete application field.

4.4 Derivation of scenario descriptions

Based on the VDs from Table 9 and the procedure from Sect. 3.4, mooring maneuvers are generated for the simulation. The VDs indicate that a ship with a length of at least 16 m and a width of 4.8m is berthing at the quay wall with a berthing angle of less than 15°. This allows to determine clear limits for the parameters of logical scenarios. Based on these boundaries, the combinatorial test case generation is performed to create concrete scenarios. To be able to quickly check the system for errors, we use the parameter boundaries. Hence, we choose a vessel with a length of 16 m and a width of 4.8m, performing a berthing maneuver with an angle of attack of 15° continuously reducing this value to 0°.

The basis of the BAS system are LiDAR sensors, which are affected by various weather effects and the reflectivity of a target. To simulate this phenomenon we use a probabilistic 3D LiDAR simulation from [55]. This allows to simulate LiDAR sensors under the influence of rain and target reflectivity. Thus, the limits of the system can be tested under bad weather conditions. Within the simulation, the prototypical setup from [53] was reproduced, with which the system was already tested under real conditions. This is a 120 m quay wall in which two 2D-LiDAR sensors with a maximum range of 250 m are integrated. The first sensor is placed at meter mark 0 m and the second one at meter mark 80 m.

Due to the updated function, the scenarios do not have to be derived again. These are only based on requirement 1 of the BAS, whereas the update only affects requirement 2. Therefore, the existing scenario catalog is reused for a new test run.

4.5 Monitoring results

Based on Table 9 and the contracts, two metrics are required for verification: (1) the accuracy of the BAS and (2) the computation time between the sensor and system output. For the evaluation of the accuracy, the ground truth distance measurement of the simulation is needed, hence the perpendicular distance between the vessel hull and the quay wall. The computation time, on the other hand, is measured by the difference in time between receiving the sensor data

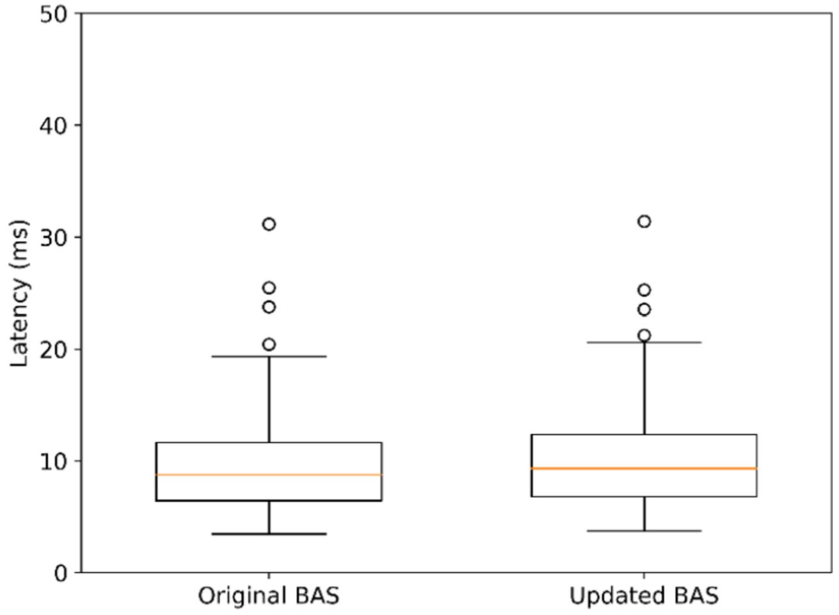

Fig. 12 Boxplot of latency monitoring results of reference points and its updated version. Both versions show comparable min/max latencies but the updated version has a higher average latency. 25 and 50 percentile are increased due to the update

and outputting the reference points. Both measurements are performed for the original form of the BAS and the updated version.

First, we evaluate the system's accuracy by observing the distance measurements. Figure 11 shows the deviation from the simulation ground truth for the original BAS version and it's updated form.

The deviation between the original BAS and the ground truth is 0.06m at the highest, with a mean deviation of 0.0007m. However, high deviations are generated by the updated BAS, with a maximum of 0.18m and an average deviation of 0.01m. As the vessel is approaching the quay wall, the angle between the sensor and the vessel gets smaller, reducing the point density. Thus, the maximum point spacing should be increased to better detect small ships. Considering that the maximum value is above the maximum defined deviation of 0.1m, the accuracy requirements cannot be met for the updated BAS.

Figure 12 shows the results of the latency measurement. For the original BAS, a minimum value of 3.48ms and a maximum value of 31.15ms were measured. The average latency is 9.62ms. Similar values are generated by the updated version. The minimum latency is 3.74ms and the maximum latency is 31.4ms. An average value of 10.15ms is determined here. The measurements suggest that the latency increased due to the update, as the computational complexity increased. However, these are still below the repetition rate of the sensors, which transfer measurements every 200ms, so that the requirements regarding the latency behavior can be met. The change in the measured run times and distance measurements can be attributed to the new calculation basis introduced by the update. The accuracy of the measurement therefore requires a revision of the algorithm. Regarding the latencies, the runtime monitor also enables the detection of faulty hardware or at the interfaces of the module.

5 Conclusion

The evaluation of the functional update of a practically applied LiDAR-based BAS [53] shows that the steps developed in the development process can be implemented using real-world SuTs. The requirements for the berthing system can be transferred from the catalog of specifications of IMO Resolution A.915 [54], which sets the minimum requirements for GNSS. These include the accuracy of positioning for automated berthing maneuvers of less than 0.1m. In addition, technical specifications of the existing system, such as a 5HZ frequency rate of the LiDAR sensor, set time limits that must continue to be adhered to after the update. This meant that both regulatory and technical requirements could be transferred to the VD in a structured manner in order to be contractually recorded in the further course and checked for compliance using generated scenarios. The measurements and comparison of the accuracy and time behavior of the original and updated SuT show both a breach of contract in terms of accuracy and compliance with the contractual specifications in terms of time requirements. This shows that the presented development process enables the structured measurement of test specifications from unstructured external sources. In addition, it could be shown that tests can be made more reliable, as requirement catalogs are structured in a measurable and formal way and thus compliance and breach of contract can be determined during an update using repeatable tests based on a defined rule set. The unreliable accuracy with a deviation of $> 0.1m$ was determined and the update rejected before deployment. Since contract-based design also enables the consistency check of the integrated construct the specifications of the components of the SuT, the comparison of the original contract model and the contracts modified by the update can also be checked for correctness. Using monitors at runtime, compliance with the rules could also be realized during operation, thus further increasing the reliability of the approved system.

6 Outlook and discussion

In this work a development approach is demonstrated that integrates scenario-based verification with formal test capabilities based on assumption-guarantee contracts. The process explicitly takes future change management, such as updates to the initial system, into account. As such, the assessor of the system, e.g., the classification society responsible in providing release approval, can re-run existing scenarios and test procedures and can track changes made to safety-relevant aspects based on the previously defined scenario and contract specifications.

The approach presented shows how unstructured requirement texts written in natural language can be transferred into a structured test procedure that extends over all phases of the development process. The introduced VD provides a bridge to build delimitable scenario cases as well as formalized and contractually verifiable requirement descriptions. The VD integrates all external requirements and regulations and can be used by the subsequent development branches. By that it helps to reduce information asymmetries between software development departments and system testers and, in the event of non-compliance, directly shows which parts have not been met.

With the focus of the development process towards changes to the overall system composition, it is also shown how feedback loops can be used to reduce re-verification effort in case of an update. The process does not have to be completely rolled out again in every case but can also be resumed at intermediate steps. Furthermore, not all development branches are always affected and existing parts can be reused. This avoids unnecessary repetition of work and streamlines the process, enabling a faster response to urgent updates. The feedback loops and the linkage of process artifacts also allow the affected area to be targeted directly when errors are identified in system development, without necessarily addressing all branches.

Overall, the development areas can thus work in parallel based on a common specification construct in the form of the VD, without all branches having to stop work in the event of breaks in one part of the overall development process. In addition, the introduction of contract-based VITs means that system analyses can be performed even before a component has been fully developed. After completion, software tests and contract-based monitoring can additionally underline the functional capability of a component.

Further research can build up on this and address the traceability of an update in the sense of an impact analysis, to be able to determine the effect of an update even more precisely and to limit the re-verification as well as the re-certification effort to the affected parts. One possibility would be the functional decomposition of the SuT and the determination of the module dependencies and the impact of the update within a SoS. This would also allow to only repeat the required test scenarios that are covering the particular module. Finally, an additional focus could be on the early verification within the development phase to reduce efforts in test beds. This is possible primarily through formal verification by means of contracts, which should be reconciled with the advantage of early security guarantees on security requirement statements without testing effort.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Jorgensen RN (2017) BIMCO and CIRM propose software maintenance standard for shipping. https://www.bimco.org/news/priority-news/20171214_software-maintenance
2. CIRM/BIMCO Joint Working Group (2017) Industry standard on software maintenance of shipboard equipment v1.0
3. International Association of Classification Societies (IACS) (2018) Recommended procedures for software maintenance of computer based systems on board
4. International Organization for Standardization (2021) ISO 24060:2021 - ships and marine technology - ship software logging system for operational technology
5. International Association of Classification Societies (IACS) (2022) Unified requirement for electrical and electronic installations (UR E26) - cyber resilience of ships
6. International Association of Classification Societies (IACS) (2022) Unified requirement for electrical and electronic installations (UR E27) - cyber resilience of on-board systems and equipment
7. Heikkilä E, Tuominen R, Tiusanen R, Montewka J, Kujala P (2017) Safety qualification process for an autonomous ship prototype – a goal-based safety case approach, in marine navigation. CRC Press, Gdynia, Poland, pp 365–370

8. Dreossi T, Fremont DJ, Ghosh S, Kim E, Ravanbakhsh H, Vazquez-Chanlatte M, Seshia SA (2019) VerifAI: a toolkit for the formal design and analysis of artificial intelligence-based systems. In Dillig I, Tasiran S (Eds) Computer aided verification. Springer International Publishing, Cham. Lecture Notes in Computer Science, pp. 432–442. https://doi.org/10.1007/978-3-030-25540-4_25
9. Riedmaier S, Ponn T, Ludwig D, Schick B, Diermeyer F (2020) Survey on scenario-based safety assessment of automated vehicles. IEEE Access 8:87456–87477. https://doi.org/10.1109/ACCESS.2020.2993730
10. Ehmen G, Koopmann B, Bebawy Y, Ittershagen P, Measurement-based online verification of timing properties in distributed systems. In: 2020 international conference on omni-layer intelligent systems (COINS) (IEEE, Barcelona, Spain), pp. 1–6. https://doi.org/10.1109/COINS49042.2020.9191647
11. Myklebust T, Stålhane T, Hanssen GK (2020) Agile safety case and DevOps for the automotive industry. In Proceedings of the 30th European safety and reliability conference and 15th probabilistic safety assessment and management conference (Research Publishing Services), pp. 4652–4657. https://doi.org/10.3850/978-981-14-8593-0_3495-cd
12. Ugarte M, Querejeta, Etxeberria L, Sagardui G (2020)Towards a DevOps approach in cyber physical production systems using digital twins. In: Casimiro A, Ortmeier F, Schoitsch E, Bitsch F, Ferreira P (Eds) Computer safety, reliability, and security. SAFECOMP 2020 Workshops, ed. by (Springer International Publishing, Cham, 2020), Lecture Notes in Computer Science, pp. 205–216.https://doi.org/10.1007/978-3-030-55583-2_15
13. International Organization for Standardization (2018) ISO 26262:2018 - Road vehicles - functional safety
14. Gautham S, Jayakumar AV, Rajagopala A, Elks C (2021) Realization of a model-based DevOps process for industrial safety critical cyber physical systems. In: 2021 4th IEEE international conference on industrial cyber-physical systems (ICPS), pp. 597–604.https://doi.org/10.1109/ICPS49255.2021.9468213
15. Munk P, Schweizer M (2022) Trapp M, Schoitsch E, Guiochet J, Bitsch F (Eds) Computer safety, reliability, and security. SAFECOMP 2022 workshops, vol. 13415. Springer International Publishing, Cham. pp. 145–157. https://doi.org/10.1007/978-3-031-14862-0_11
16. Tahvonen T, Uusitalo E (2018) Easy approach to requirements syntax in nuclear power plant safety design. In: 2018 1st international workshop on easy approach to requirements syntax (EARS), pp. 1–2.https://doi.org/10.1109/EARS.2018.00006
17. Fu R, Bao X, Zhao T (2017) Generic safety requirements description templates for the embedded software. In: 2017 IEEE 9th international conference on communication software and networks (ICCSN), pp. 1477–1481. https://doi.org/10.1109/ICCSN.2017.8230353
18. Gillani M, Ullah A, Niaz HA (2018) Survey of requirement management techniques for safety critical systems. In: 2018 12th international conference on mathematics, actuarial science, Computer Science and Statistics (MACS). pp. 1–5. https://doi.org/10.1109/MACS.2018.8628389
19. Gerwinn S, Möhlmann E, Sieper A (2019) In: Waschl H, Kolmanovsky I, Willems F (Eds) Control strategies for advanced Driver assistance systems and autonomous driving functions : development, testing and verification. Lecture Notes in Control and Information Sciences. Springer International Publishing, Cham. pp. 67–87. https://doi.org/10.1007/978-3-319-91569-2_4
20. Gomes C, Thule C, Broman D, Larsen PG, Vangheluwe H (2019) Co-simulation: a survey. ACM Comput Surv 51(3):1–33. https://doi.org/10.1145/3179993
21. Neurohr C, Westhofen L, Henning T, de Graaff T, Möhlmann E, Böde E, Fundamental considerations around scenario-based testing for automated driving. In 2020 IEEE intelligent vehicles symposium (IV) (2020), pp. 121–127. https://doi.org/10.1109/IV47402.2020.9304823
22. Guissouma H, Hohl CP, Lesniak F, Schindewolf M, Becker J, Sax E (2022) Lifecycle management of automotive safety-critical over the air updates: a systems approach. IEEE Access 10:57696–57717. https://doi.org/10.1109/ACCESS.2022.3176879
23. Holthusen S, Quinton S, Schaefer I, Schlatow J, Wegner M (2016) Using multi-viewpoint contracts for negotiation of embedded software updates. Electron Proc Theor Comput Sci 208:31–45. https://doi.org/10.4204/EPTCS.208.3
24. Watanabe K, Kang E, Lin CW, Shiraishi S (2018) Runtime Monitoring for Safety of intelligent vehicles. In 2018 55th ACM/ESDA/IEEE design automation conference (DAC). IEEE, San Francisco, CA, pp. 1–6. https://doi.org/10.1109/DAC.2018.8465912
25. Tabassam N, Fränzle MG (2022) Scenario-oriented contract based design for safety of autonomous vehicles. In Bie Y, Qu BX, Howlett RJ, Jain LC (Eds) smart transportation systems 2022. Springer Nature, Singapore. Smart innovation, systems and technologies, pp. 171–183. https://doi.org/10.1007/978-981-19-2813-0_18
26. Fremont DJ, Yue X, Dreossi T, Ghosh S, Sangiovanni-Vincentelli AL, Seshia SA (2019) Scenic: a language for scenario specification and scene generation. In: PLDI 2019: proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation, pp 63–78. https://doi.org/10.1145/3314221.3314633
27. Nuzzo P, Lora M, Feldman YA, Sangiovanni-Vincentelli AL (2018) CHASE: contract-based Requirement engineering for cyber-physical system design. In: 2018 design, automation & test in Europe conference & exhibition (DATE). IEEE, Dresden. pp. 839–844.https://doi.org/10.23919/DATE.2018.8342122
28. Philipp R, Qian H, Hartjen L, Schuldt F, Howar F (2021) Simulation-based elicitation of accuracy requirements for the environmental perception of autonomous vehicles. In: Margaria T, Steffen B. Leveraging applications of formal methods, verification and validation. Springer International Publishing, Cham. Lecture Notes in Computer Science, pp. 129–145. https://doi.org/10.1007/978-3-030-89159-6_9
29. International Maritime Organization (2003) COLREG: convention on the international regulations for preventing collisions at sea. IMO Publication (International Maritime Organization)
30. Perera L, Carvalho J, Guedes Soares C (2009) Advanced ship design for pollution prevention. Taylor & Francis Group, London, UK. pp 205–216.https://doi.org/10.1201/b10565-26
31. International Organization for Standardization (2018) ISO/IEC/IEEE international standard - systems and software engineering – life cycle processes – requirements engineering. ISO/IEC/IEEE 29148:2018(E) pp. 1–104. https://doi.org/10.1109/IEEESTD.2018.8559686
32. Martins LEG, Gorschek T (2020) Requirements engineering for safety-critical systems: an interview study with industry practitioners. IEEE Trans Softw Eng 46(4):346–361. https://doi.org/10.1109/TSE.2018.2854716
33. Fanmuy G, Fraga A, Llorens J (2012) Requirements Verification in the Industry. In: Hammami O, Krob D, Voirin JL (Eds) Complex systems design & management. Springer, Berlin, Heidelberg. pp. 145–160. https://doi.org/10.1007/978-3-642-25203-7_10
34. Rupp C (2020) Requirements-engineering Und -management. Carl Hanser Verlag GmbH & Co. KG. pp I–XVI. https://doi.org/10.3139/9783446464308.fm
35. Akkermann A, Hjollo BA (2019) Scenario-based V &V in a maritime co-simulation framework. In: 2019 spring simulation conference. SpringSim 2019) (Institute of Electrical and Electronics Engineers (IEEE), Tucson, Arizona, USA, pp. 1–12. https://doi.org/10.23919/SpringSim.2019.8732871

36. Brinkmann M, Bode E, Lamm A, Maelen SV, Hahn A (2017) Learning from automotive: testing maritime assistance systems up to autonomous vessels. In: Oceans 2017 – Aberdeen. IEEE. pp. 1–8. https://doi.org/10.1109/OCEANSE.2017.8084951
37. PEGASUS consortium (2019) Pegasus method - an overview. https://www.pegasusprojekt.de/
38. Lamm A, Hahn A (2018) Towards critical-scenario based testing with maritime observation data. In: 2018 Oceans. MTS/IEEE Kobe Techno-Oceans (OTO). IEEE. https://doi.org/10.1109/OCEANSKOBE.2018.8559045
39. Reiher D, Hahn A (2021) Review on the current state of scenario- and simulation-based V &V in application for maritime traffic systems. In: OCEANS 2021: San Diego – Porto. IEEE. pp. 1–9. https://doi.org/10.23919/OCEANS44145.2021.9705781
40. Menzel T, Bagschik G, Maurer M (2018) Scenarios for development, test and validation of automated vehicles. In: 2018 IEEE intelligent vehicles symposium (IV). pp. 1821–1827. https://doi.org/10.1109/IVS.2018.8500406
41. Schuldt F, Saust F, Lichte B, Maurer M, Scholz S (2013) Effiziente Systematische Testgenerierung Für Fahrerassistenzsysteme in Virtuellen Umgebungen. In AAET https://doi.org/10.24355/DBBS.084-201307101421-0
42. Jahanbin S, Zamani B (2018) Test model generation using equivalence partitioning. In 2018 8th international conference on computer and knowledge engineering (ICCKE). pp. 98–103. https://doi.org/10.1109/ICCKE.2018.8566335
43. Aryandana I, Permanasari A, Adji T (2020) Comparing method equivalence class partitioning and boundary value analysis with study case add medicine module. IOP Conf Ser: Mater Sci Eng 732:012072. https://doi.org/10.1088/1757-899X/732/1/012072
44. Grindal M, Offutt J, Andler SF (2005) Combination testing strategies: a survey. Softw Test Verif Reliab 15(3):167–199. https://doi.org/10.1002/stvr.319
45. Port of Hamburg (2023) Special Ever Ace. http://www.hafen-hamburg.de/en/special/ever-ace/
46. Reiher D, Hahn A (2021) Towards a model-based multi-layered approach to describe traffic scenarios on a technical level. J Mar Sci Eng. https://doi.org/10.3390/jmse9060673
47. Cimatti A, Dorigatti M, Tonetta S (2013) OCRA: a tool for checking the refinement of temporal contracts. In: 2013 28th IEEE/ACM international conference on automated software engineering (ASE) (2013), pp. 702–705. https://doi.org/10.1109/ASE.2013.6693137
48. Sharf M, Besselink B, Molin A, Zhao Q, Johansson HK (2021) Assume/guarantee contracts for dynamical systems: theory and computational tools. IFAC-PapersOnLine 54(5):25–30. https://doi.org/10.1016/j.ifacol.2021.08.469
49. Xie J, Tan W, Yang Z, Li S, Xing L, Huang Z (2022) SysML-based compositional verification and safety analysis for safety-critical cyber-physical systems. Connect Sci 34(1):911–941. https://doi.org/10.1080/09540091.2021.2017853
50. International Organization for Standardization (2005) ISO 17894:2005 - ships and marine technology - computer applications - general principles for the development and use of programmable electronic systems in marine applications
51. Reiher D, Hahn A (2022) Ad Hoc HLA simulation data model derived from a model-based traffic scenario (2022). https://doi.org/10.48550/arXiv.2208.06234
52. Francalanza A, Aceto L, Achilleos A, Attard DP, Cassar I, Della Monica D, Ingólfsdóttir A (2017) A foundation for runtime monitoring. In: Lahiri S, Reger G (Eds) Runtime verification. Springer International Publishing, Cham. Lecture Notes in Computer Science, pp. 8–29. https://doi.org/10.1007/978-3-319-67531-2_2
53. Mentjes J, Wiards H, Feuerstack S (2022) Berthing assistant system using reference points. J Mar Sci Eng 10(3):385. https://doi.org/10.3390/jmse10030385
54. International Maritime Organization (IMO) (2001) Revised maritime policy and requirements for a future global navigation satellite system (GNSS)
55. Bathmann M, Feuerstack S (2022) Validation of a probabilistic model for the consideration of rain and target reflection effects within maritime 3D LIDAR simulations. In MARESEC 2022. https://elib.dlr.de/188307/