X-by-Construction Design Framework for Engineering Autonomous and Distributed Real-time Embedded Software Systems

Research and Innovation Actions

Horizon 2020, Topic ICT-50-2020:
Software Technologies

Grant agreement ID: 957210

– Deliverable –

D5.1: Test Strategy and V&V Specification

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957210.

Document information

Document title:	Test Strategy and V&V Specification
Work package:	WP5
Editor:	QUB
Author(s):	QUB, VECTOR, KIT
Reviewer(s):	DLR, BMW
Document type:	Report
Version:	1.0
Status:	Released
Dissemination level:	Public

XANDAR consortium

No.	Short name	Name	Country
1	KIT	Karlsruher Institut für Technologie	Germany
2	UOP	University of Peloponnese	Greece
3	DLR	Deutsches Zentrum für Luft- und Raumfahrt	Germany
4	AVN	AVN Innovative Technology Solutions Limited	Cyprus
5	VECTOR	Vector Informatik GmbH	Germany
6	BMW	Bayerische Motoren Werke Aktiengesellschaft	Germany
7	QUB	The Queen's University of Belfast	United Kingdom
8	FENTISS	Fent Innovative Software Solutions SL	Spain

Copyright & disclaimer

Document revision history

Version	Date	Comments
1.0	2021-06-29	Publication of the final version.

About this document

This technical report specifies the planned Verification & Validation (V&V) and test strategies for the XANDAR project. Verification & validation (V&V) are critical activities of developing any real-time embedded software systems, including cars, airplanes, and helicopters. Validation is often defined as “Are we building the right system?”, and verification is defined as: “Are we building the system right?”. In other words, validation is concerned with checking that the system will meet the customer’s needs, while verification is concerned with whether the system is well-engineered, and defect-free.

Using four tasks, WP5 will investigate, develop, and apply methods to validate and verify functional and non-functional requirements of software systems developed by the XANDAR tools. Additionally, the XANDAR methods used by the XANDAR tools will be verified and validated via WP5 to prove X-by-Construction (XbC) guarantees. The verification tools and methods developed in WP5 will be used to evaluate the use-case applications and generate feedback for the end-user partners.

Table of contents

List of figures

List of tables

Glossary of terms

ADRESS	Autonomous & Distributed Real-time Embedded Software System
CF	Composite function
EC	Event chain
ECU	Engine control unit (ECU)
LA	Logical Architecture
MBT	Model-based testing
MiL	Model-in-the-loop
NFR	Non-functional requirement
QA	Quality Assurance
RTE	Runtime Environment
SA	Software Architecture
SCADA	Supervisory Control and Data Acquisition
SDLC	System Development Life-cycle
SIM	Simulation
SLR	Systematic Literature Review
SUA	System Under Analysis
SUT	System Under Test
SWC	Software component
TA	The Timing Architects tool developed by Vector
V&V	Verification & validation

1 Introduction

Verification & validation (V&V) are critical activities of developing any real-time embedded software systems, including cars, airplanes and helicopters. Validation is often defined as “Are we building the right system?”, and verification is defined as: “Are we building the system right?” [1]. In other words, validation is concerned with checking that the system will meet the customer’s needs, while verification is concerned with whether the system is well-engineered, and defect-free.

V&V approaches include various software and system testing methods and other software and system quality assurance (QA) activities.

At the same time, many reports clearly show that software has dominated the automotive and also aviation industry [2-7]. Many industry reports indicate that *“software drives innovation in the automotive industry”* [8].

In the past, inadequate V&V of embedded software systems such as automotive and aviation industry have led to many severe and unfortunate consequences. For example, two Boeing 737 MAX airplanes crashed (one in 2018 and one in 2019) because of various issues, including defects in software and sensors. A large number of new articles, industrial and academic technical reports and papers have covered the issues and its root causes, e.g., [9, 10].

Likewise, in the automotive industry also, a very large number of crashes and accidents have occurred because of software defects which have leaked into production systems (cars) due to inadequate V&V during the system development process. Some example reports can be found in [11-16].

For the XANDAR project, the focus is on engineering (designing, developing, and testing) clearly autonomous & distributed real-time embedded software systems, and the two selected use cases are from the above two domains (automotive and aviation industry):

- Use-case 1 (UC1): Urban air mobility
- Use-case 2 (UC2): Autonomous driving vehicles

Thus, for the XANDAR project, V&V activities are considered critical and will be applied for all the WPs and WP tasks in the project.

X-by-Construction Design framework for Engineering Autonomous & Distributed Real-time Embedded Software Systems

In this technical report (Deliverable D5.1: Test Strategy and V&V Specification), we specify the overall V&V strategy, methods and tools that will be investigated, developed, and used as part of the WP5.

The remainder of this technical report is structured as follows. A background discussion and a brief literature review are presented in Section 2. Section 3 presents the objectives and context of V&V in the XANDAR project. Section 4 discusses the V&V strategy, methods, and tools for each of the tasks in WP5. To provide traceability of WP5’s tasks to its deliverables, Section 5 provides a visual map of the tasks of WP5 to deliverables. Also, to provide traceability between WP5 and other WPs, we present the input/output dependencies of WP5 with other WPs in Section 5. Finally, Section 6 provides a summary of this technical report.

2 Background and literature review

In this section, we provide a brief literature review of different V&V approaches in software and system engineering. We then provide a focused literature review of V&V of autonomous & distributed real-time embedded software systems.

2.1 V&V approaches in software and system engineering

The history of V&V in software and system engineering is as long as such software-intensive systems have been developed, since circa 1950's. One of the earliest and classic books on software testing (as one V&V approach) was published in 1979 [17].

In the past 7 decades or so, a large number of V&V approaches and techniques have been proposed in software and system engineering. To classify those techniques the Venn diagram in Figure 2-1 shows the relationship of different V&V approaches (source: [1]). As we can see, there is adverse type and number of V&V approaches.

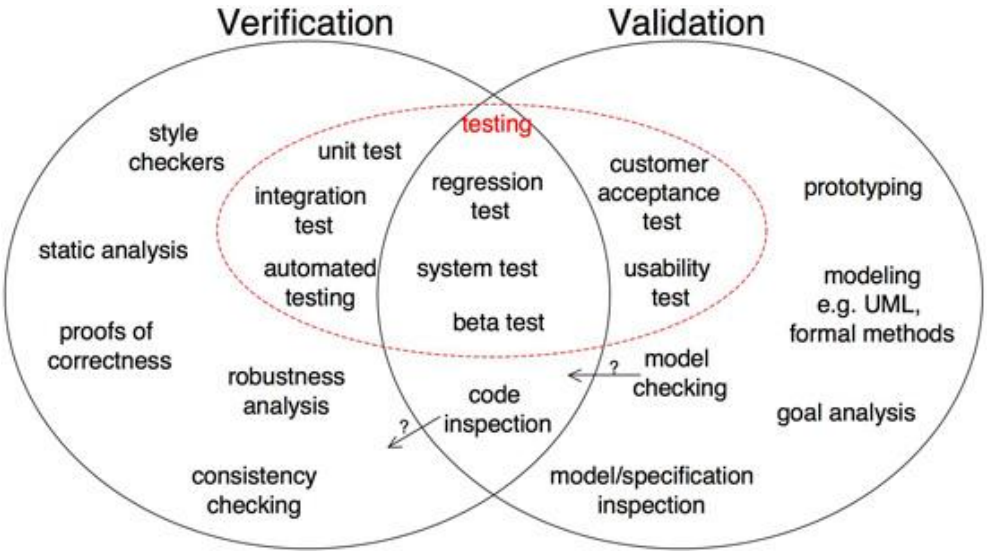

Figure 2-1: Venn diagram of V&V approaches in software and system engineering [1]

Testing (of software and systems) is considered among the most often used V&V approaches and itself has various types, e.g. unit, integration and system testing. Some types of testing are under verification, and some other type approach types are under validation.

In addition to testing, other V&V approaches include static analysis [18], model checking [19, 20], and proof of correctness [21].

Static program analysis (also called: static code analysis, or just: static analysis) [18] is the analysis of software that is performed without actually executing programs, in contrast with dynamic analysis, which is analysis performed on programs while they are executing.

Model checking [19, 20] (also called: property checking) is a method for checking whether a finite-state model of a software system meets its given specification. Model checking has been widely applied for QA of systems, where the specification contains liveness requirements as well as safety requirements, e.g., [22].

Proof of correctness [21] is an approach based on mathematical proof with the goal that a computer program or a part thereof will, when executed, yield correct results, i.e. results fulfilling specific requirements. Before proving a program correct, the theorem to be proved must, of course, be formulated.

Each of the V&V approaches shown in Figure 2-1 have a very large body of knowledge in both the academic and grey literature. There are various survey and Systematic Literature Review (SLR) papers on each of the V&V approaches, each providing references to papers in those bodies of knowledge, e.g., two survey papers on static analysis in general [23, 24], a survey paper on static analysis methods for identifying security vulnerabilities [25], three survey papers of different model checking approaches [26-28], and a survey on proof of correctness [29].

In addition to academic research, each of the V&V approaches shown in Figure 2-1 have been widely used in the industry in the past several decades. Several example case studies of using static analysis in industry can be found in [30-32]. Some example case studies of using model checking in industry can be found in [33-37]. Several example case studies of using proof of correctness in industry can be found in [38-41].

Last but not least, software testing has a very large body of knowledge and expertise in both academic and grey literature. A very large number of books and papers on new advances of testing, often focusing on specific test approaches and specific domains (e.g., automotive) are regularly published.

From another perspective, some of the embedded software systems to be developed in XANDAR will include AI components, e.g., using machine learning (ML), as part of the two use-cases: (UC1): Urban air mobility, and (UC2): Autonomous driving vehicles. Many recent efforts by researchers and practitioners for V&V of AI / ML software have appeared in the literature, e.g., [42-46].

In the XANDAR project, by being aware of the diversity of V&V approaches (as shown in Figure 2-1), our goal is to use the “right” V&V approaches for each of our V&V needs, in different WPs, which we will discuss in the rest of this technical report.

2.2 V&V of Autonomous & Distributed Real-time Embedded Software Systems

Having presented a brief review of the literature on the general scope of V&V approaches in software and system engineering, for all domains and system types, we now present a brief review of the literature on the V&V of autonomous & distributed real-time embedded software systems, since that is the scope of the XANDAR project.

A large body of knowledge has been accumulated on V&V of autonomous & distributed real-time embedded software systems in the past few decades, e.g., various books and papers have been published on the topic.

The lead of WP5 was involved as the lead author in a Systematic Literature Review (SLR) of testing embedded software which was reported in [47, 48]. The SLR systematically selected and synthesized the findings of a set of 312 technical papers (“primary” studies in the SLR terminology). Figure 2-2 was one of the outputs of the SLR, in which a generic test process

showing different types of test activities and classifying the reviewed 312 technical papers are shown.

As Figure 2-2 shows, all six test activity types have received attention in the research community: (1) Test-case design, (2) Test execution (177 of the 312 papers in this area), (3) Test automation, (4) Test-result evaluation (using test oracles), (5) Test management, (6) Other test activities, such as test-case minimization, test prioritization, and test reuse.

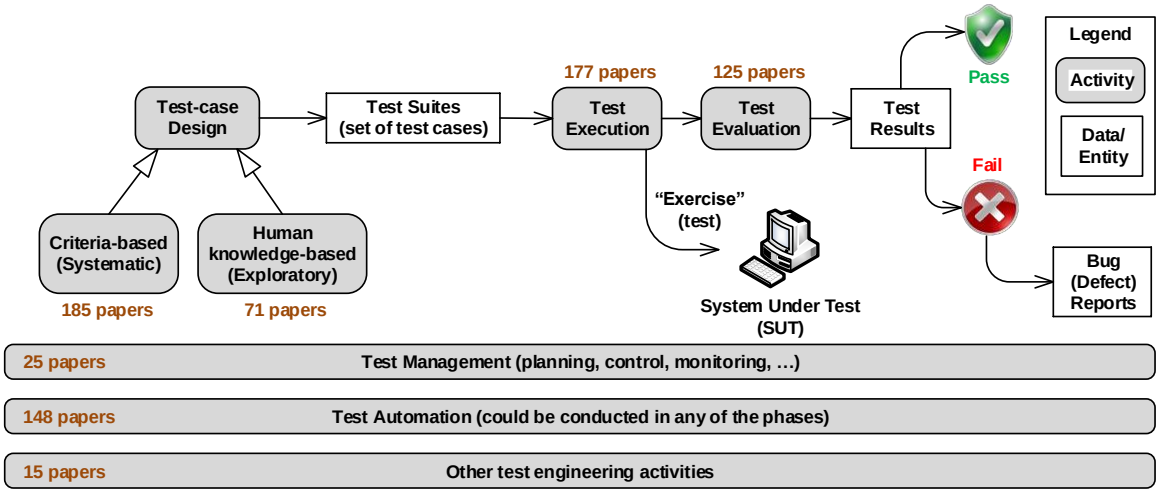

Figure 2-2: Classification of all 312 papers in testing embedded software [47, 48]

Several other survey and Systematic Literature Review (SLR) papers have also been published on V&V of Autonomous & distributed real-time embedded software systems. For example, a review of V&V for machine learning and the corresponding challenges in the automotive industry was presented in [49]. Two other surveys can be found in [50, 51].

3 Objectives and context of V&V in the XANDAR project

Objectives of V&V (WP5) in the XANDAR project are as follows:

- Designing and applying V&V techniques to validate and verify functional (**Task 5.2**) and non-functional requirements (**Task 5.3**) of the XbC approach and systems to be developed using the approach. Requirements such as correctness, trustworthiness, security, and real-time properties, developed in WP1, WP2, WP3 and WP4, will be considered.
- Providing V&V toolsets (**D5.3, D5.4** and **D5.5**) that enable end-users of the XANDAR toolchain to carry out V&V activities.
- Verifying and validating the XANDAR toolchain itself, which we refer to as *meta-V&V* (**Task 5.4**), to prove the envisioned XbC guarantees for generated programs.

The overall XANDAR development process, which is introduced and described in D2.1 of the project, is shown in Figure 3-1. The parts highlighted in green are V&V activities. From this, we can see that V&V plays a substantial role in the project, as every activity and generated artefact has to be properly verified and validated. This is especially important given the two safety-critical use-cases systems of the project: (UC1): Urban air mobility, and (UC2): Autonomous driving vehicles (as specified in WP1).

As shown in the figure, V&V activities related to functional and non-functional requirements are treated during two different phases of the development process. During the first phase, a simulation incorporating all available inputs from the developer is conducted to validate and verify the correct functionality of the system under consideration. In the second phase, this input is used for the generation of the actual system components. During this generation procedure, activities related to the V&V of non-functional requirements are automatically performed by the toolchain. It is important that this procedure preserves all relevant aspects from the functional behaviour that was already validated and verified during the first phase.

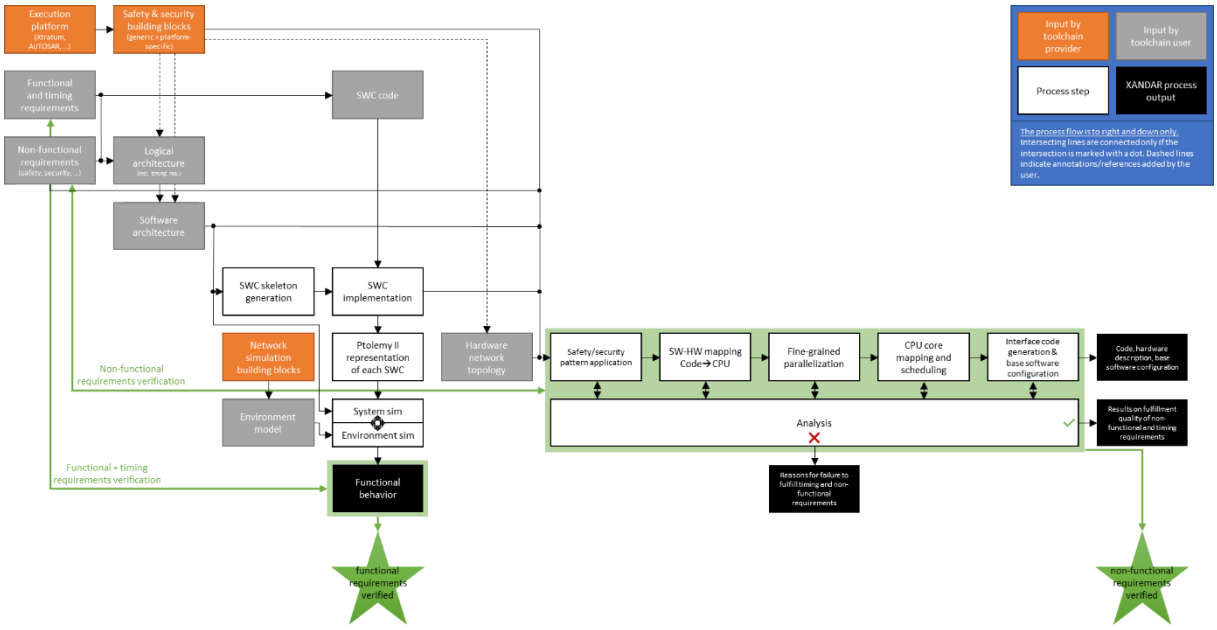

Figure 3-1: XANDAR development process as described in D2.1

Timing V&V can be conducted at different stages of the development process. In XANDAR, an early timing validation on system level is envisioned since this is currently not feasible in the proposed toolchain (missing means to model timing requirements and data). On this level, timing requirements are expected to be defined with respect to functional effect chains. Later in the process, these requirements typically need to be decomposed so they can be used on entities of the software model. In any of these cases, the timing validation serves as a mechanism that is entirely model based. As such, no source code is needed at this time. However, required (e.g., memory, CPU instructions, etc.) and provided (e.g., Flash, RAM, ROM, CPU architecture, etc.) resource estimations are key in this context. See Section 4.2 for details.

4 V&V strategy for each of WP5 tasks

We discuss next the V&V strategy and methods for each of the four follow-up tasks in WP5:

- Task 5.2: V&V of functional requirements (Lead: KIT, M7 - M24)
- Task 5.3: V&V of non-functional requirements (NFR) (Lead: Vector, M7 - M24)
- Task 5.4: Conducting V&V & meta-V&V in different levels (Lead: QUB, M7-M24)
- Task 5.5: V&V toolset integration (Lead: Vector, M25 – M33)

Note that Task 5.1 (Planning of Overall V&V/test Strategy: M1-M6) has already been completed and has generated the current technical report (D5.1).

For each task below, we discuss its goals, outcomes / deliverable of the task (if any), inputs of the task, how the task will be implemented, and any other important aspect as needed.

4.1 Task 5.2: V&V of functional requirements (Lead: KIT, M7 - M24)

4.1.1 Overview and goal of the task

The primary goal of Task 5.2 is to design and implement a toolset enabling users of the XANDAR development process to validate and verify the functional correctness of a System Under Analysis (SUA). Most importantly, this toolset builds up on the model-based description of underlying functional requirements, the Logical Architecture (LA), and the Software Architecture (SA) specified for the system.

It is tightly integrated into the early phases of the XANDAR development process, which is in part visualized in Figure 4-1. Note that the target-specific implementation and analysis procedure, which is part of the overall process as defined in D2.1, is mostly unrelated to the functional verification steps and therefore not shown in the excerpt.

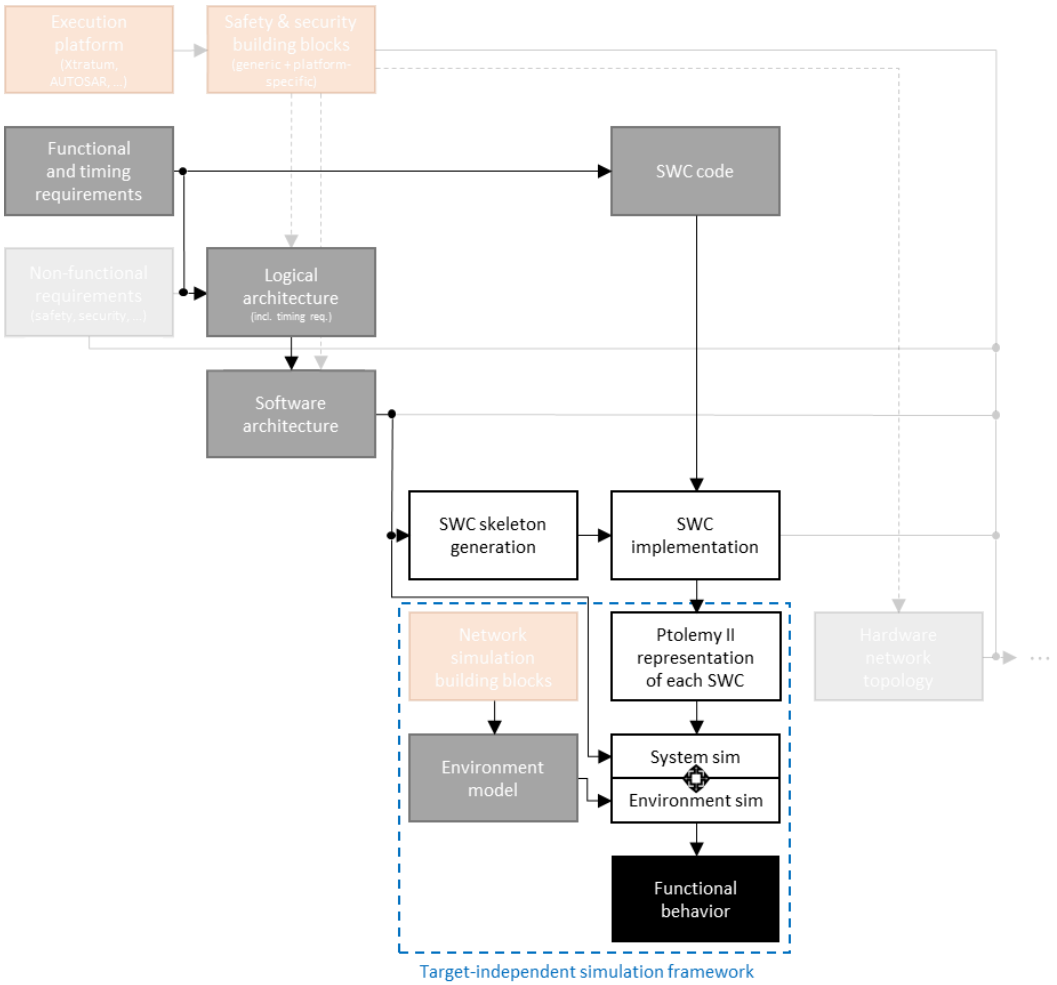

Figure 4-1: Excerpt from the XANDAR development process

4.1.2 Outcomes / deliverable of the task (if any)

Deliverable of the task will be a toolset (**D5.3**) that enables end-users to carry out V&V activities for functional correctness. This will include a MiL (model-in-the-loop) test and simulation framework (**D5.2**), the corresponding testing methods, and a set of static/formal verification modules. The toolset developed as part of this task will consist of the target-independent simulation framework, which is highlighted in blue in Figure 4-1.

4.1.3 Input(s) of the task

The target-independent simulation framework (**D5.2**), which is highlighted in blue in Figure 4-1, will automatically execute open-loop or closed-loop simulations of the behaviour that the system under consideration exhibits. It will then report this behaviour to the developer (for validation) or automatically verify that it conforms to the functional requirements. Therefore, it expects several input artefacts to be provided during the early phases of the development process.

One of these input artefacts is the Software Architecture, which describes the network of software components (SWC) of the system. In addition, the framework expects a SWC implementation for each of the SWCs that are described as part of this network. Details on how these SWC implementations are derived can be obtained from D3.1 of the project.

For closed-loop simulations, the framework must further be supplied with an environment model developed using Ptolemy II. In any case, the Software Architecture must be annotated with the relevant open-loop and closed-loop test cases to execute.

It is important to understand that from the perspective of the target-independent simulation framework, the network of SWC implementations is the “model” of the system under consideration that is validated and/or verified using test cases supplied by the developer.

As part of the entire functional verification toolset, it will further be possible to annotate the Software Architecture with functional requirements that must be verified by performing a static analysis of the network of SWCs. Within the scope of the XANDAR project, however, such techniques will only be employed where the simulation-based approach is no longer able to produce the desired X-by-Construction guarantees.

4.1.4 How the task will be implemented

The most suitable approaches for V&V of functional requirements (correctness) for automotive and aviation software are model-based testing (MBT) [52-55] and model checking [56]. Given the team’s expertise in those areas, e.g., from our past projects [57-62], we plan to develop new MBT and model checking approaches for Task 5.2.

As described in the previous section, the envisaged MBT strategy is highly related to XANDAR’s software component design methodology introduced as part of D3.1.

Figure 4-2 visualizes how the XANDAR toolchain generates two adapters for each SWC that is specified using the XANDAR programming model: the simulation (SIM) adapter targeting the MiL simulation and the runtime environment (RTE) adapter targeting the actual RTE.

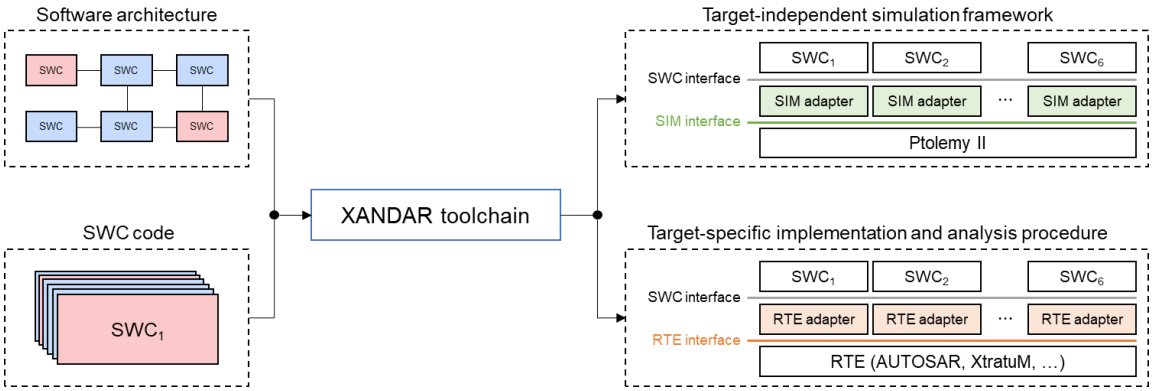

Figure 4-2: SIM and RTE adapter generation introduced in D3.1

Therefore, the design and the implementation of a generic approach to derive SIM adapters is an essential part of Task 5.2. A specific challenge that needs to be solved during this endeavour is the need to guarantee that all generated SIM adapters accurately represent the functional behaviour that the corresponding RTE adapters exhibit. Therefore, the requirements formulated by the use case providers in WP1 will be analysed with respect to the mechanisms that the adapters are expected to support. Based on this analysis, a representative set of such mechanisms will be defined and integrated into the toolset developed as part of this task. A careful analysis will then be performed to show that for the supported mechanisms, both the SIM and the RTE adapters are functionally equivalent.

Subsequently, the target-independent simulation framework will be analysed with respect to the functional guarantees that it is able to provide to the developer. Functional guarantees that are required by the use cases and cannot be provided by the framework will be separately considered and solved by developing suitable approaches (such as model checking, static analysis, code inspection, etc.). These approaches will be integrated into the developed toolset to complement the simulation framework.

Furthermore, as discussed in Section 2.1, some of the embedded software systems to be developed in XANDAR will include AI components, e.g., using machine learning (ML), as part of the two use-cases: (UC1): Urban air mobility, and (UC2): Autonomous driving vehicles. V&V of those AI-based software components will have both functional and also non-functional aspects. Functional V&V of a given AI component refers to whether the AI component provides the "right" output (prediction) based on functional requirements, and there is a quite vast literature on the topic, e.g., [42-46], which we plan to use.

Non-functional V&V of a given AI component, in our project, refers to whether the AI component would satisfy the three NFRs in our context (timing, safety, and security), i.e., answering the following questions:

- Does the AI component provide the output in the expected timing constraints? Some of existing works in this area are [63, 64].
- Does AI component satisfy its safety constraints? Some of existing works in this area are [65-67].
- Does AI component satisfy its security constraints? Some of existing works in this area are [67, 68].

4.2 Task 5.3: V&V of non-functional requirements (NFR) (Lead: Vector, M7 - M24)

4.2.1 Overview and goal of the task

This task will cover test planning, test-case design, test-model preparation, and V&V/Test execution with focus on non-functional requirements (NFR). In software and system engineering [69], a NFR is a requirement that specifies certain criteria that are used to judge the operation of a system, rather than specific behaviours, e.g., response time of the car breaking control system shall be less than 20ms. NFRs are contrasted with functional requirements that define specific behaviour or functions, e.g., the braking control system shall apply the brake on the car's wheels, with the intensity proportionate to the amount of pressure applied by driver on the brake pedal.

In software and system engineering [69], NFRs are divided into two main categories:

- *Execution qualities*, such as safety, security, and usability, which are observable during operation (at run time).
- *Evolution qualities*, such as testability, maintainability, extensibility, and scalability, which are embodied in the static structure of the system.

There are various international standards which cover NFRs in software and system engineering, e.g., the ISO/IEC 25010:2011 standard [70], fully named as: Systems and

software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE).

In the scope of XANDAR, NFRs of interest include timing requirements (also known as: timeliness and real-time requirements), functional safety, trustworthiness, and security aspects on different levels of abstraction.

T5.3 will investigate and develop V&V methods for NFRs using V&V approaches such as simulation, model-based testing, proofs of correctness, model consistency checking, model checking, model inspection, static analysis, and code inspection techniques that aim for asserting NFRs.

Since V&V of NFR are based on NFR requirements models in system specifications, T5.3 will closely relate to WP2 (Modelling, Behavioural Specification and Software Synthesis, M01-M24), in which the requirements for specification models (M1-M6) have been designed in T2.1.

4.2.2 Outcomes / deliverable of the task (if any)

Output deliverable of the T5.3 will be **D5.4** (V&V Toolset for Non-Functional Properties), on M24, to be led by the project partner Vector.

4.2.3 Input(s) of the task

To execute task T5.3, the following inputs will be used:

- **D2.1:** Modelling Requirements for System's and Interface Specifications (Report, M6, Vector), generated by **T2.1**
- **D2.4**: Architecture and Safety/Security Modelling (Report, M24, Vector), generated by **T2.2**
- **D5.1**: Test Strategy and V&V Specification (Report, M6, QUB), generated by **T5.1**

4.2.4 How the task will be implemented

As stated in Section 4.2.1, the relevant NFRs within the scope of the XANDAR project are real-time (timing) requirements, safety, security, and trustworthiness.

While there are various commercial and academic tools to verify and validate timing and safety properties, security and trustworthiness are new emerging V&V topics with limited tool support.

There are different aspects to consider for V&V of NFRs:

1. Data models suited for exchanging information about model artefacts relevant for the specific V&V activities (impacts result quality)
2. Levels of abstraction/granularity supported by these data models and the corresponding V&V activities (impacts suitability for a specific design process)
3. Available tools that provide means to conduct and support V&V activities (impacts acceptance during and beyond XANDAR)

For each of the above three aspects, we will outline our strategies in the following subsections.

4.2.5 Specifications languages / notations for specifying NFRs

As it has been discussed in D2.1, there are various specifications / requirements languages / notations for specifying different types of NFRs, including safety, security and safety requirements that are the focus on XANDAR.

A detailed review of five requirements languages / notations (potentially suitable for our purpose) are presented in D2.1. Those five requirements languages include: AMALTHEA APP4MC [71], AUTOSAR [72], EAST-ADL2 [73], Temporal Logic [74], UML MARTE [75], TIMMO2 (TADL2) [76]. Also, a comparison of those five requirements languages using four criteria are presented in D2.1, as follows: (1) Suitability to generate code using each of the requirements languages, (2) Meeting our use case requirements (from WP1), (3) Expressiveness of the requirements language, (4) Fitness for testing (testability).

As per the assessments presented in D2.1, the tentative plan is to use AUTOSAR [72] mainly for the timing requirements. For security and safety requirements, selection of suitable requirements languages and/or development of new requirements languages will be explored.

4.2.6 Approaches for V&V of NFRs

In principle, V&V of each of the three NFRs in our context (timing, safety, and security) will be planned and conducted separately in the first phase, and in a second phase will be conducted jointly. This is an established V&V approach (process) [77] for real-time embedded software systems. We discuss next our planned V&V approaches for each of the three NFR types: timing, safety, and security.

As discussed in Section 2.1, in the past several decades, a large number of different V&V approaches in software and system engineering have been proposed in industry and academia. Our team is well aware of the catalogue of existing V&V techniques, e.g., the lead of WP5 was involved as the lead author in a Systematic Literature Review (SLR) of testing embedded software [47, 48].

In a broad classification, V&V approaches for NFRs requirements are of three types: (1) testing (runtime verification), (2) runtime verification monitoring, and (3) formal static verification (without running the SUA). For each of the three NFR types in our context, we will review our planned approaches under the above three V&V approach types.

4.2.6.1 V&V approaches for timing requirements

Timing requirements are some of the most critical NFRs of real-time embedded software systems. One of the widely used V&V approaches for timing requirements is simulation, which is also referred to as *simulation testing* [78-81].

Other V&V approaches for timing requirements have also been used in the literature and also some of the team members in previous works, e.g., performance, load, and stress testing, e.g., [60, 61, 82-84]. Some of those performance / load testing approaches are model-based, e.g., the approach proposed in [84] uses UML models of the SUA.

4.2.6.2 V&V approaches for security requirements

We review our planned V&V approaches for security requirements, under the three categories of: (1) testing (runtime verification), (2) runtime verification monitoring, and (3) formal static verification (without running the SUA).

Various testing approaches for security requirements in the context of automotive systems have been proposed, e.g., [55, 85-89]. For the XANDAR toolchain and our specific use-cases, a novel extension of those approaches will be developed in the project.

Under the runtime verification monitoring category, there are approaches such as [90-92]. For the XANDAR toolchain and our specific use-cases, a novel extension of those approaches will be developed in the project.

Various formal verification approaches for security requirements in the context of automotive systems have been proposed, e.g., [93]. There are also model-based approaches in this category, e.g., [94]. For the XANDAR toolchain and our specific use-cases, a novel extension of those approaches will be developed in the project.

4.2.6.3 V&V approaches for safety requirements

For V&V of safety requirements too, a number of techniques have been proposed in the academia and grey literature, e.g., [95, 96]. A subset of those works have focused on formal verification of safety (without running the SUA), e.g.. [93].

Also, a number of works for functional safety verification have been based on the ISO26262 standard (named: Road vehicles – Functional safety), e.g., [97, 98].

For the XANDAR toolchain and our specific use-cases, a novel extension of those approaches will be developed in the project. We will describe our extension plans for each of the three NFRs in the following subsections.

4.2.7 Timing

When conducting V&V of NFRs, an important issue is the abstraction/granularity levels of V&V, e.g., in any process of the System Development Life-cycle (SDLC) in which V&V of NFRs is going to be performed, what level of the system (e.g., unit, integration or system) level perspective will be considered.

4.2.7.1 Timing Abstraction/Granularity Levels

There can be various points in the SDLC where it makes sense to do a timing validation and verification. A timing simulation is rather low level and requires fine granular information about the system to simulate. In cases where there is less detail known, a timing simulation may not work or be suited at all. In order to better grasp this situation of varying levels of granularity of timing-relevant data we defined three information levels which relate to specific points in the design process. We will describe the three levels to clarify where these points are in the following:

1. Component architecture (without implementation – no source code) and an initial expected resource demand is defined. There are no details about the (hardware)

platform or the operating system, yet. An abstract resource providing description should be given (for resource utilization estimations). Components are rather black boxes here, however, the communication between them is roughly specified. This level can mostly be represented by the AUTOSAR Abstract Platform model.
There is an additional, optional level in between 1 and 2 with one key difference. There, the following additional information is available: More details on the execution platform – these can be used to decide which operating system might be suited for resource management, i.e., depends on whether the platform is a microcontroller or a microprocessor.

2. The software components are now grey boxes – they contain information about their implementation (i.e. runnable), more detailed resource demand definitions, and possibly analysed or measured runtimes. Activation or trigger patterns are also specified in this level. The communication contains more information about the data sizes and how often data will be written/read.
3. At this level of granularity all information are present and available to do a timing simulation of the software on ECU level. There are runnable, tasks, their mappings to events and cores, scheduling configurations (i.e. priorities), the execution order of runnable is given, the software components are white boxes – there will be at least prototypically implemented, communication is completely defined (messages mapped to buses, etc.), and validation and verification scenarios are defined awaiting their execution. Depending on the tool capabilities and model granularity, this timing simulation includes timing effects of the operating system and hardware interactions. The accuracy of the timing simulation is generally closely related to model granularity and simulation depth.

To clarify which data is available for level 1, consider the following example in Figure 4-3. There are three composite functions denoted CF_x, each containing one or more elementary functions (not further decomposed – these represent the black boxes) denoted EF_x. Event chains are called EC_x, and signals/channels between elementary functions are named S_x. In the example there are two event chains: EC_1 and EC_2. EC_1 includes EF_1 (which could be a sensor function), S_1, EF_2, S_2, EF_3, S_3, and EF_4 (which may be an actuator function). It requires that the functions and their produced/processed signals shall be observable to in this particular order, and the time frame in which this shall happen should not exceed 10ms. Since every event chain (and sub event chain) consists of a stimulus and a response event, the overall event chain needs to be decomposed into sub chains, e.g. EF_1 start and finish time points are the first sub chain, S_1 transmission start and finish time points are the second sub chain, etc.

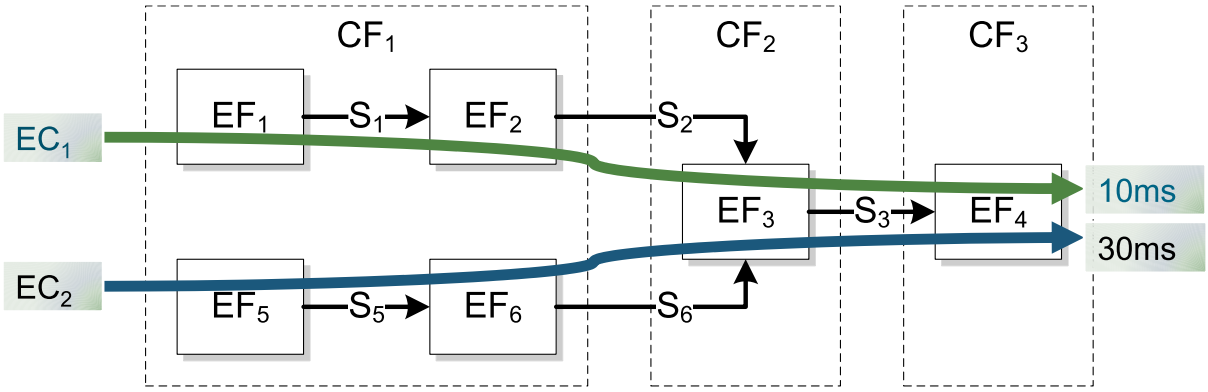

Figure 4-3: Example with two event chains.

These sub chains also need to have timing requirements summing up to EC_1's 10ms budget. Together with the activation patterns (i.e. how often are the EFs executed, and how often are signals transmitted), they represent rough resource demand estimations.

There is also a second event chain: EC_2. It contains several sub chains, some of which are also part of EC_1. The difficult task here for V&V activities is to ensure that the specifications of EC_1 and EC_2 are consistent and do not contradict each other. Any change on either event chain may invalidate the other. For a simple example like this, it may be simple to validate the consistency manually, but for higher degrees of complexity, we would like to automate this validation in order to early filter out impossible event chain constellations.

On top of the above example, consider the addition of new complex functions and a subsequent update of one or more event chains. This is shown in Figure 4-4. There is a new complex function CF_4, and CF_2 now contains an additional new elementary function NF_9. Obviously, the time budgets for EC_2 have to be updated since it now contains NF_9 and S_9. But how does this impact EC_1? How can we ensure that EC_1 and EC_2 are still valid without revalidating the whole system?

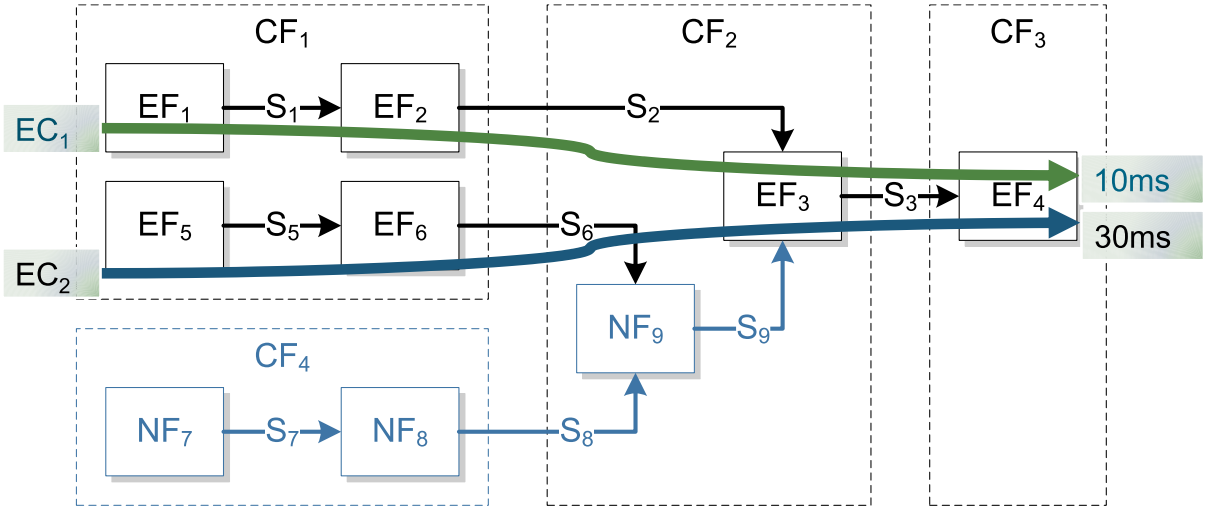

Figure 4-4: Example with two event chains and new functions.

As for level 2, consider the example given in Figure 4-5. This is the level 1 example enriched with resource provider information (the abstract hardware architecture) and the high-level

deployment to these resources. Complex (and consequently their elementary) functions are deployed to computing units, denoted CU_x, and signals are transmitted via buses, denoted B_x.

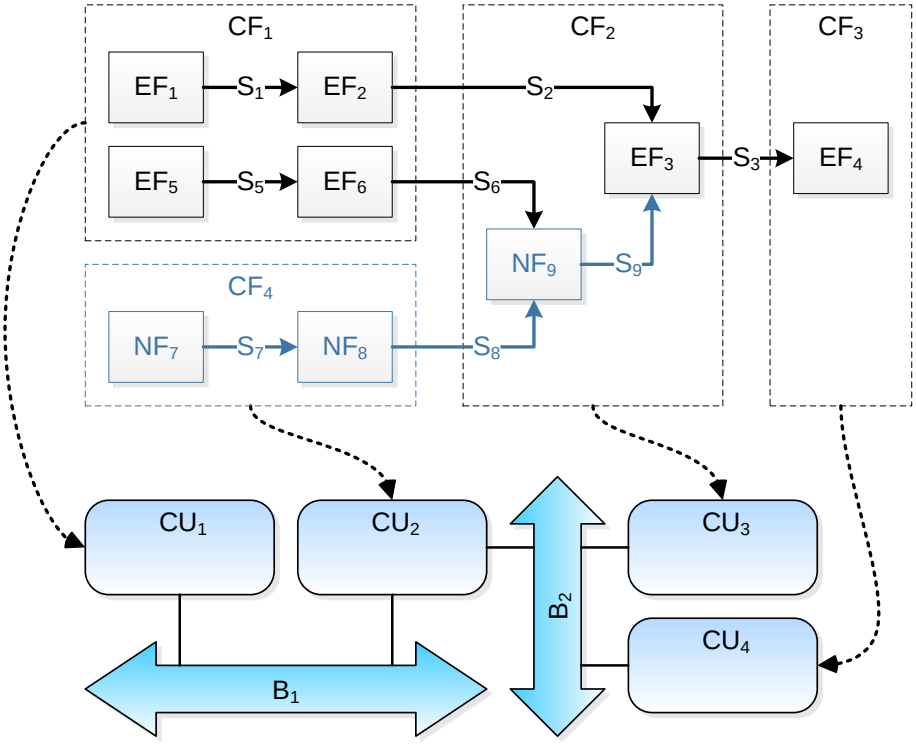

Figure 4-5: Level 1 example from Figure 4-4 with resource providers

On this level we can validate that the actual resources provided by the hardware architecture are enough to still satisfy the event chains from above. There are some cases where one can directly see that there are not be enough resources to even derive a feasible schedule configuration (e.g. the resource consumption is greater than 100%). Other validation cases may produce only warnings indicating that a resource is almost full.

Level 3 then has enough details for a more precise timing simulation as described above. An initial version of all necessary level 3 data may be derived automatically from levels 1 and 2.

Our plan with these three levels is to define new timing V&V methods for levels 1 and 2. For level 3 we will define which required information can be synthesized from the existing data of level 1 and 2, and how this synthesis can be realized.

4.2.7.2 Tool support

Within the XANDAR consortium, the timing aspect of non-functional requirements and their verification and validation is already supported by the Vector TA (Timing Architects) Tool Suite[1]. The Tool Suite, however, is best suited for timing validations on a rather low abstraction level, where there are already a lot of details defined about the system under design (i.e. tasks, priorities, schedulers, processors, cores, etc. are specified).

[1] www.vector.com/int/en/products/products-a-z/software/ta-tool-suite

Planned Extensions: For higher levels of abstraction, there is no tool support available yet. Therefore, we will investigate new timing related V&V methods and their integration into the XANDAR process. These may include simple consistency checks for event and event chain specifications. We will also research upon timing design alternative analysis (e.g. variations in the runnable to task mapping) to provide the end user with viable alternatives for him to choose. We plan to automatically derive feasible scheduler configurations from higher abstraction levels which can then be validated in the Tool Suite.

4.2.8 Safety

In the XANDAR project, the fulfilment of reliability-related safety requirements is covered by a pattern-based approach. Patterns describe transformations of the system model or SWC code and are mapped to system components during the system design step of the development process. This mapping is then used during system generation to apply the respective transformations.

During system design, system developers model system elements, such as logical functions, software components, and hardware elements, to describe the target system architecture. In doing so, they assign safety patterns to system elements to fulfil non-functional requirements. This mapping is derived based on the non-functional requirements towards the system and – where necessary – safety analyses. In this development step, verification is needed to ensure that the pattern assignment is done in a complete and correct way, i.e. that all relevant requirements have been considered and that suitable patterns have been selected to meet these requirements. To do so, especially formal static verification, model inspection, and model (consistency) checking approaches will be considered.

During system generation, which is also referred to as the target-specific generation and analysis procedure, these patterns are applied. Depending on the pattern, this may include code transformation, code generation, model transformations, and the derivation of additional restrictions for subsequent system generation steps, e.g. scheduling or hardware mapping. At this stage, verification approaches need to ensure that these transformations were applied in a correct way and achieve the intended increase in safety. Such approaches may either be done using additional verification steps during the development process (especially formal static verification) or by meta-verification of the pattern itself.

4.2.9 Security

Based on the two XANDAR application use cases defined in D1.1, the following are the non-functional security V&V activities for each:

- Resilient Avionic Architecture
    a. Ensure integrity while data-in-motion
    b. Ensure integrity and authenticity of program code
- Autonomous System with Integrated Machine Learning Applications
    a. Ensure integrity while data-in-motion
    b. Ensure integrity and authenticity of program code
    c. Ensure detection of runtime manipulation

Table 4-1 lists the driven verification and validation requirements for each use-case driven security requirement.

Non-Functional Requirements		
Security	**Verification**	**Validation**
Ensure integrity while data-in-motion	Does the XANDAR Platform/Security Kernel/ Security Pattern and their Interfaces identified as safety/security-critical, **cover/implement** the data-in-motion security requirements?	Does the XANDAR Platform/Security Kernel/Security Patterns and their Interfaces identified as safety/security-critical, **meet/achieve** the data-in-motion security requirements?
Ensure integrity and authenticity of program code	Does the XANDAR Platform and Software system **cover/implement** firmware confidentiality, integrity, and authentication during the platform/system boot process?	Does the XANDAR Platform and Software system **meet/achieve** firmware confidentiality, integrity, and authentication during the system boot process?
Ensure detection of runtime manipulation	Does the XANDAR Platform and Software system **cover/implement** code integrity during software update process?	Does the XANDAR Platform and Software system **meet/achieve** code integrity during software update process?

Table 4-1: Non-functional security verification and validation requirements

The following verification and validation methods and techniques will be considered (as appropriate):

- Model inspection, Model checking, Model consistency checking
- Static code analysis
- Behavioural/Functional Simulation
- Formal static verification
- Dynamic code analysis
- Runtime testing/validation

Since various XANDAR design and implementation decisions (hardware/software architecture and runtime system) are not finalised, the presented security verification and validation requirements are subject to further research during the project.

4.3 Task 5.4: Conducting V&V & meta-V&V in different levels (Lead: QUB, M7-M24)

4.3.1 Overview and goal of the task

T5.4 will combine the V&V techniques and tools from: **T5.2** and **T5.3** to leverage V&V for all levels of the V-model (shown in Figure 4-6), including unit testing, integration testing, and system testing. This overall V&V process will account for code-transformations, design methods, and safety/security concept models used by the XANDAR toolchain to establish end-to-end XbC guarantees. To achieve that, the XANDAR toolset prototype and the individual components will be continuously analysed, assessed, and validated by meta-V&V techniques in **T5.4**.

T5.4 will also verify and validate the XANDAR toolchain itself, which we refer to as *meta-V&V*, to prove the envisioned XbC guarantees for generated programs.

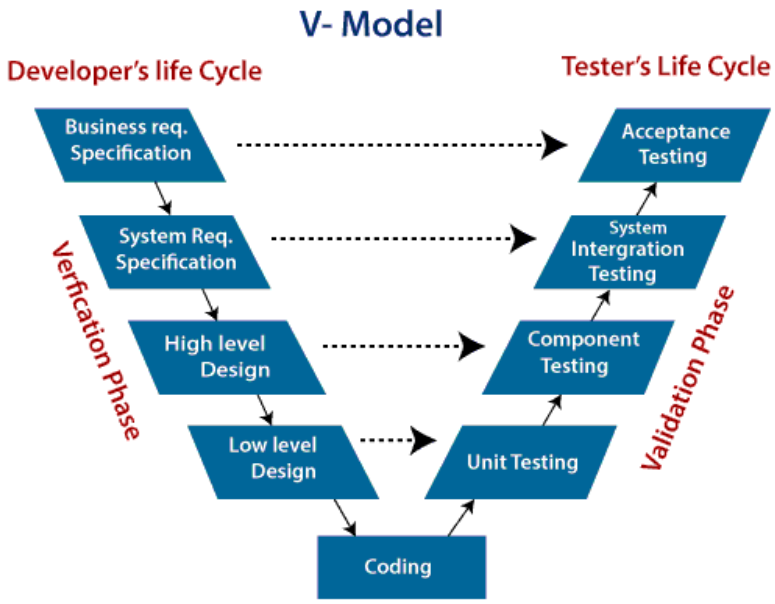

Figure 4-6: The V-model

4.3.2 Outcomes / deliverable of the task (if any)

- D5.5: Multi-Level V&V Toolset (Others, M33, QUB)
- D5.6: Toolchain V&V Report (Report, M33, QUB)

4.3.3 Input(s) of the task

- Task 5.2: V&V of functional requirements (Lead: KIT, M7 - M24)
  - D5.2: Model-based Simulation Framework (Others, M17, KIT)
  - D5.3: V&V Toolset for Functional Correctness (Others, M24, QUB)
- Task 5.3: V&V of non-functional requirements (NFR) (Lead: Vector, M7 - M24)
  - D5.4: V&V Toolset for Non-Functional Properties (Others, M24, VECTOR)

4.3.4 How the task will be implemented

The operational schematic of Task 5.4 is shown in Figure 4-7. One of the two activities in Task 5.4 will be to combine the V&V techniques and tools from T5.2 and T5.3 to conduct V&V in all levels of the V-model, including unit testing, integration testing, and system testing. In that activity, the System Under Test (SUT) is the Autonomous & Distributed Real-time Embedded Software System (ADRESS) system, that is under development using the XANDAR toolchain.

The other activity of Task 5.4 will be to combine, verify and validate the XANDAR toolchain itself, which we refer to as *meta-V&V*, to prove the envisioned XbC guarantees for generated programs. As shown in Figure 4-7, for this second activity, the System Under Test (SUT) is the XANDAR toolchain itself. It is apparent that the two activities of Task 5 are somewhat interrelated, i.e., if a given system under development by the XANDAR toolchain fails any of the V&V checks, then there is an issue somewhere in the XANDAR toolchain. However, V&V of the XANDAR toolchain itself will enable us to check various parts of the toolchain itself, in isolation, and independently form any systems under development, using the toolchain. Such an approach will provide additional confidence in the XANDAR toolchain.

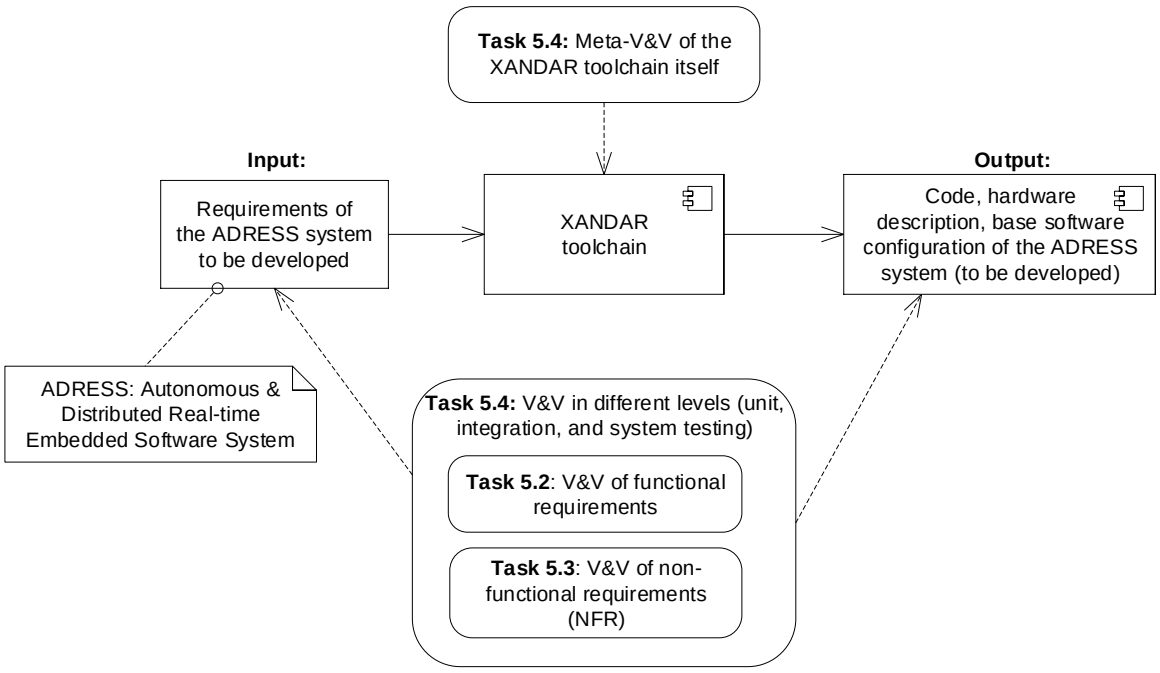

Figure 4-7: Operational schematic of Task 5.4

Meta-V&V activities will be implemented and conducted by developing automated test suites in unit, integration, and system testing levels for the XANDAR toolchain itself. Depending on the programming language of our choice for each component of the XANDAR toolchain, proper choice of unit test frameworks, such as JUnit, CUnit and CppUnit will be made. Our team has past experience in automated testing of large industrial software, e.g., testing Supervisory Control and Data Acquisition (SCADA) systems [84, 99], testing real-time software [61, 100], and testing helicopter simulation software [101].

4.4 Task 5.5: V&V toolset integration (Lead: Vector, M25 – M33)

In the final task of WP5, Task 5.5, we will integrate the V&V toolsets from T5.2, T5.3 and T5.4 with the model-based design tools developed in WP2. This includes, e.g., feedback of the V&V results to the original models, their visualization, and the generation of V&V reports for the end-user.

The main source of models, at least in the early phases of the SDLC, will be Vector's *PREEvision* tool[2]. Within the scope of WP2, new modelling means will be prototypically implemented in PREEvision according to the needs of the use cases from WP1 and the modelling requirements.

In the scope of WP5, we will assume that the necessary works in WP2 has been done to ensure that the necessary data for the V&V of functional and non-functional requirements has been modelled and provided.

Task 5.5 will prototype the Integration of V&V tools with the modelling tools, i.e. ensuring that the tools of the XANDAR process and tool-chain exchange model data without losing crucial

[2] www.vector.com/int/en/products/products-a-z/software/preevision/

information. For instance, Vector plans to exchange timing data and requirements via the AUTOSAR XML format between PREEvision and the TA Tool Suite[3].

Support for other exchange formats may have to be added depending on the use case needs. However, a full prototypical tool integration is an extensive effort with potential risks associated with it. As such, the use and potential extension of existing formats is expected to be more feasible.

[3] www.vector.com/int/en/products/products-a-z/software/ta-tool-suite

5 Mapping WP5 tasks to deliverables, and dependencies with other WPs

To provide traceability of WP5's tasks to its deliverables, we provide a visual map among the tasks and deliverables of WP5 in Figure 5-1. We also provide, in this figure, the input/output dependency links between WP5 and other WPs.

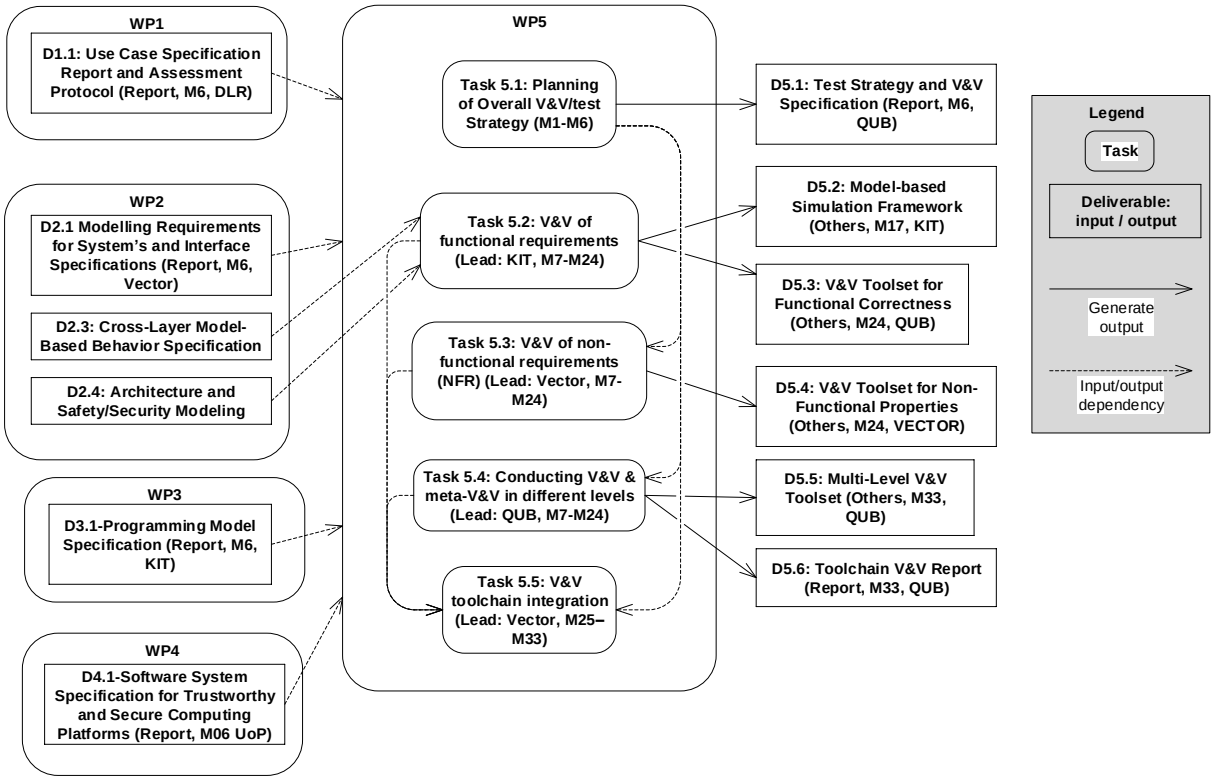

Figure 5-1: Mapping WP5 tasks to its deliverables, and dependencies with other WPs

6 Summary

In summary, using four tasks, WP5 will investigate, develop, and apply methods to validate and verify functional and non-functional requirements of software systems developed by the XANDAR tools. Additionally, the XANDAR methods used by the XANDAR tools will be verified and validated via WP5 to proof XbC guarantees. The verification tools and methods developed in WP5 will be used to evaluate the use-case applications and generate feedback for the end-user partners.

References

[1] S. Easterbrook, "The difference between Verification and Validation," *www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/,* Last accessed: May 2021.

[2] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE,* vol. 95, no. 2, pp. 356-373, 2007.

[3] T. Zurawka, J. Schaeuffele, and R. Carey, *Automotive software engineering*. SAE, 2016.

[4] T. Clarence-Smith, "How Software Will Dominate the Automotive Industry," *https://www.toptal.com/insights/innovation/how-software-will-dominate-the-automotive-industry,* Last accessed: May 2021.

[5] Wikipedia, "Avionics software," *https://en.wikipedia.org/wiki/Avionics_software,* Last accessed: May 2021.

[6] C. Anderson and M. Dorfman, *Aerospace software engineering*. American Institute of Aeronautics, 1991.

[7] D. Locke, L. Lucas, and J. B. Goodenough, "Generic Avionics Software Specification," *Software Engineering Institute (SEI) of Carnegie Mellon University, https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=11181,* Last accessed: May 2021.

[8] C. Gehman, "How to Develop Software For Cars Faster (And Drive Innovation in the Automotive Industry)," *https://www.perforce.com/blog/vcs/software-in-cars-how-to-drive-innovation-automotive-industry,* Last accessed: May 2021.

[9] G. Travis, "How the Boeing 737 Max disaster looks to a software developer," *IEEE Spectrum,* vol. 18, 2019.

[10] P. Johnston and R. Harris, "The Boeing 737 MAX Saga: Lessons for Software Organizations," *Software Quality Professional Magazine,* vol. 21, no. 3, 2019.

[11] A. Hinkov, "Bugs in automotive software," *https://blog.caristaapp.com/bugs-in-automotive-software-5adf9138d71b,* Last accessed: May 2021.

[12] C. Mckeon, "The Recent, Worrisome Spike of Software-Based Defects in Vehicles," *https://autotrends.org/2020/02/10/the-recent-worrisome-spike-of-software-based-defects-in-vehicles/,* Last accessed: May 2021.

[13] A. Zeichick, "Automotive Software Engineering Defects on the Rise," *https://www.parasoft.com/automotive-software-engineering-defects-on-the-rise/,* Last accessed: May 2021.

[14] N. Valery, "Cars and software bugs," *https://www.economist.com/babbage/2010/05/16/techview-cars-and-software-bugs,* Last accessed: May 2021.

[15] IT Pro team, "Driverless Cars: Uber car involved in fatal crash had software flaws," *https://www.itpro.co.uk/strategy/27302/driverless-cars-news,* Last accessed: May 2021.

[16] T. Lee, "Software bug led to death in Uber's self-driving crash," *https://arstechnica.com/tech-policy/2018/05/report-software-bug-led-to-death-in-ubers-self-driving-crash/,* Last accessed: May 2021.

[17] G. Myers, *The Art of Software Testing*. John Wiley, 1979.

[18] B. Chess and J. West, *Secure programming with static analysis*. Pearson Education, 2007.

[19] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.

[20] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[21] C. A. R. Hoare, "Proof of correctness of data representations," in *Programming methodology*: Springer, 1978, pp. 269-281.

[22] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj, "Using abstraction and model checking to detect safety violations in requirements specifications," *IEEE Transactions on software engineering,* vol. 24, no. 11, pp. 927-948, 1998.

[23] W. Wögerer, "A survey of static program analysis techniques," Citeseer, 2005.

[24] A. Gosain and G. Sharma, "Static analysis: A survey of techniques and tools," in *Intelligent Computing and Applications*: Springer, 2015, pp. 581-591.

[25] M. Pistoia, S. Chandra, S. J. Fink, and E. Yahav, "A survey of static analysis methods for identifying security vulnerabilities in software systems," *IBM Systems Journal,* vol. 46, no. 2, pp. 265-288, 2007.

[26] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena, "A survey of regular model checking," in *International Conference on Concurrency Theory*, 2004: Springer, pp. 35-48.

[27] G. Agha and K. Palmskog, "A survey of statistical model checking," *ACM Transactions on Modeling and Computer Simulation (TOMACS),* vol. 28, no. 1, pp. 1-39, 2018.

[28] S. Edelkamp, V. Schuppan, D. Bošnački, A. Wijs, A. Fehnker, and H. Aljazzar, "Survey on directed model checking," in *International Workshop on Model Checking and Artificial Intelligence*, 2008: Springer, pp. 65-89.

[29] V. D'silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, no. 7, pp. 1165-1178, 2008.

[30] M. Bradley, F. Cassez, A. Fehnker, T. Given-Wilson, and R. Huuck, "High performance static analysis for industry," *Electronic Notes in Theoretical Computer Science,* vol. 289, pp. 3-14, 2012.

[31] D. Baca, B. Carlsson, K. Petersen, and L. Lundberg, "Improving software security with static automated code analysis in an industry setting," *Software: Practice and Experience,* vol. 43, no. 3, pp. 259-279, 2013.

[32] B. A. Wichmann, A. Canning, D. Clutterbuck, L. Winsborrow, N. Ward, and D. Marsh, "Industrial perspective on static analysis," *Software Engineering Journal,* vol. 10, no. 2, pp. 69-75, 1995.

[33] A. Pakonen, T. Tahvonen, M. Hartikainen, and M. Pihlanko, "Practical applications of model checking in the Finnish nuclear industry," in *10th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC & HMIT 2017). American Nuclear Society*, 2017, pp. 1342-1352.

[34] O. Grumberg and H. Veith, *25 years of model checking: history, achievements, perspectives*. Springer, 2008.

[35] A. Fantechi and S. Gnesi, "On the adoption of model checking in safety-related software industry," in *International Conference on Computer Safety, Reliability, and Security*, 2011: Springer, pp. 383-396.

[36] A. Biere, E. Clarke, R. Raimi, and Y. Zhu, "Verifying safety properties of a PowerPC−microprocessor using symbolic model checking without BDDs," in *International Conference on Computer Aided Verification*, 1999: Springer, pp. 60-71.

[37] M. Bennion and I. Habli, "A candid industrial evaluation of formal software verification using model checking," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 175-184.

[38] D. M. Russinoff, "A mechanically checked proof of correctness of the AMD K5 floating point square root microcode," *Formal Methods in System Design,* vol. 14, no. 1, pp. 75-125, 1999.

[39] H. Barendregt, "Proofs of correctness in mathematics and industry," *Wiley Encyclopedia of Computer Science and Engineering,* 2007.

[40] D. Weber-Wulff, "Selling formal methods to industry," in *International Symposium of Formal Methods Europe*, 1993: Springer, pp. 671-678.

[41] A. Cimatti, F. Giunchiglia, P. Traverso, and A. Villafiorita Monteleone, "Run-time result formal verification of safety critical software: an industrial case study," in *FLoC99 Workshop on Run-Time Result Verification*, 1999.

[42] K. Patel and R. M. Hierons, "A mapping study on testing non-testable systems," *Software Quality Journal,* vol. 26, no. 4, pp. 1373-1413, 2018.

[43] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering,* 2020.

[44] X. Huang *et al.*, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review,* vol. 37, p. 100270, 2020.

[45] H. B. Braiek and F. Khomh, "On testing machine learning programs," *Journal of Systems and Software,* vol. 164, p. 110542, 2020.

[46] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empirical Software Engineering,* vol. 25, no. 6, pp. 5193-5254, 2020.

[47] V. Garousi, M. Felderer, Ç. M. Karapıçak, and U. Yılmaz, "Testing embedded software: A survey of the literature," *Information and Software Technology,* vol. 104, pp. 14-45, 2018.

[48] V. Garousi, M. Felderer, Ç. M. Karapıçak, and U. Yılmaz, "What we know about testing embedded software," *IEEE Software,* vol. 35, no. 4, pp. 62-69, 2018.

[49] M. Borg *et al.*, "Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry," *arXiv preprint arXiv:1812.05389,* 2018.

[50] H. Altinger, F. Wotawa, and M. Schurius, "Testing methods used in the automotive industry: Results from a survey," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, 2014, pp. 1-6.

[51] A. Dadwal, H. Washizaki, Y. Fukazawa, T. Iida, M. Mizoguchi, and K. Yoshimura, "Prioritization in Automotive Software Testing: Systematic Literature Review," in *QuASoQ@ APSEC*, 2018, pp. 52-58.

[52] M. Markthaler *et al.*, "Improving model-based testing in automotive software engineering," in *IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track* 2018, pp. 172-180.

[53] E. Bringmann and A. Krämer, "Model-based testing of automotive systems," in *2008 1st international conference on software testing, verification, and validation*, 2008: IEEE, pp. 485-493.

[54] K. Lamberg, M. Beine, M. Eschmann, R. Otterbach, M. Conrad, and I. Fey, "Model-based testing of embedded automotive software using MTest," *SAE transactions,* pp. 132-140, 2004.

[55] M. Kastebo and V. Nordh, "Model-based security testing in automotive industry," 2017.

[56] Y. Choi, "Model checking Trampoline OS: a case study on safety analysis for automotive software," *Software Testing, Verification and Reliability,* vol. 24, no. 1, pp. 38-60, 2014, doi: https://doi.org/10.1002/stvr.1482.

[57] V. Garousi, A. B. Keleş, Y. Balaman, Z. Ö. Güler, and A. Arcuri, "Model-based testing in practice: An experience report from the web applications domain," *https://arxiv.org/abs/2104.02152*, 2021.

[58] Ş. Şentürk, A. Akın, A. B. Karagöz, and V. Garousi, "Model-based testing in practice: An experience report from the banking domain," in *Proceedings of the Turkish National Software Engineering Symposium (UYMS)*, 2019.

[59] D. Akdur, V. Garousi, and O. Demirörs, "A survey on software modeling and model-driven techniques in embedded systems engineering: results from Turkey," in *Proceedings of the Turkish National Software Engineering Symposium (UYMS)*, 2015.

[60] V. Garousi, "UML Model-driven Detection of Performance Bottlenecks in Concurrent Real-Time Software," in *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2010, pp. 317-323.

[61] V. Garousi, L. Briand, and Y. Labiche, "Traffic-aware Stress Testing of Distributed Real-Time Systems Based on UML Models using Genetic Algorithms," *Elsevier Journal of Systems and Software, Special Issue on Model-Based Software Testing,* vol. 81, no. 2, pp. 161-185, 2008.

[62] V. Garousi, "A Formalism for Arrival Time Analysis of Real-Time Tasks based on UML Models," in *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2008, pp. 1575-1579.

[63] R. Chen, F. B. Bastani, and T.-W. Tsao, "On the reliability of AI planning software in real-time applications," *IEEE Transactions on Knowledge and Data Engineering,* vol. 7, no. 1, pp. 4-13, 1995.

[64] D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, and C. Paul, "The challenges of real-time AI," *Computer,* vol. 28, no. 1, pp. 58-66, 1995.

[65] F. Wotawa, "On the Importance of System Testing for Assuring Safety of AI Systems," in *AISafety@ IJCAI*, 2019.

[66] J. McDermid and Y. Jia, "Safety of Artificial Intelligence: A Collaborative Model."

[67] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," *arXiv preprint arXiv:1606.06565,* 2016.

[68] R. V. Yampolskiy, *Artificial intelligence safety and security*. CRC Press, 2018.

[69] M. Glinz, "On non-functional requirements," in *IEEE International Requirements Engineering Conference*, 2007: IEEE, pp. 21-26.

[70] International Organization for Standardization (ISO), "ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE)," *https://www.iso.org/standard/35733.html,* Last accessed: May 2021.

[71] L. Krawczyk, M. Bazzal, R. P. Govindarajan, and C. Wolff, "Model-based timing analysis and deployment optimization for heterogeneous multi-core systems using eclipse app4mc," in *International Conference on Model Driven Engineering Languages and Systems Companion*, 2019, pp. 44-53.

[72] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller, "Timing modeling and analysis for AUTOSAR-based software development-a case study," in *Design, Automation & Test in Europe Conference & Exhibition*, 2010: IEEE, pp. 642-645.

[73] P. Cuenot *et al.*, "Developing automotive products using the east-adl2, an autosar compliant architecture description language," in *4th International Congress ERTS 2008*, 2008.

[74] M. Mehrabian *et al.*, "Timestamp temporal logic (TTL) for testing the timing of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 16, no. 5s, pp. 1-20, 2017.

[75] M. Z. Iqbal, S. Ali, T. Yue, and L. Briand, "Experiences of applying UML/MARTE on three industrial projects," in *International Conference on Model Driven Engineering Languages and Systems*, 2012: Springer, pp. 642-658.

[76] M.-A. Peraldi-Frati, D. Karlsson, A. Hamann, S. Kuntz, and J. Nordlander, "The TIMMO-2-USE project: Time modeling and analysis to use," in *Embedded Real Time Software and Systems (ERTS2012)*, 2012.

[77] B. Broekman and E. Notenboom, *Testing embedded software*. Pearson Education, 2003.

[78] I. Kendall and R. Jones, "An investigation into the use of hardware-in-the-loop simulation testing for automotive electronic control systems," *Control Engineering Practice,* vol. 7, no. 11, pp. 1343-1356, 1999.

[79] S. S. You and D. Fricke, "Advances of virtual testing and hybrid simulation in automotive performance and durability evaluation," *SAE International Journal of Materials and Manufacturing,* vol. 4, no. 1, pp. 98-110, 2011.

[80] C. Faure, M. B. Gaid, N. Pernet, M. Fremovici, G. Font, and G. Corde, "Methods for real-time simulation of Cyber-Physical Systems: application to automotive domain," in *2011 7th International Wireless Communications and Mobile Computing Conference*, 2011: IEEE, pp. 1105-1110.

[81] J. Wagner and J. Furry, "A real time simulation environment for the verification of automotive electronic controller software," *International Journal of Vehicle Design,* vol. 13, no. 4, pp. 365-377, 1992.

[82] V. Garousi, "Fault-driven Stress Testing of Distributed Real-Time Software based on UML Models," *Wiley InterScience Journal of Software Testing, Verification and Reliability (STVR),* vol. 21, no. 2, pp. 101-124, 2011.

[83] V. Garousi, "A Genetic Algorithm-based Stress Test Requirements Generator Tool and its Empirical Evaluation," *IEEE Transactions on Software Engineering, Special Issue on Search-Based Software Engineering (SBSE),* vol. 36, no. 6, pp. 778-797, 2010.

[84] V. Garousi, "Experience and Challenges with UML-Driven Performance Engineering of a Distributed Real-Time System," *Journal on Information and Software Technology,* vol. 52, no. 5, pp. 625–640, 2010.

[85] S. Bayer, T. Enderle, D.-K. Oka, and M. Wolf, "Automotive security testing—the digital crash test," in *Energy Consumption and Autonomous Driving*: Springer, 2016, pp. 13-22.

[86] I. Pekaric, C. Sauerwein, and M. Felderer, "Applying Security Testing Techniques to Automotive Engineering," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1-10.

[87] S. Bayer, T. Enderle, D.-K. Oka, and M. Wolf, "Security crash test-practical security evaluations of automotive onboard it components," *Automotive-Safety & Security 2014,* 2015.

[88] J. Dürrwang, J. Braun, M. Rumez, R. Kriesten, and A. Pretschner, "Enhancement of automotive penetration testing with threat analyses results," *SAE International Journal of Transportation Cybersecurity and Privacy,* vol. 1, no. 11-01-02-0005, pp. 91-112, 2018.

[89] D. S. Fowler, J. Bryans, S. A. Shaikh, and P. Wooderson, "Fuzz testing for automotive cyber-security," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018: IEEE, pp. 239-246.

[90] G. Spanoudakis, C. Kloukinas, and K. Androutsopoulos, "Towards security monitoring patterns," in *Proceedings of the 2007 ACM symposium on Applied computing*, 2007, pp. 1518-1525.

[91] R. Khoury and N. Tawbi, "Which security policies are enforceable by runtime monitors? a survey," *Computer Science Review,* vol. 6, no. 1, pp. 27-45, 2012.

[92] S. Das and S. Shiva, "CoRuM: collaborative runtime monitor framework for application security," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018: IEEE, pp. 201-206.

[93] L. Huang and E.-Y. Kang, "Formal verification of safety & security related timing constraints for a cooperative automotive system," in *International Conference on Fundamental Approaches to Software Engineering*, 2019: Springer, pp. 210-227.

[94] A. M. Shaaban, C. Schmittner, T. Gruber, A. B. Mohamed, G. Quirchmayr, and E. Schikuta, "Ontology-based model for automotive security verification and validation," in *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services*, 2019, pp. 73-82.

[95] G. Xie, G. Zeng, Y. Liu, J. Zhou, R. Li, and K. Li, "Fast functional safety verification for distributed automotive applications during early design phase," *IEEE Transactions on Industrial Electronics,* vol. 65, no. 5, pp. 4378-4391, 2017.

[96] J. Nilsson, *Computational verification methods for automotive safety systems*. Chalmers University of Technology, 2014.

[97] F. A. da Silva, A. C. Bagbaba, S. Hamdioui, and C. Sauer, "Efficient methodology for ISO26262 functional safety verification," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2019: IEEE, pp. 255-256.

[98] A. Sherer, J. Rose, and R. Oddone, "Ensuring functional safety compliance for ISO 26262," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1-3.

[99] S. A. Jolly, V. Garousi, and M. M. Eskandar, "Automated Unit Testing of a SCADA Control Software: An Industrial Case Study based on Action Research," in *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2012, pp. 400-409.

[100] G. Urul, V. Garousi, and G. Urul, "Test Automation for Embedded Real-time Software: An Approach and Experience Report in the Turkish Industry," in *Proceedings of the Turkish National Software Engineering Symposium (UYMS)*, 2014.

[101] V. Garousi *et al.*, "Experience in automated testing of simulation software in the aviation industry," *IEEE Software,* July/August 2019.