XANDAR: Verification & Validation Approach for Safety-critical Systems

Balmukund Sonigara*, Sakir Sezer*, Fahad Siddiqui*, Raphael Weber†, Konstantinos Antonopoulos‡, Christos Panagiotou§, Christos P. Antonopoulos‡, Georgios Keramidas‡, Nikolaos Voros‡, Sena Busra Yengec-Tasdemir*, Henry Hui*, Kieran McLaughlin*
*Queen's University Belfast, United Kingdom
†Vector Informatik GmbH, Germany
‡ University of Peloponnese, Greece
§ AVN Innovative Technology Solutions Limited, Cyprus
*b.sonigara, s.sezer, f.siddiqui, s.yengectasdemir, h.hui, kieran.mclaughlin@qub.ac.uk,
†raphael.weber@vector.com
‡ k.antonop, ch.antonop, g.keramidas, voros@esdalab.ece.uop.gr
§chpanag@avntechgroup.com

***Abstract*—The integration of connected and autonomous technologies in safety-critical brought significant system design challenges. These systems are constantly evolving and becoming more complex. With their connection to the cloud and the internet, these safety-critical systems are now exposed to greater risks of cyber-attacks, which poses new challenges to their safety, reliability and resilience. To approach these complex system design challenges, this paper proposes XANDAR's Verification & Validation strategy using Static Analysis, Timing Analysis, Model-in-loop and Network simulation tool. To ensure functional correctness, the proposed XANDAR Verification and Validation approach utilizes early integration of simulation and static analysis techniques during the development cycle. This proposed approach differs from existing methods by emphasizing early integration, rather than applying it to later stages of development cycle to begin verification. In addition, the proposed approach utilizes timing analysis to ensure non-functional timing aspects meet the timing requirements. The approach applies tools such as Polyspace Bug Finder and Code Prover for static analysis, Timing Architect for timing analysis, NS3 simulator for network architecture simulation. The proposed approach aims to ensure system safety and security through a rigorous and comprehensive verification process. These verification approaches will be validated by applying it to automotive and avionics use cases.**

***Index Terms*—Verification, Validation, Static Analysis, Model-in-Loop Simulation, Timing Analysis and Network Simulation.**

I. Introduction

With the emergence of modern technologies, safety-critical systems like automotive and avionics systems are continuously evolving and increasing in complexity. They are now connected to cloud/internet which poses greater risks of cyber-attacks and presents new challenges for their safety and reliability [1]. To ensure the safety and reliability of these systems, a rigorous approach to design, development, *Verification and Validation* (V&V) is necessary [2], [3]. In the past, insufficient V&V of embedded software systems, particularly in industries such as automotive and aviation, have resulted in numerous severe consequences. For example, the crashes of two Boeing 737 MAX airplanes in 2018 and 2019 were attributed to various issues such as flaws in the software and sensors. This incident, along with others, has been extensively covered in a range of new articles, technical reports, and academic papers, which have delved into the root causes of the problems [4], [5]. Similarly, the automotive industry has also experienced numerous crashes and accidents resulting from software defects that have made their way into production systems due to insufficient V&V during the system development process [6], [7]. These safety critical systems require a rigorous V&V processes in the software development life cycle to ensure safety and reliability. Thus, the outputs of these processes provide evidence that safety requirements have been satisfactorily met [8]. Yet, with the increasing complexity of software designs because of new features addition and AI/ML models which exhibits non-deterministic behavior [9] being part of these systems, verification is becoming challenging.

To address these challenges, the XANDAR project is one such effort that aims to deploy static analysis based verification technique for the development of safety-critical systems in the automotive and avionics industries to guarantee functional correctness. The XANDAR project is focused on creating a robust software tool chain that caters to the industry's needs for quick prototyping of autonomous and inter-operable embedded systems. This tool chain includes all the stages, from requirements analysis to code integration on the target, and V&V. The project uses a model-based system design approach and leverage automatic model synthesis and software parallelization techniques to meet specific non-functional requirements. This lays the foundation for a new paradigm known as X-by-Construction (XbC) [3], ensuring systems are constructed with safety and security capabilities by-design [10], [11], [12].

In this regard, this paper proposes XANDAR's V&V strategy using Static Analysis, Timing Analysis, Model-in-loop and Network simulation tools in section IV. Section II & III presents the essential background of the tools used in V&V strategy and related work on verification tools and approaches. Section IV introduces the V&V approach for XANDAR.

II. Background

Traditional V&V approach as illustrated in Fig. 1 includes software and system testing methods (i.e. unit, integration and systems testing). Current verification methodologies lack formal foundations and proof of soundness, which hinders their effectiveness in ensuring the safety and security of the system. In an ideal scenario, an efficient V&V approach should guarantee the safety and security of the system being tested. Apart from testing, static analysis [13], model checking [14], and proof of correctness are some other V&V approaches.

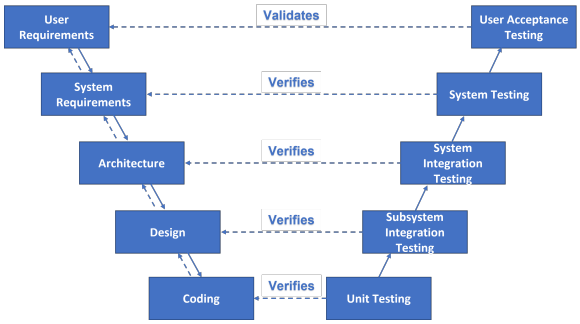

Fig. 1. Traditional V&V process.

Static analysis is one of the techniques used to analyze the source code of a program without actually executing it. This technique is especially useful in the development of safety-critical systems, where it is crucial to identify and eliminate potential defects early in the development process. Static code analysis is a highly recommended software unit verification method for *Automotive Safety Integrity Level* (ASIL) for ensuring software safety in the automotive industry, as defined by ISO 26262 [15]. This standard, along with DO-178C in the avionics industry, guides system development processes and activities to achieve optimal safety levels.

To automate the source code anaylsis and to reduce the manual efforts required, static analysis tools are used. These tools can analyze millions of lines of code and provide a report of potential issues. One of the popular tools used for static code analysis in safety-critical domains is Polyspace from MathWorks [16], which provides a precise and efficient solution for identifying and addressing software bugs and vulnerabilities. The generated report includes details such as the location of the issue, severity level, and recommended actions. The developers can use this report to fix the issues and improve the software quality.

Timing requirements are among the most critical non functional requirements 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. In XANDAR's V&V approach, simulation testing plays a crucial role in verifying and validating timing requirements, given their significance in real-time embedded software systems. Funcsim tool is an implementation of simulation based concept using the Ptolemy II framework. In this testing approach a behavior specification and simulation methodology based on the *Logical Execution Time* (LET) paradigm allows for the explicit specification of non-negligible execution times and efficient compilation to complex target platforms [17]. The methodology combines a metamodel capturing the software architecture of a system with a code-based programming model to describe the functional behavior of individual software entities. The simulation strategy translates the specified behavior into deterministic discrete-event (DE) simulations that can be executed in custom environments [18]. NS-3 is a discrete event network simulator, which is widely used by the industry and the research community as a tool of choice for network performance and evaluation simulations. The simulator has a multi-layered framework, while each layer depends on its lower layers. Section III discuss methods of verifying and validating safety-critical systems.

III. Related Work

There are a plethora of commercial and open-source tools that can be used for static analysis. However, many open-source tools lack support for coding standards, which can pose a limitation when performing static analysis on source code for safety-critical systems. Coding standards consist of guidelines and rules that outline how software should be written and structured. They are particularly crucial in safety-critical systems, where software must be consistent, readable, and maintainable. Adhering to coding standards can also help prevent common programming errors and defects, and promote the development of high-quality software.

Astrée, is a popular static code analyzer that is capable of proving the absence of runtime errors and invalid concurrent behavior in safety-critical software written or generated in C or C++ [19]. Another tool, Coverity, is a static analysis tool that supports multiple languages and can be integrated with multiple integrated development environments (IDE) [20].Another enterprise-grade tool is Helix QAC, which is designed specifically for embedded software and supports the MISRA, CERT, and AUTOSAR coding standards [21]. LDRA is another tool suite that includes static analysis (TBVISION) for various standards, including MISRA C & C++, JSF++ AV, CWE, CERT C, CERT C++, and Custom Rules [22]. PC-lint Plus is a static analysis tool that finds defects by employing data flow analysis, abstract interpretation, value tracking, and other various mechanisms. These tools can generate reports consisting of defects/bugs found, code-rule violations, and code quality metrics including cyclomatic complexity [23].

MathWorks verification tool, Polyspace Bug Finder [24], is a powerful tool for identifying a wide range of defects in C and C++ embedded software. The tool uses static analysis, including semantic analysis, to analyze software control flow, data flow, and inter-procedural behavior, enabling it to identify runtime errors, concurrency issues, security vulnerabilities, and other defects. By detecting defects early in the development process, Polyspace Bug Finder allows developers to address them before they become more costly and difficult to fix. Polyspace Code Prover, a verification tool from MathWorks, utilizes formal methods to prove the absence

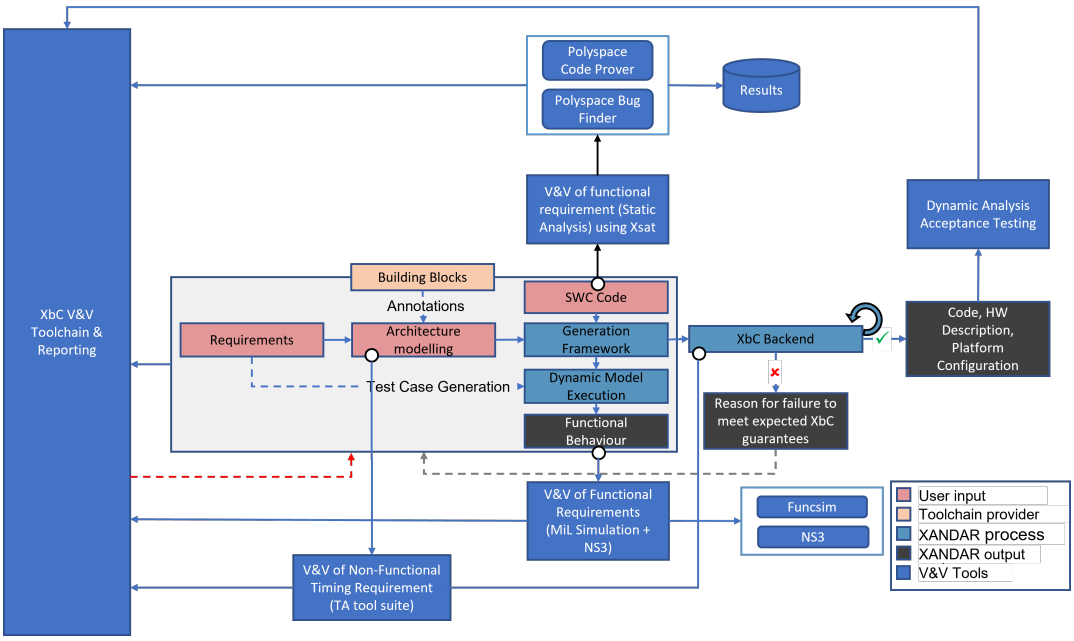

Fig. 2. V&V strategy mapped to the XANDAR development process.

of certain runtime errors in C and C++ source code. It does not require program execution, code instrumentation, or test cases to produce results. Instead, it applies static analysis and abstract interpretation to analyze the code. The tool can analyse both handwritten and generated code.

It is important to note that the choice of a static analysis tool depends on the specific needs of the project and the software development process. It is recommended to evaluate multiple tools and select the one that best fits the project requirements. Additionally, open-source static analysis tools are available but may not provide support for coding standards, which can be a limitation when using these tools for static analysis of source code in safety-critical systems.

The TA Tool Suite is a comprehensive collection of software products designed for designing, simulating, and verifying embedded real-time systems. It provides a unified solution that covers the non-functional aspect of timing throughout the entire development cycle. The suite of tools facilitates in-depth analysis of timing behavior, making it easier to integrate software into an ECU. This streamlined process improves the efficiency and responsiveness of embedded real-time systems. Section IV introduces the XANDAR V&V approach and how it will be mapped to the XANDAR process.

IV. XANDAR Verification & Validation Approach

To ensure the functional correctness and safety of embedded software for automotive and avionics use cases, we propose a verification approach that uses a combination of static analysis, model-in-loop simulation, and time analysis technique. Fig. 2 illustrates the XANDAR V&V Strategy by mapping it to XANDAR development process [10], [17].

A. Static Analysis using Polyspace Tools

Currently, for XANDAR project, we have exclusively considered the use of MathWorks' Polyspace tools for static analysis. It is important to note that these tools can be substituted with other available static analysis tools if deemed appropriate. As a part of the XANDAR project, a static analysis toolset automation script ("Xsat") was developed to invoke the Polyspace Bug Finder and Code Prover tools from MathWorks. These tools are commercial static analysis products designed to reveal runtime bugs and ensure functional correctness of software code. By automating the process of invoking these tools, Xsat makes it easier for developers to perform static analysis on their code and identify potential defects early in the development process.

Polyspace Bug Finder also checks compliance with coding rule standards such as MISRA C, MISRA C++, AUTOSAR C++14, CERT C, CERT C++, and custom naming conventions. By ensuring compliance with coding standards, it helps to maintain code quality and avoid potential issues that could arise from non-compliant code. The tool generates detailed reports that include information about the defects and bugs it has found, code-rule violations, and code quality metrics such as cyclomatic complexity. These reports provide valuable insights into the quality of the code and allow developers to prioritize and focus their efforts on the most critical issues.

Polyspace Code Prover uses a color-coding system to indicate the status of each operation in the code. The colors signify whether an operation is free of runtime errors, proven to fail, unreachable, or unproven. Additionally, the tool displays range information for variables and function return values and can prove whether certain variables exceed specified range limits. With the ability to verify the absence of runtime errors,

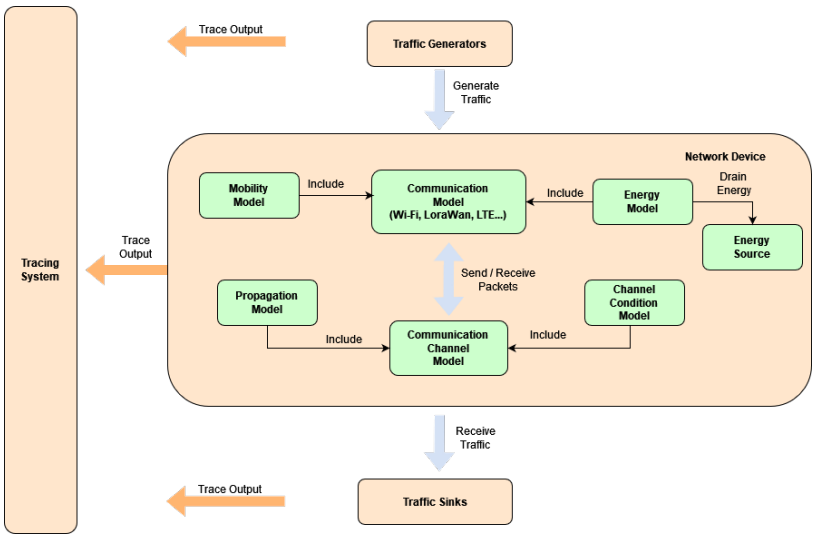

Fig. 3. High Level Architecture of NS3 Network Simulator

Polyspace Code Prover allows developers to identify and fix potential issues in their code early in the development process. This helps reduce the likelihood of costly bugs and defects in the final product, improving overall software quality. Moreover, the results of the tool can be published on a dashboard, which enables tracking quality metrics and compliance with software quality objectives.

The ”Xsat” script is particularly useful when we need to integrate the Polyspace toolset in continuous integration/deployment environment for automated static analysis verification process.The script can be integrated into the software build process and used to perform continuous static analysis on the code as it is being developed.

B. Model-in-Loop Simulation using Fucsim Tool

The funcsim tool, an integral component of the XANDAR V&V tool set, focuses primarily on verifying the functional and timing behavior of systems [18]. Ptolemy II [25] serves as the simulation backend for this tool. By combining logical execution time (LET) abstraction and a code-based approach, it performs discrete event simulation to assess system responses to specific input stimuli.

C. Timing Analysis using TA tool suite and PREEvision

Considering timing properties and requirements at an early stage of the design may prove beneficial since some timing design errors can be avoided from the start. This can be achieved through a two step process. In the first step during architecture modelling the system should be considered from the outside and identify how often data should be sent (sensor input) and produced (input for actuators). Also, from the outside we take a “gray-box” look into the system architecture and identify critical data flows that we would like to observe (this can later be used to derive trace points). Based on these dataflows we formulate timing requirements and optionally distribute them to parts of these dataflows. In the second step of the timing design, we look at the atomic functions themselves, and define time budgets for each of them. If multiple data flows and varying data input frequencies pass through an atomic function, this must be considered when defining the time budgets. The main difference is that the first step is concerned with the outside view of the functions and the data flow between them. In the second step we take a closer look at the atomic functions individually. Once either of these two steps is completed, we can validate the consistency of the data flows and their requirements, and ensure that the function time budgets are in line with these requirements. Both these early timing validations are prototypically implemented in the PREEvision [26] modelling tool. In a later step (in the XbC Backend), there is more information about the hardware of the system, and how the provided hardware resources are shared among different application parts. The above mentioned top-level timing requirements still have to be fulfilled, but now they may have been refined. Given enough information about the execution times of the application parts (as a model), we can now simulate the timing behaviour of the embedded system including the resource sharing overheads caused by, e.g., the operating system, driver parts, communication controllers, etc. This timing simulation can be done in the TA Tool Suite [27] which provides an evaluation report as a result. If this report indicates no timing errors, the XbC workflow can continue, otherwise the design has to be corrected.

D. Network Simulation using NS3 network simulator

A high-level architecture of the NS3 simulator is depicted in Fig. 3. Starting from the northbound of the architecture, a traffic generator is used to represent different application specific scenarios, feeding in different traffic type patterns to the simulations. The main component in the NS3 network simulator is the network device, where different models (auxiliary and communication models) are defined. Each network device consists of a Communication model, Mobility model,

Energy model, Propagation models, Communication channel models and Channel condition models. At the southbound of the architecture operate the traffic sinks, where the generated traffic is received. The final objective of each simulation is to export and collect the simulation results (westbound of the architecture) for further analysis and semantic enrichment. NS3 simulator provides a horizontal tracing system, allowing the collection of traces at different levels.

1) **Traffic generators/sink models** plays critical role to model and simulate real-world scenarios to analyse how much data is generated and circulated in the network. Two parameters needed to dynamically configure the available traffic models are the traffic size and frequency. These traffic can be categorized into the following categories: Constant traffic, Uniform traffic and finally Poisson traffic. Poisson distribution traffic generation is close to traffic generated in real world environments.
2) **Mobility models** are used to simulate and evaluate the performance on mobile wireless systems. The definition of realistic mobility models is one of the most critical and, at the same time, difficult aspects of the simulation of applications and systems designed for mobile environments. NS3 supports mobility models that can be applied to different use-case scenarios. Concerning XANDAR project avionics use-case scenario, we mainly focus on UAVs mobility models. These models are identified by five categories: Pure Randomized, Time - Dependent, Path-Planned, Group and Hybrid.
3) **Energy Models** are critical for analysing energy consumption in wireless devices with constrained resources. NS3 provides two abstraction models for simulating energy consumption at node-level, the Device Energy Model, and the Energy Source. The Device Energy Model is the actual energy consumption model of the network device attached to the network interface. It is designed to be a state-based model, where each state is associated with a specific power consumption value. Whenever the state of the network interface changes, the Device Energy Model notifies the Energy Source, which in turn, drains the corresponding current. Finally, the Energy Source represents the power supply that is attached on each node. A node can have one or more energy sources, and each energy source can be connected to multiple device energy models. Connecting an energy source to a device energy model implies that the corresponding device draws power from the source. Every time the Energy Source receives a notification from the Device Energy Model, it calculates the new total current draw and updates the remaining energy.
4) **Communication channel models** are used to simulate the wireless signal attenuation, allowing network engineers to apply propagation models which simulate different environments (e.g., open/urban areas. A communication channel model is compromised by Propagation Delay Models & Propagation Loss Models. Propagation delay models are simple and a Constant Speed Propagation Delay Model is used, which calculates the delay based-on the distance between transmitter and receiver.

An important part of any wireless network is the appropriate choice of the Propagation Loss Model, which is required to compute the signal strength of a wireless transmission at the receiving stations. Propagation loss depends on the condition of environment (urban, rural, dense urban, suburban, open, forest, sea etc.), operating frequency, atmospheric conditions, indoor/outdoor & the distance between the transmitter and receiver. Propagation Loss Models can be categorized into three main groups (all supported from NS3), Abstract, Deterministic and Stochastics. The main differentiation is on the results that each model produces. Abstract models do not provide realistic results while deterministic models are affected mainly by the distance between the sender and the receiver. Finally, the stochastic models are used in combination with deterministic models to provide non-deterministic results. Finally, an important information required by the propagation loss models is the knowledge of the obstacles between nodes and the presence of Line-Of-Sight (LOS) or not (No-Line-of-Sight (NLOS)). This information is provided by channel condition models that are included in propagation loss models.

E. Framework integration of NS3 building blocks

In this section, the NS3-based simulation framework and integration APIs that allows the network simulation framework to integrate with the rest of the XANDAR toolchain and/or external systems are presented. There are relevant works that integrate the NS3 simulator with other simulators aiming to apply network simulation to their scenarios. Liu et al. [28] integrate the NS3 simulator with Gazebo Robot Simulator, for simulating UAV systems. In the same paper a second integration of NS3 with the SUMO (Simulation of Urban Mobility) framework is presented to illustrate different automotive scenarios. The definition of network simulation analysis stored in a git repository, which in turn is synchronized with the Simulation Orchestrator, structure listed in Table I.

The "src" folder contains the actual network simulation scenario source code. A scenario source is implemented using C++ or Python programming languages. At the next level of the process, network engineers need to define the network configuration parameters (parameters.yaml) that they want to evaluate. NS3 provides APIs for running simulations with dynamic input, and the parameter definitions must be aligned with this API, and the source code. Using the defined parameters, the simulation framework is generating the corresponding

TABLE I
NS3 SCENARIO FILE STRUCTURE

Folder/File	Description
src/	NS3 executables
parameters.yaml	Parameters Pool
requirements.yaml	Application Requirements
contrib/	NS3 Custom Libraries/Modules

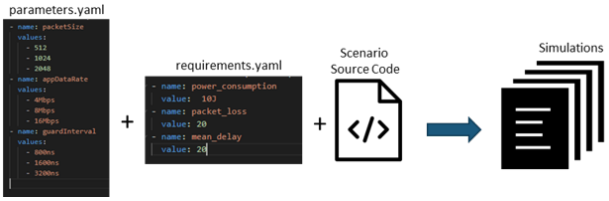

Fig. 4. Simulation generation part

batch of simulations. The necessary files for generating the corresponding simulations are shown in Fig 4. During the execution process the system evaluates the simulation results over the application requirements, while trying to find the optimal network deployment configuration. These application requirements are defined in the requirements.yaml.

The simulation running process is handled by the simulation executor's deployment, where the actual NS3 binaries are running in a containerized environment. Moreover, an application is implemented and executed on top of the NS3 simulator that handles the simulation processes, such as simulation execution and trace processing. Finally, trace processors are deployed to process different trace outputs provided by the NS3 simulator and persist any aggregated results on a database.

V. Conclusion & Future Work

As highlighted, it is fundamental that the verification approach for a given safety-critical system must be deployed in early development cycle to reduce the bug density and to prove the functional correctness of the system. This paper proposes XANDAR's Verification & Validation strategy using Static Analysis, Timing Analysis, Model-in-loop and Network simulation tool that can be deployed for verification of safety critical systems using XANDAR toolchain. The proposed approach guarantees the safety and reliability of the system software by proving the absence of runtime errors. This approach is an additional effort in the V&V life cycle of the system and must be used in conjunction with the traditional V&V methodologies. In future, to assess its effectiveness of the presented V&V approach, it will be integrated into the XANDAR toolchain for the automotive and avionics use cases.

Acknowledgment

This research work was funded by the European Union's Horizon 2020 Research and Innovation Programme under Grant 957210 (XANDAR).

References

[1] F. Siddiqui, R. Khan *et al.*, "Bird's-eye view on the Automotive Cybersecurity Landscape & Challenges in adopting AI/ML," in *Proc. 6th IEEE International Conference on Fog and Mobile Edge Computing (FMEC)*, Gandia, Spain, 2021, pp. 1–6.

[2] I. Van de Poel and Z. Robaey, "Safe-by-design: From safety to responsibility," *Nanoethics*, vol. 11, no. 3, pp. 297–306, 2017.

[3] J. Becker, L. Masing *et al.*, "XANDAR: X-by-Construction Design framework for Engineering Autonomous & Distributed Real-time Embedded Software Systems," in *Proc. IEEE International Conference on Field-Programmable Logic and Applications (FPL)*, Dresden, Germany, 2021, pp. 382–383.

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

[5] P. Johnston and R. Harris, "The Boeing 737 MAX saga: lessons for software organizations," *Software Quality Professional*, vol. 21, no. 3, pp. 4–12, 2019.

[6] P. Koopman, "A case study of Toyota unintended acceleration and software safety," *Presentation. Sept*, 2014.

[7] "Teslas running Autopilot involved in 273 crashes reported since last year," 2023. [Online]. Available: https://www.washingtonpost.com/technology/2022/06/15/tesla-autopilot-crashes/

[8] A. Gannous, A. Andrews *et al.*, "Toward a Systematic and Safety Evidence Productive Verification Approach for Safety-Critical Systems," in *Proc. IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Memphis, USA, 2018, pp. 329–336.

[9] C. W. Johnson, "The increasing risks of risk assessment: On the rise of artificial intelligence and non-determinism in safety-critical systems," in *the 26th Safety-Critical Systems Symposium*. Safety-Critical Systems Club York, UK., 2018, p. 15.

[10] L. Masing, T. Dörr *et al.*, "XANDAR: Exploiting the X-by-Construction Paradigm in Model-based Development of Safety-critical Systems," in *Proc. Design, Automation Test in Europe Conference Exhibition (DATE)*, Antwerp, Belgium, 2022, pp. 1–5.

[11] F. Siddiqui, R. Khan *et al.*, "XANDAR: A holistic Cybersecurity Engineering Process for Safety-critical and Cyber-physical Systems," in *Proc. IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*, Helsinki, Finland, 2022, pp. 1–5.

[12] T. Dörr, F. Schade *et al.*, "Safety by Construction: Pattern-Based Application of Safety Mechanisms in XANDAR," in *Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 369–370.

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

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

[15] "ISO 26262-6:2018 - Road vehicles — Functional safety — Part 6: Product development at the software level," 2023. [Online]. Available: https://www.iso.org/standard/68388.html

[16] "Polyspace - Matlab & Simulink," 2023. [Online]. Available: https://uk.mathworks.com/products/polyspace.html

[17] R. Weber, N. Adler *et al.*, "Towards Automating a Software-Centered Development Process that considers Timing Properties," in *Proc. IEEE 35th International System-on-Chip Conference (SOCC)*, Belfast, UK, 2022, pp. 1–6.

[18] T. Dörr, F. Schade *et al.*, "A Behavior Specification and Simulation Methodology for Embedded Real-Time Software," in *Proc. IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Alès, France, 2022, pp. 151–159.

[19] P. Cousot, R. Cousot *et al.*, "The ASTRÉE analyzer," in *Programming Languages and Systems: 14th European Symposium on Programming, ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005. Proceedings 14*. Springer, 2005, pp. 21–30.

[20] "Coverity Scan - Static Analysis," 2023. [Online]. Available: https://scan.coverity.com/

[21] "Helix QAC for C and C++ — perforce," 2023. [Online]. Available: https://www.perforce.com/products/helix-qac

[22] "LDRA tool suite®-LDRA," 2023, accessed: May 4, 2023. [Online]. Available: https://ldra.com/products/ldra-tool-suite/

[23] "PC-lint Plus — Static Code Analysis for C and C++," 2023. [Online]. Available: https://pclintplus.com/pc-lint-plus/

[24] "Polyspace Bug Finder," 2023. [Online]. Available: https://uk.mathworks.com/products/polyspace-bug-finder.html

[25] C. Ptolemaeus, *System design, modeling, and simulation: using Ptolemy II*. Ptolemy. org Berkeley, 2014, vol. 1.

[26] "PREEvision — The E/E Engineering Solution — Vector," 2023. [Online]. Available: https://www.vector.com/int/en/products/products-a-z/software/preevision/

[27] "TA Tool Suite — Vector," 2023. [Online]. Available: https://www.vector.com/int/en/products/products-a-z/software/ta-tool-suite/

[28] W. Liu, X. Wang *et al.*, "Coordinative simulation with SUMO and NS3 for vehicular ad hoc networks," in *2016 22nd Asia-Pacific Conference on Communications (APCC)*. IEEE, 2016, pp. 337–341.