Elektrotechnik & Informationstechnik (2018) 135/4-5: 322–327. https://doi.org/10.1007/s00502-018-0630-7

Quality assurance methodologies for automated driving

F. Wotawa, B. Peischl, F. Klück, M. Nica

For safety critical systems like cars, trains, or airplanes quality assurance methods and techniques are crucial for preventing situations that may harm people. The case of automated driving represents the next level of safety critical systems where additional challenges arise. This includes the question of how to assure that artificial intelligence and machine learning based systems fulfill safety criticality requirements under all potential conditions and situations that may emerge during operation. In this paper, we first review simulation-based verification and validation methods for such systems and afterwards discuss necessary requirements and future research challenges that have to be solved in order to bring automated driving into practice without compromising safety requirements.

Keywords: functional safety; SOTIF; ADAS; testing; verification

Qualitätssicherungsmaßnahmen für das automatisierte Fahren.

Für sicherheitskritische Systeme wie Autos, Züge oder Flugzeuge sind Methoden und Techniken zur Qualitätssicherung entscheidend, um den Schutz von Leib und Leben zu garantieren. Automatisiertes Fahren stellt eine neue Herausforderung für die Entwicklung und den Test von sicherheitskritischen Systemen dar. Dazu gehört die Frage, wie sichergestellt werden kann, dass Systeme, die auf künstliche Intelligenz und maschinellen Lernen aufbauen, alle sicherheitskritischen Anforderungen, die während des Betriebs auftreten können, erfüllen. In diesem Beitrag stellen die Autoren zunächst simulationsbasierte Verifikations- und Validierungsmethoden für solche Systeme vor und diskutieren anschließend zukünftige Herausforderungen, die gelöst werden müssen, um automatisiertes Fahren in die Praxis umzusetzen, ohne die Sicherheitsanforderungen zu verletzten.

Schlüsselwörter: funktionale Sicherheit; SOTIF; ADAS; Testen; Verifikation

Received March 16, 2018, accepted June 28, 2018, published online July 10, 2018

1. Introduction

Complex electronic-based systems are nowadays realized with a huge amount of software that controls mechanical, electrical or hydraulic components. In such systems software-based subsystems interact with the physical world employing sensors and actuators to fulfill their intended purpose. An example is the anti-lock braking system of a vehicle, which senses the rotation of the wheels and software decides on how to steer the brakes to avoid slipping wheels. To an increasing extent such software-based functional features like Advanced Driver Assistance Systems (ADAS) and automated/autonomous driving capabilities are integrated into modern vehicles [5]. As a consequence, the amount of software in modern vehicles is growing and software subsystems become increasingly interconnected and complex [6]. Such systems exhibit critical properties in a sense that unhandled malfunctioning of parts of the system may result in unacceptable harms to the system itself, to the physical system environment and—most crucial—to the safety of human beings.

1.1 Safety and malfunctioning

Safety-critical systems need to contain mechanisms to detect, localize and repair these malfunctions properly such that no harm can occur. In case that hardware or software elements fail, these elements have to be isolated to prevent the fault propagation to the residual system. There exist several ways to react to a failure at time of operation. In present automotive systems it is often possible to reach a safe state by shutting down parts of the system in case malfunctioning has been detected. For example, failure of a brake-by-wire function could be handled by turning of the units implementing this function without losing control over the driving behavior as there is a mechanical backup for the braking mechanism. However, with the emerging progress towards automated driving functions, critical functionality has to remain operational. Thus, considering a brake-by-wire function having no mechanical backup in place, this function cannot be shut down in case of a failure as this would result in losing the ability to brake. Such systems are referred to as fail-operational and are able to operate with no change in objectives or performance despite of any single failure [15].

In engineering safety-critical system we thus distinguish between safety in use, functional safety (ISO 26262 [1]) and Safety Of The Intended Functionality (SOTIF, ISO PAS 21448 [10]). Safety in use refers to the absence of hazards induced by human error. According to ISO 26262 part 1 [1] functional safety means the absence of unreasonable risk due to hazards caused by malfunctioning behavior. Functional safety is thus primarily dealing with safety in the presence of failures.

Wotawa, Franz, Graz University of Technology, Institute for Software Technology, Christian Doppler Laboratory for Quality Assurance Methodologies for Autonomous Cyber-Physical Systems (QAMCAS), Inffeldgasse 16b/2, 8010 Graz, Austria (E-mail: wotawa@ist.tugraz.at); **Peischl, Bernhard,** Graz University of Technology, Institute for Software Technology, Christian Doppler Laboratory for Quality Assurance Methodologies for Autonomous Cyber-Physical Systems (QAMCAS), Inffeldgasse 16b/2, 8010 Graz, Austria; **Klück, Florian,** Graz University of Technology, Institute for Software Technology, Christian Doppler Laboratory for Quality Assurance Methodologies for Autonomous Cyber-Physical Systems (QAMCAS), Inffeldgasse 16b/2, 8010 Graz, Austria; **Nica, Mihai,** AVL LIST GmbH, Hans-List-Platz 1, 8020 Graz, Austria

According to ISO 26262, part 1 [1] a failure is defined as the termination of the ability of a system element to perform a function as required. In particular, there are two different types of failures: Random failures are failures that can occur unpredictably during the lifetime of a hardware element and in general follow a probability distribution. In contrast, systematic failures are deterministically related to a certain cause. Systematic faults are produced by human error during system development and operation. They can be created in any stage of the system's life including specification, design, manufacture, operation, maintenance and decommissioning. After a systematic fault has been created, the observed failure will always appear, when the circumstances are exactly the same, until the fault is removed. However, it is difficult to predict the occurrence of systematic faults and their effect on the safety of a system. This is because of the difficulty to anticipate when the failure-revealing circumstances will arise [7].

Simple hardware failures are primarily random in nature rather than systematic. While it is possible that hardware can be subject to systematic failures, the level of complexity of hardware means that predominately failures are random in nature. However, this is changing with the growing complexity of processors and the use of Application-Specific Integrated Circuits (ASICS) [7]. Moreover, all software faults are systematic and due to its complexity, the risk of systematic faults is becoming increasingly prevalent. Unlike to random faults, it is not possible to predict the probability of systematic faults. Instead the associated risks are addressed in terms of process-based [8] (e.g., following well-defined development procedures such as recommended by ISO 26262), constructive [9] (e.g., using fault-tolerant system and software architectures) and analytical approaches (e.g., static and dynamic verification) to quality assurance.

1.2 Automated driving and safety of the intended functionality

Safety of intended functionality is the absence of unreasonable hazardous functionality that may occur. In contrast to functional safety, the system does not exhibit malfunctions in its intended function. ISO 26262 does not address the nominal performance of electronic-based systems. Instead SOTIF will cover the safety regarding the nominal behavior. It addresses the fact that for some ADAS applications, a fault-free system can still suffer from safety violations (for example, a false-positive detection of an obstacle by radar). This situation may occur because developing a system that can address every possible scenario is infeasible. For instance, in ADAS deep-learning techniques improve the detection and recognition of objects or humans on the road and furthermore enable the recognition and prediction of actions. The idea of implementing safety-related functions by training neural networks, however, brings empirical design choices and heuristics into the rigorous and rather predictable development cycle as practiced in the automotive industry.

2. Verification, validation and testing for automated driving

To detect the failures as early as possible, one strives to cover the typical as well as corner-case scenarios at the MiL (Model in the Loop), SiL (Software in the Loop) and finally the HiL (Hardware in the Loop) level. Typically, a modern car goes through different test cycles covering the most important scenarios and amounting to about one million miles prior to start of production (SOP). With the advent of autonomous driving, this amount dramatically increases as the number of possible scenarios is exploding and there is need to carry out test cycles that reach (maybe more than) 275 millions of miles [11].

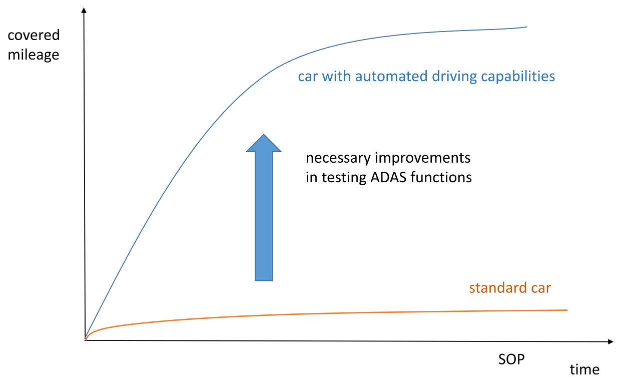


Fig. 1. ADAS increases the mileage to be covered by orders of magnitude

Due to the vast amount of different traffic situations, road conditions, and other influencing parameters, the mileage to be covered dramatically increases as Fig. 1 schematically illustrates.

This increase in the functional range of present-day vehicles rapidly brings new challenges to the automotive industry in terms of development and testing. Following a continuous and traceable process is state of the art and demanded by proven standards like ISO 26262 [1]. Nevertheless, rising system complexity combined with high quality standards and the need for detecting faults as early as possible motivates the industry to employ simulation-based approaches. Particularly when it comes to testing ADAS functions, novel methods are required to assure properties such as safety and adequate fail-operational behavior.

ADAS development starts with a definition of the functional requirements in terms of the desired functions. Hazard and risk analysis are therefore performed to identify the safety requirements. For example, simulation models are nowadays used to support tasks like Fault Tree Analysis (FTA) [12] or Failure Modes and Effects Analysis (FMEA) [12, 13]. This primarily aids in anticipating the potential effects of random and systematic faults and thus in the design and verification of fail-safe or fail-operational systems.

From the functional safety requirements, a system specification is produced to define the precise operation of the system. After implementation of the individual software modules, system integration testing takes place by assembling the complete system from its component modules. In every phase, integration takes place to determine whether the output of a phase meets its specification. This testing demonstrates compliance with the specification, however, potential errors in the specification may result in a faulty ADAS function. It is therefore important to perform validation of the integrated system against its requirements. Usually the development process involves several iterations and the results of verification and validation are used to modify the system specification and design before the next test cycle is conducted. Due to the complexity of ADAS functions, there is the increasing need to obtain reproducible test results as early as possible. Therefore, the various in-the-loop simulation tools are increasingly being used for verification and validation of such functions.

It is state of the art to deploy the software into simulated or real hardware and to provide simulations for the different environments like sensors or actors. In this approach the sensors or actors are replaced with the signals that a sensor or an actor would send respectively receive under given physical conditions. The actions that an actor performs as a response to the received signals are simulated.

The initial design of ADAS functions is supported by MiL simulations, where the function is simulated in closed-loop with models and vehicle dynamics, sensors, actuators and the traffic environment. When MiL simulations have provided sufficient results, software code is compiled from the simulation model of the ADAS function. This code can then be verified with SiL simulation, where hardware components, sensors and vehicle dynamics are simulated. HiL simulation allows to validate real hardware in an early development phase without the need for a prototype vehicle as any missing vehicle components can be simulated. Thus, HiL simulators are much cheaper than test drives and are extensively used for the development of ADAS functions. Because environment sensors should receive a real input, an artificial environment must be created to test an ADAS-equipped vehicle in HiL simulation.

Due to the complexity of the environment and the infinite amount of driving scenarios and parameter combinations that have to be considered, there is urgent need for capable virtual simulation and testing platforms. In the robotics field, Sotiropoulos et al. [14] report on an exploratory study of detecting bugs regarding outdoor robot navigation. The analysis of the triggers and the effects of these bugs shows that most of them could be detected using simulation. Simulation thus can serve both, the testing of the application but also to explore the boundaries of the system.

Also heading into this direction, Mathworks released an application-oriented Automated Driving System Toolbox™ for its product Matlab [2] to design, simulate and test autonomous driving systems. One of the centerpieces of a self-driving car is the vision system, allowing the vehicle to perceive the surrounding environment and trigger actions accordingly. The Automated Driving System Toolbox™ can, for example, be used to develop and verify a perception system for object and lane detection that is capable of tracking and fusing data from multiple sensors connected to the vehicle. Automatically labeled video data can then be used, to test the perception system against various predefined driving scenarios, to visualize the sensor coverage and to check the object classification against ground truth. The toolbox also supports C-Code generation for prototyping and testing of the embedded application.

In the following we list tools that support the procedures described previously: Providing simulation models for failure analysis, the various in-the-loop scenarios for iterative refinement and subsequent integration testing as well as the elicitation of safety and fail-operational requirements and their traceability to test cases. In Table 1, the first column refers to the specific tool, the second column describes its main focus and column three list the modeling language, respectively the types of models being supported.

3. Challenges

Given the situation outlined above, ADAS features and automated/autonomous driving capabilities pose major challenges in the following areas:

Elicitation of safety hazards at the system level: One of the key challenges of automated driving systems is to demonstrate functional safety and SOTIF. In particular, the elicitation of hazards to safety and fail-operational requirements regarding the nominal system behavior is difficult, as such properties need to be analyzed and validated at the system level. For example, due to the intrinsic limitations in sensor performance, wrong perception or erroneous analysis of driving situations is a major hazard. This in turn may result in wrong information provided to the driver, or more severely, the system may trigger a hazardous action such as unintended braking or steering [3]. The elicitation of safety and fail-operational requirements is a foundation for a (requirements-based) test strategy. However, as safety is a system level property, the problem in software-intensive systems is not primarily failure, but the lack of appropriate constraints on software [27].

Generation of test scenarios and test data: A major challenge is to generate meaningful test scenarios that capture the relevant hazardous corner cases for automated driving. NVIDIA's DriveSIM—a virtualized testing environment for self-driving cars—may be a promising tool in this direction. The tool is not only capable of simulating a wide range of traffic scenarios and weather conditions, but other road users are virtualized agents too, which are operating in the same virtual environment. Such testing with virtual environment eases test regressions and shortens test cycles which may be the key to archive sufficient mileage coverage [4].

Providing guarantees for fail-operational behavior: Fully automated driving will require guarantees for fail-operational behavior. To cope with this, the development process needs to incorporate a formal, structural analysis to anticipate the effects of the isolation of single components. This will help to address failures (i.e., violation of safety properties due to random faults, systematic faults, or safety-violations of the fault-free system) in terms of redundant deployments and failover scenarios. Having such failover scenarios in place, however, two major challenges arise. First, the reconfiguration of the remaining computing resources due to the isolation of a specific component poses a challenge. In this respect, work on automated fault localization and program repair appears to be promising [17, 18]. Second, the verification that fail-operational requirements can be met in all considered failover scenarios is an unsolved issue.

Verification and validation of intelligent/adaptive systems: The growth of deep learning in implementing ADAS functions raises important methodological challenges. The rigorous development process followed in the automotive industry clashes with the empirical design choices driven by heuristics when deep-learning is applied within this development lifecycle. Despite neural networks have achieved impressive experimental results, for example in image classification, they can be surprisingly unstable with respect to adversarial perturbations. Minimal changes to their input images may cause the network to misclassify the image [16]. As such misclassifications may result in violation of safety requirements, the verification of safety-relevant functions relying on deep-neural networks is an important challenge that urgently needs to be addressed.

In developing smart and autonomous systems, it is a promising approach to bring verification and testing techniques to the run-time environment. It is an open issue on how to best implement the feedback loop between the development time and runtime. According to [37] an evidence-driven development feedback loop considerably aids in mastering the above-mentioned challenges and makes use of tools such as:

- A continuous integration/delivery/deployment infrastructure that continuously executes a range of test cases on a new version of the intelligent system. This enables the team to obtain continuous feedback regarding critical quality metrics.
- Simulation and simulation-based testing (see Table 1 for the various tools available) allows the development team to better understand the boundaries of experimentation and to detect faults in the early development phases. For this purpose, the simulation needs to be integrated into the continuous integration infrastructure.
- In later stages, dedicated test beds offer a more realistic environment (MiL, HiL, SiL). Some behaviors may risk damage to hardware and mechanics, consequently the boundaries for experimentation tend to be narrower compared to simulation.

Table 1. Overview on tools for virtual development and testing

TOOLS	MAIN FOCUS	MODELS
Dynamic Modelling Laboratory [38]	Description and simulation of physically correct system behavior.	Discrete or continuous models of vehicle sub-systems and components.
Matlab/Simulink [2]	Design and analysis of automotive control system software.	Vehicle control software models.
Unity 3D Game Engine [19]	Describe and model complex environmental conditions for interactive/closed-loop simulation.	Sensor, road and environmental models.
MORSE [20]	Realistic 3D simulation of environments and generic vehicle-like robots for academic usage.	Sensor, actuator and environmental models.
AutoFOCUS 3 [21, 29]	AutoFOCUS 3 is a model-based development tool for distributed, reactive, embedded software systems.	The behavior description is carried out using state automata, source code, or tables.
OSATE [22]	Creation of AADL models for Functional Hazard Assessment, Fault Tree Analysis, Failure Modes and Effects Analysis, and Dependence Diagrams.	Architecture Analysis & Design Language (AADL) models.
ERNEST [23]	Verification and validation of non-functional properties (time) of networked embedded systems with a focus on early design steps.	UML, EAST-ADL (Electronics Architecture and Software Technology—Architecture Description Language) and Artop/AUTOSAR.
OpenCert [24]	Compliance assessment and certification of safety-critical products including the construction of safety cases.	No formal model in place as the focus of OpenCert is on argumentation, evidence and process management.
Eclipse Safety Framework [25]	Analysis tool for FTA analysis.	SysML (System Modeling Language) and MARTE (Modeling and Analysis of Real-time Embedded Systems).
UNISIM-VP [26]	Cross-platform open source simulation environment with a focus on co-design and integration and validation of hardware/software systems. A virtual platform in which—unlike on the real hardware—software can be debugged and tested without affecting either its functional and/or temporal behavior.	Matlab, Simulink, Rational Statemate, Stateflow.
Dynacar RT [28]	Configurable vehicle model running in a real-time system for testing tasks.	Overall vehicle model, third party models e.g. Simulink can be integrated.
NVIDIA DriveSIM [4]	Virtualized testing environment for self-driving vehicles, simulating a wide range of traffic scenarios and environmental conditions.	Road and environmental models as well as other road users being self-driving AI-agents.
Virtual Test Drive (VTD) [30]	VTD provides a complete tool-chain for creation and simulation of virtual worlds. Virtual worlds can be modified to include dynamic content, realistic road properties and any number of externally computed entities.	Virtual worlds can be designed from existing database tiles and 3D model libraries to generate SiL and HiL scenarios.
PreScan [31]	PreScan is a physic-based simulation platform for sensor-driven ADAS development, model-based controller design (MIL) as well as real-time software-in-the-loop (SIL) and hardware-in-the-loop (HIL) testing.	Interface to Matlab/Simulink for designing and testing perception algorithms and sensor models as well as further flexible interfaces to link 3rd party vehicle dynamics models.
CarMaker [32]	CarMaker is an integration and testing platform to realistically model real-world test scenarios, comprising the surrounding environment, driver behavior and traffic situations, in the virtual world.	Complete environment model including an intelligent driver model, a variety of accurate vehicle type models as well as highly adaptable models for roads and traffic.
UnReal Engine [33]	The Unreal Engine provides a sophisticated graphical development environment and a variety of specialized tools. The engine includes modules handling input, output (3D rendering, 2D drawing, sound), networking and physics and dynamics.	The 3-D environment in defined in much the same way as VRML (Virtual Reality Markup language).

Such an evidence-driven development loop allows to refine the various environments towards the real conditions. For example, an ADAS function under test might be carried over from the simulation environment (scenario generation), to the various co-simulation and integration environments (MiL, e.g., early evaluation of boundaries and corner cases), SiL- and HiL testbeds (sensor validation), finally reaching the proving ground (e.g., testing of detailed scenarios and maneuvers) and a public road test field (e.g., testing of regional-specific scenarios). The development of ADAS functions is thus intertwined with systematic experimentation and collection of data to better adjust the models to the real environments.

Online safety analysis: In contrast to basic functionality, ADAS such as parking assistant systems or lane departure warning often provide functionality relevant only in specific driving situations. Hence activating such systems only when needed or expected to be useful promises savings in energy. This is especially relevant to electric vehicles. To realize a system that dynamically might activate and deactivate vehicle functions in order to adapt the system to changing contexts, such as different driving situations [34], self-adaption of ADAS functions must however still meet safety requirements. Therefore, online safety analysis will become a challenge. One attractive idea is to monitor the runtime behavior of such systems against its safety case. As this monitoring mechanism itself is implemented in software, it also becomes the part which inherits the issues with regards to safety assurance [35]. Another view to safety applied to adaptive systems is presented in [36]. There the authors propose Conditional Saftey Certificates (ConSerts). ConSerts [36] are post-certification artefacts (i.e., the certification has been conducted at time of development) equipped with variation points that are bound to formalized external dependencies. These dependencies are resolved at runtime [35, 36].

4. Conclusion

With the advent of ADAS features and automated/autonomous driving capabilities in modern vehicles, software-based functional features become increasingly interconnected and complex. In this article we discuss two different types of failures in the context of functional safety: random failures that occur unpredictably in a system's lifecycle and systematic failures that are produced by human error during system development and operation. Further we argue, that in presence of intelligent systems—such as deep neural networks—even a fault-free system may suffer from safety violation. Thus, the needed increase in the covered mileage due to ADAS features requires considerable improvements regarding simulation-based verification, validation and testing methods. We provide a brief overview of available tools and discuss the methodological challenges in the context of autonomous driving. Among the major challenges are the elicitation of safety hazards at the system level, the generation of (realistic) test scenarios and test data, the provision of guarantees for fail-operational behavior and the safety-verification of intelligent systems employing neural-networks.

Acknowledgements

Open access funding provided by Graz University of Technology. The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

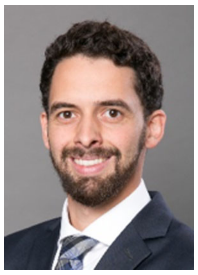
1. ISO (2011): ISO 26262—road vehicles—functional safety. International organization for standardization.
2. De.mathworks.com (n.d.): Automated Driving System Toolbox, [online]. Available at: https://de.mathworks.com/products/automated-driving.html. [Accessed 28 Feb. 2018].
3. Fayolle, G., Raffaëlli, L., Vallée, F., De Souza, P., Rouah, X., Pfeiffer, M., Géronimi, S., Petront, F., Ahiad, S. (2016): Facing ADAS validation complexity with usage-oriented testing. In 8th European congress embedded real time software.
4. Nvidianews.nvidia.com (2018): NVIDIA announces worlds first functionally safe ai self-driving platform, [online]. Available at: https://nvidianews.nvidia.com/news/nvidia-announces-worlds-first-functionally-safe-ai-self-driving-platform.2018. [Accessed, 28 Feb. 2018].
5. Esch, S., Lang, B. (2008): Elektronik- und Vernetzungsarchitektur mit gesteigerter Leistungsfähigkeit. In ATZextra (pp. 194–199).
6. Lammering, D., Balbierer, N., Abdulkhaleq, A. (2016): Automatisiertes Fahren: Keimzelle neuer Architekturkonzepte. In Hanser automotive 11–12/2016, pp. 34–37).
7. Safetyengineering.wordpress.com (2008): Systematic and random failure [online]. Available at: 2018. https://safetyengineering.wordpress.com/2008/04/09/systematic-and-random-failure/. [Accessed 28 Feb. 2018].
8. Kelemen, Z. D., Trienekens, J., Kusters, R. J., Balia, K. (2009): A process based unification of process-oriented software quality approaches. In 4th IEEE international conference global software engineering (pp. 285–288).
9. Wagner, S. (2013): Software product quality control. Berlin: Springer.
10. ISO (2016): ISO/AWI PAS 21448—road vehicles—safety of the intended functionality. International organization for standardization.
11. Nidhi, K., Paddock, S. (2016): Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability? Transp. Res., Part A, Policy Pract., 94, 182–193.
12. Bozzano, M., Villafiorita, A. (2010): Design and safety assessment of critical systems. Florida: CRC Press.
13. Aljazzar, H., Fischer, M., Grunske, L., Kuntz, M., Leitner-Fischer, F., Leue, S. (2009): Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples. In Quantitative evaluation of systems, 6th international conference on quantitative evaluation of systems (pp. 299–308).
14. Sotiropoulos, T., Waeselynck, H., Guiochet, J., Ingrand, F. (2017): Can robot navigation bugs be found in simulation? An exploratory study. In IEEE international conference on software quality, reliability and security (pp. 150–159).
15. Blanke, M., Staroswiecki, M., Wu, E. (2001): Concepts and methods in fault-tolerant control. In Proceedings of the American control conference (Vol. 4, pp. 2606–2620).
16. Huang, X., Kwiatkowska, M., Wang, S., Wu, M. (2017): Safety verification of deep neural networks. In 29th international conference on computer aided verification (pp. 3–29).
17. Jobstmann, B., Griesmayer, A., Bloem, R. (2005): Program repair as a game. In International conference on computer aided verification (pp. 226–238).
18. Brandstotter, M., Hofbaur, M. W., Steinbauer, G., Wotawa, F. (2007): Model-based fault diagnosis and reconfiguration of robot drives. In Proceedings of the IEEE international conference on intelligent robots and systems (pp. 1203–1209).
19. Yamaura, M., Arechiga, N., Shiraishi, S., Eisele, S., Hite, J., Neema, S., Scott, J., Bapty, T. (2016): ADAS virtual prototyping using modelica and unity co-simulation via OpenMETA. In Japanese modelica conference (pp. 43–48).
20. Openrobots.org (n.d.): The MORSE simulator documentation, [online]. Available at: https://www.openrobots.org/morse/doc/stable/morse.html. [Accessed 28 Feb. 2018].
21. AutoFOCUS3 (n.d.): AutoFOCUS 3. Available at: https://af3.fortiss.org/. [Accessed 28. Feb. 2018].
22. Osate.org. (2018): OSATE 2.3.1 documentation, [online]. Available at: 2018. http://osate.org/about-osate.html. [Accessed 28 Feb. 2018].
23. Esk.frauenhofer.de (2015): ERNEST, Early verification and validation of networked embedded systems, [online]. Available at: https://www.esk.fraunhofer.de/content/dam/esk/dokumente/PDB-ERNEST-dt.pdf. [Accessed 28 Feb. 2018].
24. PolarSys.org (n.d.): Open platform for evolutionary certification of safety-critical systems, [online]. Available at: https://www.polarsys.org/proposals/opencert. [Accessed 28 Feb. 2018].
25. PolarSys.org (n.d.): Eclipse safety framework, [online]. Available at: http://www.polarsys.org/esf/. [Accessed 28 Feb. 2018].
26. Unisim-vp.org (n.d.): UNISIM virtual platforms, [online]. Available at: http://unisim-vp.org/site/index.html. [Accessed 28 Feb. 2018].
27. Leveson, N. G. (2004): A systems-theoretic approach to safety in software-intensive systems. IEEE Trans. Dependable Secure Comput., 1(1), 66–86.
28. Dynacar RT (n.d.): Dynacar RT. Available at: http://dynacar.es. [Accessed 28. Feb. 2018].
29. Aravantinos, V., Voss, S., Teufl, S., Hölzl, F., Schätz, B. (2015): AutoFOCUS 3: tooling concepts for seamless, model-based development of embedded systems. In ACES-MB&WUCOR@MoDELS (pp. 19–26).

30. VTD (n.d): Virtual test drive, [online]. Available at: http://www.vires.com/products.html. [Accessed 07. Mar. 2018].
31. TASS International (n.d.): PreScan, [online]. Available at: https://tass.plm.automation.siemens.com/prescan. [Accessed 14. Mar. 2018].
32. IPG Automotive (n.d.): CarMaker, [online]. Available at: https://ipg-automotive.com/products-services/simulation-software/carmaker. [Accessed 14. Mar. 2018].
33. Unreal Engine (n.d.): Unreal engine, [online]. Available at: http://unrealengine.com. [Accessed 14. June 2018].
34. Weiss, G., Grigoleit, F., Struss, P. (2013): Context modeling of dynamic configuration of automotive functions. In 16th international IEEE conference on intelligent transportation systems (pp. 839–844).
35. Ruiz, A., Juez, G., Schleiss, P., Weiss, G. (2015): A safe generic adaption mechanism for smart cars. In 16th IEEE international symposium on software reliability engineering (ISSRE) (pp. 161–171).
36. Schneider, D., Trapp, M. (2013): Conditional safety certification of open adaptive systems. ACM Trans. Auton. Adapt. Syst., 8(2), 8–20.
37. Bosch, J., Holstöm-Olsson, H. (2016): Data driven continuous evolution of smart systems. In 11th IEEE/ACM international symposium on software engineering for adaptive and self-managing systems (pp. 28–34).
38. Dymola (n.d.): Dynamic modeling labaratory, [online]. Available at: https://www.3ds.com/de/produkte-und-services/catia/produkte/dymola/. [Accessed 07. July 2018].

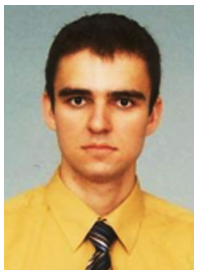
Authors

Franz Wotawa
received a M.Sc. in computer science (1994) and a Ph.D. (1996), both from the Vienna University of Technology, Austria. He is currently professor of software engineering at the Graz University of Technology, Austria. From the founding of the Institute of Software Technology in 2003 to the year 2009, Franz Wotawa was the head of the institute. His research interests include model-based and qualitative reasoning, theorem proving, mobile robots, verification and validation, and software testing and debugging. Beside theoretical foundations he has always been interested in closing the gap between research and practice. For this purpose, he founded Softnet Austria in 2006, which is a nonprofit organization carrying out applied research projects together with companies. Starting from October 2017, Franz Wotawa has been the head of the Christian Doppler Laboratory for Quality Assurance Methodologies for Cyber-Physical Systems. During his career Franz Wotawa has written more than 330 papers for journals, books, conferences, and workshops. He supervised 84 masters' and 34 Ph.D. students. For his work on diagnosis he received the Lifetime Achievement Award of the Intl. Diagnosis Community in 2016. Franz Wotawa has been member of a various number of program committees and organized several workshops and special issues of journals. He is a member of the Academia Europaea, the IEEE Computer Society, ACM, the Austrian Computer Society (OCG), and the Austrian Society for Artificial Intelligence and a Senior Member of the AAAI.

Bernhard Peischl
is a senior scientist at the Institute of Software Technology and the scientific coordinator for the competence network Softnet Austria (managing the innovation programs K-net Softnet Austria and COMET K-Projekt Softnet Austria II). He received a M.Sc. in telecommunications engineering (2001) and a Ph.D. in computer science (2004) from the Graz University of Technology, Austria. He is responsible for managing the network's R&D activities and a number of applied research projects dealing with software test, fault localization and quality assurance. He has co-authored over 85 scientific articles on peer-reviewed workshops, conferences and in scientific journals.



Florian Klück
is a Ph.D. candidate at the Institute of Software Technology at Graz University of Technology, and Reliability Engineer at the department of Reliability Engineering, AVL List GmbH. Currently, he works on his Ph.D. thesis within the doctoral program of technical sciences. His thesis focuses on quality assurance methodologies for autonomous cyber-physical systems. His main research area is centered on verification of highly autonomous vehicles. He completed his Master of Science at the RWTH Aachen University, Germany, with the thesis "Development of a Smart Testing Methodology Focusing on the Service Life of a HV-Battery System", in collaboration with AVL List GmbH.



Mihai Nica
is currently leading the team responsible for the reliability engineering and validation methods at AVL List GmbH, Graz, Austria (headquarters)—powertrain division. He has more than 12 years of work experience in the field of verification and validation, 8 years' experience in the area of R&D project management and more than 6 years' experience in the area reliability engineering and functional safety development and management (ISO 26262). He worked in over 30 customer projects and 10 EU research projects with customers and partners from over 25 countries (China, India, Japan, US, UK, South Korea, Germany, etc.). Mihai Nica received his Ph.D. (Dr. techn.) in the area of software debugging and testing, from Graz University of Technology, Austria.