Editor: **Christof Ebert**
Vector Consulting Services
christof.ebert@vector.com

Validation of Autonomous Systems

Christof Ebert and Michael Weyrich

From the Editor

Autonomous systems are widely used. Yet, for lack of transparency, we are increasingly suspicious of their decision making. Traditional validation, such as functional testing and brute force, won't help, due to complexity and cost. To achieve dependability and trust we need dedicated, intelligent validating techniques that cover, for instance, dynamic changes and learning. Michael Weyrich and I provide industry insights into validating autonomous systems. I look forward to hearing from both readers and prospective authors about this article and the technologies you want to know more about. —*Christof Ebert*

SOCIETY TODAY DEPENDS on autonomous systems, such as intelligent service systems, self-driving trains, and remote surgeries.[1] The ultimate validation of the Turing test is that we often do not recognize autonomous systems. This growing usage poses many challenges, such as how to provide transparency, which rules or learning patterns are applied in a complex situation, and if these rules are the right ones. Validation is the key challenge, of which we will provide an overview in this article.

With machine learning and continuous over-the-air upgrades and updates, a core tenant of any quality strategy is continuous verification and validation. Corrections and changes must be deployed in a fluid and continuous scheme, reliably over the air. We will face future scenarios where software-driven systems, and maybe whole infrastructures, must not be started if they do not include all of the latest software upgrades. Automobiles and manufacturing processes that are safety critical fall into that category. Even more demanding are medical devices, which must provide a hierarchical software assurance because there is no room for failure.

Autonomous systems have multiple complex interactions with the real world. They perceive and act in the environment, based upon the reflections of an intelligent control system, and they have an increasing impact on our lives as they implement and execute high-level tasks without detailed programming or direct human control. Unlike automated systems, which execute a carefully engineered sequence of actions, they are self-governing their course of action to independently achieve their goals.

Figure 1 indicates the five steps from automation to autonomy as we know them from human learning, where we advance from novice to expert. Those steps exemplify the progress of a simple and "assisted

Digital Object Identifier 10.1109/MS.2019.2921037
Date of publication: 20 August 2019

0740-7459/19©2019IEEE

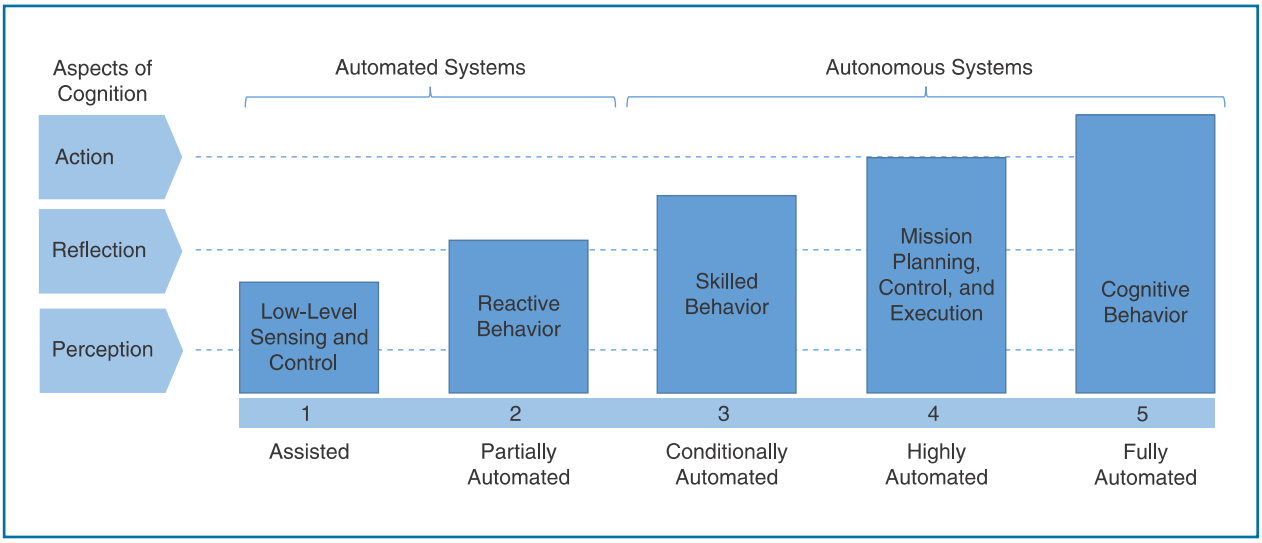

FIGURE 1. The five steps from automation to autonomy.

behavior" from low-level sensing and control toward "full cognitive systems" with a very high degree autonomy. Automated systems are gradually enhanced to develop a skilled behavior along with enhanced mission planning and control and execution capabilities that will eventually lead to the full cognitive actions of an autonomous system. It is expected that an intelligent behavior can be identified by acquiring knowledge and understanding, which entails system functionalities such as perception, reflection, and action in terms of a cognition.

A completely autonomous car on level 5 is supposed to drive with no human intervention, even in dire situations. This implies that the car must have intelligence on par with or better than humans to handle not just regular traffic scenarios but unexpected ones. Although several players, such as Google and Uber, are granted permission to operate their self-driving services, deadly incidents put our faith in these cars to a test.[2] It is quite apparent that existing validation measures aren't enough.[3] We need new test methods that can envision fatal traffic situations that humans haven't encountered yet. In addition, testing cannot simply be isolated to the final development stages. It must be part of every phase in the product lifecycle. A sensible engineering process must be adopted in the development of autonomous cars that lays enough emphasis on testing and validation.

Unlike an automated system, which cannot reflect on the consequences of its actions and cannot change a predefined sequence of activities, an autonomous system is meant to understand and decide how to execute tasks based on its goals, skills, and a learning experience. While contemplating the deficiencies of autonomous systems, we should acknowledge that humans have natural limits, in terms of processing speed, repeatability of tasks, handling complexity, and so forth. In fact, in aerospace, we already trust autonomous flying, and for automotive applications, automation is forecast to reduce deadly accidents by 90%.[4] Autonomous systems can become an aid in the future, in areas such as automated and autonomous driving, flying, and production robotics.

Validation of Autonomous Systems

Autonomous systems provide efficiency and safety as they relieve human operators from tedious manual activities. For instance, the widespread use of self-driving cars could eliminate as much as 50% of a person's daily commuting time.[4] As exciting as this may sound, the question "Can we trust the autonomous systems?" will grow for years to come. Public confidence in autonomous systems depends heavily on algorithmic transparency and continuous validation.

Recently, we have seen several dramatic accidents, such as an automated car misinterpreting a white truck as a white cloud, and another one overlooking pedestrians on a road, thus, killing people. One spectacular accident happened when an automated vehicle continued along while its driver had a heart attack and could not supervise it. Within a few seconds, the automated vehicle

killed a mother and child as it tried to avoid colliding with a tree. Hitting the tree might have killed the driver, but innocent people in the surrounding environment would have been safe.

There are many open questions about the validation of autonomous systems: How do we define reliability? How do we trace back decision making and judge it after the fact? How do we supervise these systems? How do we define liability in the event of failure?

Figure 2 provides an overview of validation technologies for autonomous systems. We distinguish, horizontally, the transparency of the validation. *Black box* means that we have no insight to the method and coverage, while *white box* denotes transparency. The vertical axis classifies the degree to which we can automate validation techniques and, for instance, facilitate regression strategies through software updates and upgrades.

Let us look at traditional testing techniques (see Figures 1 and 2) and evaluate their behaviors. Table 1 provides the complete evaluation of static and dynamic validation technologies for autonomous systems. Negative requirements (such as safety and cybersecurity) are typically implied and not explicitly stated in the system specifications.[5] The following sections explain how these methods are applied to validate autonomous cars.

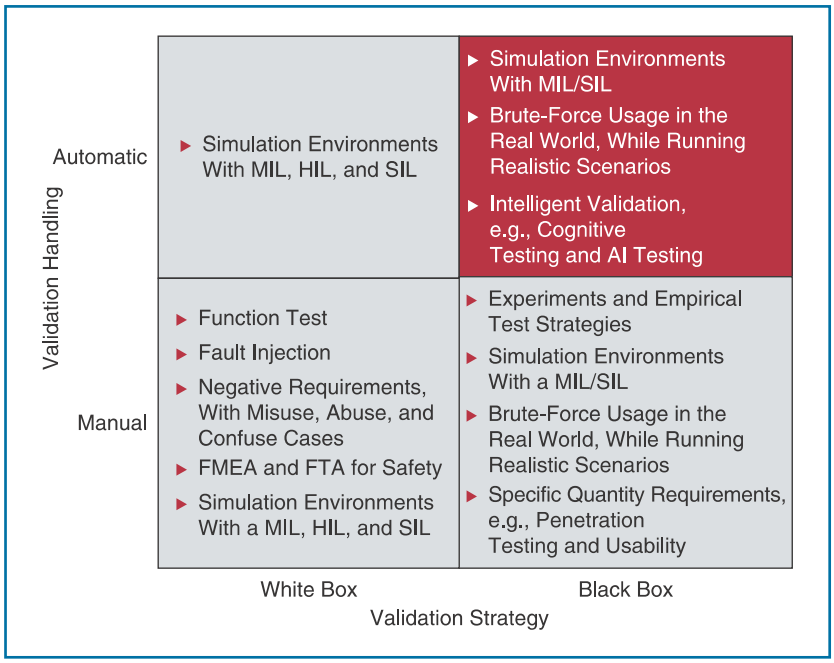

FIGURE 2. The validation technologies for autonomous systems. FMEA: failure mode and effects analysis; FTA: fault tree analysis; AI: artificial intelligence; MIL: model in the loop; HIL: hardware in the loop; SIL: software in the loop.

Fault Injection

Fault injection techniques make use of external equipment to insert faults into a target system's hardware, with or without direct contact. By having direct contact, faults, such as forced current addition, forced voltage variations, and so forth, can be injected to observe the behavior of the system. Faults can be introduced without making physical contact by using methods such as heavy-ion radiation, exposure to electromagnetic fields, and so on. Such fault injections can cause bit flips, hardware failure, and similar events that are not tolerated in safety-critical systems.

Functionality-Based Testing

Functionality-based test methods categorize the intelligence of a system into three classes: 1) sensing, 2) decision, and 3) action functionalities. The idea behind such methods is that an autonomous vehicle should be able to retrieve various functionalities for a given task analogous to human beings. For example, a vehicle should be able to recognize other cars and trucks, pedestrians, and so forth for vision-based functionality. Combinations of these recognized objects can act as inputs to decision functionality, and several decisions can lead to actions. Functionality-based testing breaks down the scenarios into various operational components that can be tested individually.

Hardware in the Loop

Although simulation tries to encapsulate the real world as closely as possible, inherent limitations invariably create a void between the two. Hardware in the loop (HIL) closes this gap a little by using physical components for certain aspects of simulation. For example, a camera model in a simulation technique can be replaced by an actual camera. The input to the camera can be fed by means of a computer screen where videos of various real-time traffic conditions are played to

Table 1. The evaluation of validation technologies for autonomous systems.

Method	Characteristics	Tool support and technologies	Coverage	Regression strategy	Strengths	Weaknesses	Effectiveness	Efficiency
Modeling and simulation environments with SIL, HIL, and MIL	Static and dynamic	Model checker, e.g., MATLAB, dSPACE, Vector VT System, NovaCarts, Vires, PreScan	0	Repeat impacted scenarios (low efficiency)	• Reduces validation costs • Decouples hardware and software development	• Brute force, for high coverage • Requires a large amount of computation power • Tests only for known scenarios • Scenario banks are not comprehensive to validate autonomous systems • Intransparent dependencies	0	0
Function test	Dynamic, all functions	Modeling tool for functional abstraction, with unit test tools (for example, JUnit and PHPUnit) Dedicated test environments for stub generation	0	Repeat functional test cases for impacted operations	• Tests all AI aspects: sensing, decision making, and action taken • Validates all the functional requirements	• Insufficient to validate complete systems	+	+
Integration test	Dynamic	Test suites, test management, combinatorial tools (such as AETG and Citrus and so forth)	0	Regenerate test cases	• Tests component integration	• Large number of interfaces; E to miss some links • Fault localization is difficult	+	+
Fault injection	Static, for residual defect estimation	Test environment and defect modeling, e.g., beSTORM, Security Innovation	–	Introduce few selected defects	• Provides an estimate of residual defects and coverage • Exposes weaknesses, enabling designers to strengthen vulnerabilities	• Requires a concrete understanding of underlying system architecture and behavior	+	–
Negative requirements, with misuse, abuse, and confuse cases	Static, specifically for safety, security, and usability	Directly modeled and traced with requirements tools, e.g., DOORS, Visure, PTC, PREEvision, Enterprise Architect, HP ALM	0	Reuse situational negative cases	• Good for identifying scenarios to be avoided • Formalizes nonfunctional requirements • Strengthens system security	• Difficult to set up systematically • The tests cases do not necessarily cover all possible negative cases	+	+

Continued

Table 1. The evaluation of validation technologies for autonomous systems (cont.).

Method	Characteristics	Tool support and technologies	Coverage	Regression strategy	Strengths	Weaknesses	Effectiveness	Efficiency
FMEA and FTA	Static, specifically for safety-critical systems	FMEA worksheets, component abstractions, and reuse library	0	Retest for the changed components	• Well established for safety and security (attack tree) • Enables designers to foresee system interface failures	• Depends heavily on human knowledge • Labor intensive	+	+
Experiments and empirical test strategies	Empirical test generation for load test, performance, thermal, etc.	Experiment specific test tools, such as Parasoft DTP, EggPlant, Thermal imager, etc.	+	Repeat the test strategies for changed function	• Relatively easier to frame the test cases • Covers a wide range of electrical systems	• Depends heavily on human knowledge • Labor intensive • Very little or no test automation	+	0
Specific quality requirements tests, for instance, penetration testing and fuzzing	Dynamic, specifically for quality requirements	Dedicated test tools, for example, automatic fuzzing extensions, OWASP ZAP, Vega, and so on	–	Retest for impacted components	• Well established for security • Effective in ensuring that the system meets known quality requirements	• Often insufficient to validate complete system security and safety	–	+
Brute-force usage in the real world while running realistic scenarios	Dynamic, for ensuring situational coverage	Recording and replay with actual scenario libraries with data loggers from various sensor systems, e.g., Tecnomatix, CarMaker, EB Assist, CANape	0	Repetition (low efficiency)	• Closest to real world and thus highly effective • Validates all systems at once • Comprehensive view and coverage • Standardizes scenario storage format and tagging	• High effort for coverage • Unclear coverage • Most of the test cases are redundant • Intransparent situational coverage	+	–
Intelligent validation, for instance, cognitive and AI testing	Dynamic test generation and selection depending on situation and environment	Machine-learning frameworks such as Tensorflow, Apache Spark, and so on; open data sets, such as nuScenes	+	Reuse generated test cases from the dependency database	• Improved transparency • Automatically considers dependencies to external environment and internal functions • Automates major part of test procedure • Standardizes scenario storage format and tagging • Sharing test scenarios across V-Model abstraction levels	• High effort to set up AI-based test environment • Needs large computation power • Growing discipline, that is, not many methods and tools available	+	+

validate the behavior of car. A more advanced technique has been proposed for autonomous systems that are tested by robots, for instance, vehicle HIL, where the simulated vehicles in traffic have been replaced by moving robots. This has the advantage that, in addition to the camera, radar and lidar hardware can be tested using HIL.

Vehicle in the Loop

Human interaction can have a drastic influence on the behavior of partially automated cars. The methods specified earlier fail to account for this reality. In vehicle-in-the-loop simulations, real cars are used, though in a safe environment. A driver is shown simulated feeds of the external environment to capture his interaction with the car. The car travels across a ground devoid of obstacles, simulating inertial effects and simultaneously responding to the external feed. The greatest advantage to this method is safety: Since there are no real obstructions involved, no harm will be incurred by the test drivers, even if they encounter dangerous situations.

Simulators

Simulators are closed, indoor cubicles that act as substitutes for physical systems. They can replicate the behavior of any system by using hardware and a software model. The behavior of a driver can be captured by immersing him a replicated external environment. Since simulators employ hydraulic actuators and electric motors, the inertial effects they generate feel nearly the same as the real-life version. They are used for robots in industrial automation, surgery planning in medicine, and railway and automotive applications.

Brute Force

Nothing can come closer to the real world than the real world itself. This is perhaps the final validation phase, where a completely ready system is physically driven onto roads with actual traffic. The sensor data are recorded and logged to capture behavior in critical situations. They are analyzed to accommodate and fine-tune the system according to everyday scenarios. The challenge in this stage, however, lies in the sheer amount of test data that are generated. A stereo video camera, alone, generates 100 GB of data for every kilometer driven. In such situations, big data analysis becomes extremely important.

Intelligent Validation Techniques

Intelligent validation techniques tend to automate the complete testing process or certain aspects of testing. This eliminates the potential errors associated with manual derivations of test cases, since humans may fail to recognize and think about certain scenarios. It also eradicates the enormous amount of time that needs to be invested to obtain the test cases. The "Intelligent Testing" section summarizes some approaches that attempt to derive such validation techniques.

Truly transparent validation methods and processes assume the utmost relevance and will be challenged by the progress of technology through the five steps toward autonomous behavior that are sketched in Figure 1. Although they are still relevant, traditional validation methods aren't enough to completely test the growing complexity of autonomous cars. Machine learning, with situational adaptations and software updates and upgrades, demands novel regression strategies. Figure 2 provides a map of the different testing techniques.

Intelligent Testing

With AI and machine learning, we need to satisfy algorithmic transparency. For instance, what are the rules, in a neural network that is obviously no longer algorithmically tangible, to determine who gets a credit or how an autonomous vehicle might react with several hazards at the same time? Classic traceability and regression testing will certainly not work. Future verification and validation methods and tools will include more intelligence based on big data exploits, business information, and the processes' ability to learn about and improve software quality in a dynamic way.[4]

A key question concerns which way AI can support the process of validation. Obviously, there are many AI approaches, ranging from rule-based systems, fuzzy logic,[6] and Bayesian nets to the multiple neural network approaches to deep learning. However, the process of validating an autonomous system is multilayered and rich in detail. Various levels of validation testing can be distinguished, such as the systems level, the components, and the modules.

The potential for intelligent testing is manifold. On a system level, there are questions about which test cases must be executed and to what extent. This means that intelligent validation is required to help with the selection and even the creation of test cases. A first step in that direction would be an assistance functionality that helped to identify priorities in an existing set of cases. As a result, a validation expert would be able to test faster and with a better coverage of situationally

relevant scenarios. On the level of a component or module,[7] testing it is also required to identify relevant cases. This can range from a simple support mechanism for how to feed a system with adequate inputs and checks on the outputs, to complex algorithms that automatically create test cases based on code or a user interface. Figure 3 provides an overview of intelligent testing as we ramp up for autonomous systems. Unlike brute force, intelligent testing considers the white-box and black-box dependencies and, thus, balances efficiency and effectiveness. See "Cognitive Testing for Autonomous Systems" for a concrete case study.

Perspectives

Verification and validation depend on many factors. Every organization implements its own methodology and

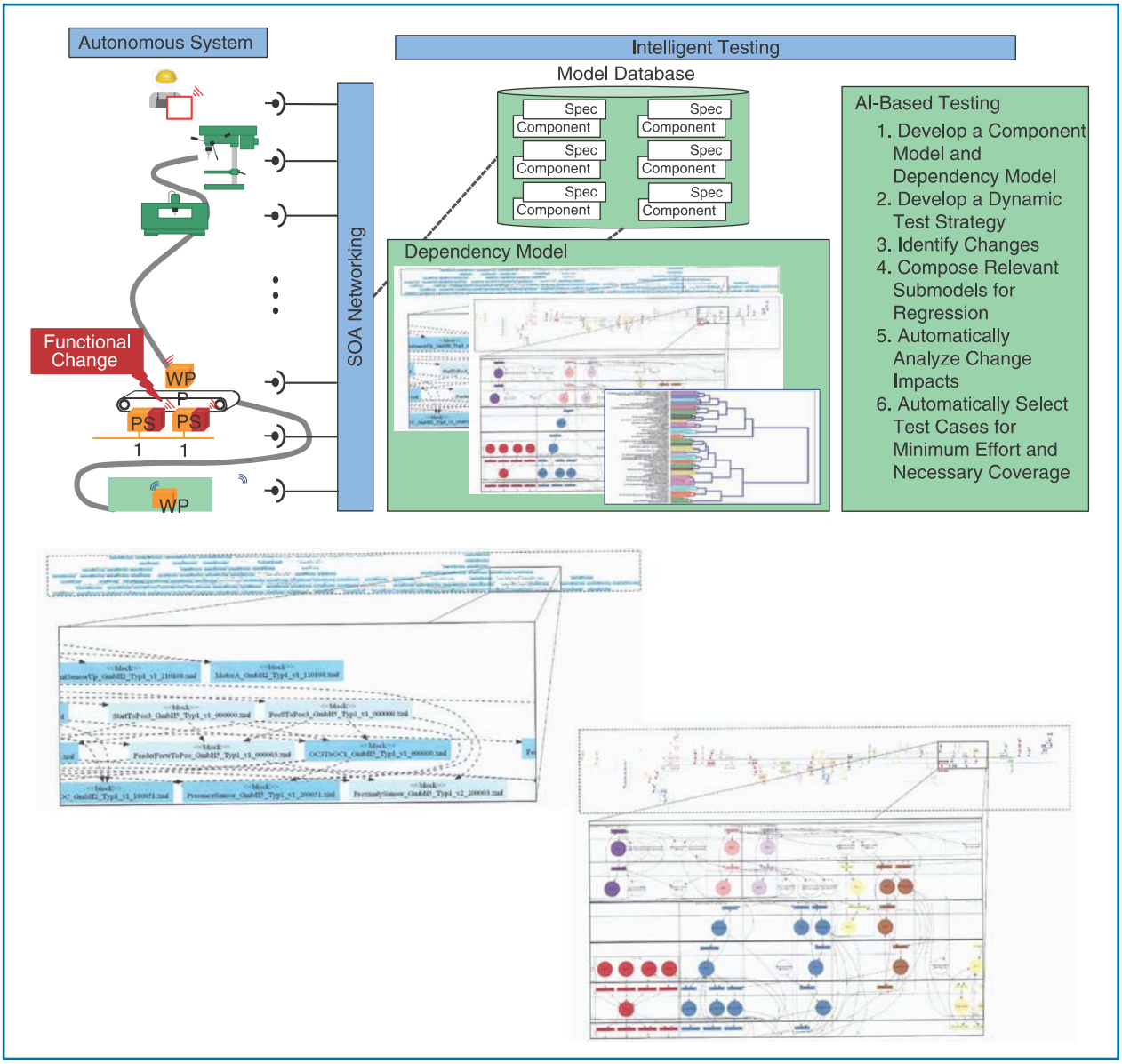

FIGURE 3. Intelligent testing for autonomous systems. SOA: service-oriented architecture; P: process; PS: production sensor; WP: work package.

COGNITIVE TESTING FOR AUTONOMOUS SYSTEMS

In our industrial projects, we often face the challenge of how systems can be validated, and safety assured, when they undergo a change during operation. Updates over the air are commonly used for functional modifications of software-based automated systems. Be they in manufacturing, automotive applications, or intelligent building, automated systems are mostly component based; they consist of multiple control units that are distributed. Each unit is in a certain location and has a specific functionality that it provides to the overall system.

Unwanted behavior and basic functional errors might occur somewhere in a distributed system because of an alteration elsewhere. How can such a system be safeguarded when changes in its components occur during runtime? How can safety and security certifications be maintained after a software modification happens within a single module?

A test certification requires an understanding of the effect of a change that is triggered somewhere in a software module and has impacts elsewhere. How can this interaction be deduced and the consequences for all modules be verified without testing the whole system again from scratch? The method presented here applies an artificial intelligence (AI) that can ascertain the consequences of an individual change in all the control units.

From our industry experience, we recommend a three-step approach to assess the impacts of software updates and upgrades (see Figure 3). First, the alteration in the system needs to be identified in terms of its origin in a module and its localization in the network. Second, a logical model of the overall system is composed to understand the impact on other modules. However, this model is distributed and needs to be automatically processed from the multiple submodules of the components that are available.

Third, a process of functional verification is required to check how the change is propagated and what it means with respect to potential malfunctions in the distributed system. This AI can be used to test and safeguard following a stepwise procedure for testing. It only requires the specification of the control models and their intended interaction with the other modules, upon which the overall functionality can be deduced and test certificates can be obtained on request.

ABOUT THE AUTHORS

CHRISTOF EBERT is the managing director of Vector Consulting Services. He is on the *IEEE Software* editorial board and teaches at the University of Stuttgart, Germany, and the Sorbonne in Paris. Contact him at christof.ebert@vector.com.

MICHAEL WEYRICH is the director of the Institute of Industrial Automation and Software Engineering at the University of Stuttgart, Germany. Contact him at michael.weyrich@ias.uni-stuttgart.de.

development environment, based on a combination of several of the tools presented in this article. It is important not only to deploy tools but to build the necessary verification and validation competences. Too often we see solid tool chains but no tangible test strategies. To mitigate these purely human risks, software must increasingly be capable of detecting its own defects and failure points. Various intelligent methods and tools will evolve that can assist with smart validation of autonomous systems. However, even with the support of the smartest intelligent algorithms, the question remains how to build the public's trust that autonomous systems can be validated while

considering ethical dilemmas, such as the accident when the mother and child were killed.

With the growing concern of users and policy makers about the impact of autonomous systems on our lives and society, software engineers must ensure that autonomy acts better than humans. Clearly, we are not talking about few percentage points. To build trust, we need a level of quality at least one order of magnitude higher than human-operated systems. It is, above all, a question of validation to achieve trust. Alan Turing, who was one of the first to consider AI in real life, remarked wisely, "We can only see a short distance ahead, but we can see plenty there that needs to be done." This remains true for a rather long transition period, and intelligent validation will play a pivotal role.

References

1. M. Weyrich and C. Ebert, "Reference architectures for the Internet of Things," *IEEE Softw.*, vol. 33, no. 1, pp. 112–116, Jan.–Feb. 2016.
2. M. Santori and D. A. Hall. (2016). Tackling the test challenge of next generation ADAS vehicle architecture. National Instruments. Austin, TX. [Online]. Available: http://download.ni.com/evaluation/automotive/Next_Generation_ADAS_Vehicle_Architectures.pdf
3. M. Rodriguez, M. Piattini, and C. Ebert, "Software verification and validation technologies and tools," *IEEE Softw.*, vol. 36, no. 2, pp. 13–24, Mar. 2019.
4. P. Gao, .H.-W. Kaas, D. Mohr, and D. Wee, (2016, Jan.) Automotive revolution: Perspective towards 2030. McKinsey & Co., New York. [Online]. Available: https://www.mckinsey.com/~/media/mckinsey/industries/high%20tech/our%20insights/disruptive%20trends%20that%20will%20transform%20the%20auto%20industry/auto%202030%20report%20jan%202016.ashx
5. *Road vehicles—Safety of the indented functionality*, International Organization for Standardization, 21448, 2019.
6. C. Ebert, "Rule-based fuzzy classification for software quality control," *Fuzzy Sets Syst.*, vol. 63, no. 3, pp. 349–358, May 1994. doi: 10.1016/0165-0114(94)90221-6.
7. A. Zeller and M. Weyrich, "Composition of modular models for verification of distributed automation systems," in *Proc. 28th Int. Conf. Flexible Automation and Intelligent Manufacturing (FAIM2018)*, Columbus, OH, 2018, pp. 870–877.