Validation of Autonomous Vehicles

Automated and connected driving up to autonomous vehicles is increasingly being used. But the distrust in their reliability is growing. The underlying algorithms are difficult to understand and thus intransparent. Traditional validations are complex, expensive and expensive. In addition, no transparent coverage for regression strategies for upgrades and updates is achieved. In this article Vector Consulting and the IAS of the University of Stuttgart show that classical methods have to be supplemented with cognitive test methods.

Society today depends on autonomous systems. Often, we do not even recognize them which is the ultimate proof of the Turing test. The potential of automated and autonomous driving systems is enormous: for example, the use of autonomous vehicles will eliminate up to 90% of accidents and reduce up to 50% of commuting time per user per day [1].

**FIGURE 1** indicates the five steps from automation to autonomy as also known from human learning. Those steps exemplify the way of a simple and “assisted behavior” in terms of low-level sensing and control towards “full cognitive systems” with a very high degree autonomy.

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

Unlike an automated system which cannot reflect the consequences of its actions and cannot change a predefined sequence of activities, an autonomous system is meant to understand and decide about how to execute based on its goals, skills and a learning experience.

This article introduces validation and certification as well as the general approval (homologation) of autonomous vehicles and their components. It pro-
Automated Driving

provides insights into the validation of autonomous systems, such as those used in automation technology and robotics and gives an overview of methods for verification and validation of autonomous vehicles, sketches current tools and shows the evolution towards AI-based techniques for the influence analysis of continuous changes.

VALIDATION OF AUTONOMOUS VEHICLE SYSTEMS

Autonomous vehicle systems have complex interactions with the real world. This raises many questions about the validation of autonomous vehicle systems: “How to trace back decision making and judge afterwards about it?”, “How to supervise?”, or “How to define reliability in the event of failure?”. FIGURE 2 provides an overview on validation technologies for autonomous systems. The transparency of the validation is horizontally distinguished. Black box means that there is no insight to the method and coverage, while white box provides transparency. The vertical axis classifies to the degree we can automate the validation techniques and thus for instance facilitate regression strategies with software updates and upgrades.

TABLE 1 provides a complete evaluation on static and dynamic validation technologies for autonomous systems. It mentions some tools, but they are to be seen as an impulse, rather than a complete list or even a recommendation. Every company today implements its own methodology and development environment. Too often one sees ambitious development teams, complex tool

![Figure 2](image)

![Figure 1](image)
<table>
<thead>
<tr>
<th>Method</th>
<th>Characteristics</th>
<th>Tool Support, Technologies</th>
<th>Coverage</th>
<th>Regression strategy</th>
<th>Strength</th>
<th>Weakness</th>
<th>Effectiveness</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling and simulation environments with SIL, HIL, MIL</td>
<td>Static and dynamic</td>
<td>Model checker, e.g., Matlab, dSPACE, Vector VT System, NovaCarts, Vires, PreScan</td>
<td>0</td>
<td>Repeat impacted scenarios (low efficiency)</td>
<td>&gt; Reduces validation cost. &gt; Decouples hardware and software development.</td>
<td>&gt; Brute Force, for high coverage. &gt; Too much oriented towards components &gt; Tests only for known scenarios &gt; Scenario banks are not comprehensive to validate autonomous systems &gt; Intransparent dependencies</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Function test</td>
<td>Dynamic, all functions</td>
<td>Modeling tool for functional abstraction with unit test tools (Ex. JUnit, PHPUnit), dedicated test environments for stub generation</td>
<td>0</td>
<td>Repeat functional test cases for impacted functions</td>
<td>&gt; Tests all AI aspects: sensing, decision making and action taken. &gt; Validates all the functional requirements.</td>
<td>&gt; Too much oriented towards components. &gt; Insufficient to validate complete systems</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>Integration test</td>
<td>Dynamic</td>
<td>Test suites, test management, Combinatorial tools such as AETG, Citrus etc.</td>
<td>0</td>
<td>Regenerate test cases</td>
<td>&gt; Tests integration of components.</td>
<td>&gt; Large number of interfaces: easy to miss some links. &gt; Fault localization is difficult.</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Fault injection</td>
<td>Static for residual defect estimation</td>
<td>Test environment and defect modeling e.g. beSTORM, Security Innovation</td>
<td>~</td>
<td>Introduce few selected defects</td>
<td>&gt; Provides estimate on residual defects and coverage. &gt; Exposes weak, enabling designers to strengthen them.</td>
<td>&gt; Need concrete understanding of underlying system architecture and behaviour.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Negative requirements with misuse, abuse, confuse cases</td>
<td>Static specifically for Safety, Security, Usability</td>
<td>Directly modeled and traced with requirements tools, e.g. DOORS, Visure, PTC, PREvision, Enterprise Architect, HP ALM</td>
<td>0</td>
<td>Reuse situational negative cases</td>
<td>&gt; Good for scenarios to be avoided. &gt; Formalizes non-functional requirements. &gt; Strengthens system security.</td>
<td>&gt; Difficult to set up systematically. &gt; No coverage schemes. &gt; The test cases do not necessarily cover all possible negative cases.</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>FMEA, FTA</td>
<td>Static, specifically for safety critical systems</td>
<td>FMEA worksheets, component abstraction, reuse library</td>
<td>0</td>
<td>Retest for the changed components</td>
<td>&gt; Well established for safety and security (attack tree). &gt; Enables designers to foresee system interface failures.</td>
<td>&gt; Depends heavily on human knowledge. &gt; Labour intensive.</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Experiments, empirical test strategies</td>
<td>Empirical test generation for load test, performance, thermal, etc.</td>
<td>Experiment specific test tools, such as Parasoft DTP, EggPlant, Thermal imager etc.</td>
<td>+</td>
<td>Repeat the test strategies for changed functions</td>
<td>&gt; Relatively easier to frame the test cases. &gt; Covers wide range of electrical systems.</td>
<td>&gt; Depends heavily on human knowledge. &gt; Labour intensive. &gt; Very little or no test automation.</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>Specific quality requirements test e.g., pen testing, fuzzing</td>
<td>Dynamic, specifically for quality requirements</td>
<td>Dedicated test tools, e.g. automatic fuzzing extensions e.g. CANoe, OWASP ZAP, Vega etc.</td>
<td>~</td>
<td>Retest for impacted components</td>
<td>&gt; Well established for security. &gt; Effective in ensuring that the system meets known quality requirements.</td>
<td>&gt; Often insufficient to validate complete system security and safety.</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>Brute force usage in real-world while running realistic scenarios</td>
<td>Dynamic for ensuring situational coverage</td>
<td>Recording and replay with actual scenario libraries with data loggers from various sensor systems, e.g. Tecnomatix, CarMaker, EB Assist, CANape</td>
<td>0</td>
<td>Repetition (low efficiency)</td>
<td>&gt; Closest to real-world and thus highly effective &gt; Validates all systems at once &gt; Comprehensive view and coverage &gt; Standardizes scenario storage format and tagging</td>
<td>&gt; High effort to capture all relevant scenarios with underlying real-time data analysis &gt; Unclear coverage &gt; Most of the test cases are redundant &gt; Intransparent situational coverage</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Intelligent validation e.g., cognitive testing</td>
<td>Dynamic test generation and selection depending on situation and environment</td>
<td>Machine-learning frameworks, such as TensorFlow, Apache Spark, and so onOpen data sets, such as nubecues</td>
<td>+</td>
<td>Reuse generated test cases from dependency database</td>
<td>&gt; Improved transparency &gt; Automatically considers dependencies to external environment and internal functions &gt; Automates major part of test procedure &gt; Standardizes scenario storage format and tagging &gt; Sharing test scenarios across V-Model abstraction levels</td>
<td>&gt; High effort to set up AI based test environment. &gt; Needs large computation power &gt; Growing discipline, i.e. not much method and tools available</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

**TABLE 1** Evaluation of validation technologies for autonomous systems (© EbertlWeyrich)
Validation technologies for autonomous systems (© Ebert|Weyrich) This eliminates the potential
in safety critical systems. Hardware etc. which are not tolerated
injections can cause bit flips, failure of
emissions etc. Such fault
make physical contact using methods
age variations etc. can be injected to
to physical hardware. Faults
without
Fault injection techniques make use
of external hardware to inject faults
into target system’s hardware. Faults
are injected either with or without
direct contact with physical hardware.
By having direct contacts, faults such
as forced current addition, forced voltage
variations etc. can be injected to
observe the behavior of the system.
Faults can also be injected without
make physical contact using methods
such as heavy-ion radiation, exposure
 electromagnetic fields etc. Such fault
injections can cause bit flips, failure of
hardware etc. which are not tolerated
in safety critical systems.

Functionality based test methods
categorize the intelligence of a system
into three categories: 1. Sensing function-
tivity, 2. Decision functionality and
3. Action functionality. The idea behind
such methods is that the autonomous
vehicle should be able to retrieve various
functionalities for a given task analogous
to human beings. For example, the
vehicle should be able to recognize vehicles,
pedestrians etc. for vision-based func-
tionalities. Combinations of these recog-
nized objects can then act as inputs to
decision functionality and several decisions
can then lead to actions. Functionality-based
 testing therefore breaks down the scenarios into various func-
tional components which can be tested
individually.
Simulators are closed indoor cubicles,
which act as substitute to physical sys-
tems. These simulators can simulate
the behavior of any system either by
using physical hardware or by using
the software model. The behavior of
driver can then be captured by feeding
him simulated external environment.
Since the simulators employ hydraulic
actuators and electric motors, the
inertial effects generated feel nearly the
same as real system. They are used
for robots in industrial automation and
surgery planning in medical, train sys-
tems and automotive.

Nothing can come close to the real
world than the real world itself. This is
perhaps the final validation phase where
completely ready system is driven out
into real roads with real traffic. The sen-
sors data is recorded and logged to cap-
ture the behavior in critical situations.
It is then later analyzed to accommodate
and fine tune the systems according to
real word scenarios. The challenge in
this stage however lies in the sheer
amount of test data that is generated.
A stereo video camera alone is found
to generate 100 GB of data for every
kilometer driven. In such situations,
big data analysis becomes extremely
important. The approval of autonomous
vehicles therefore requires regressive
validation, i.e., a test that, after changing
the control algorithms, performs a new
check and ensures the function. Thus,
safety, reliability and reliability can be
obtained both in development, testing
and in use, even when the system
adapts, i.e. is changed.

While still relevant, traditional vali-
dation methods are not enough to fully
test the growing complexity of autono-
umous cars. Machine learning with situa-
tional adjustments as well as software
updates and upgrades require novel
regression strategies. Intelligent valida-
tion techniques tend to automate com-
plete testing or certain aspects of testing.

Figure 3. This eliminates the potential
errors associated with manual deriva-
tions of test cases since humans may
fail to derive or think about certain sce-
narios. It also eliminates the enormous
amount of time that needs to be invested
to derive the test cases. The following
sub-sections summarize some of the
papers that attempt to derive such vali-
dation techniques.

Truly transparent validation methods
and processes become of an uttermost
relevance and will be challenged by
the progress of technology over the
five sketched steps towards autono-
umous behavior. Although still relevant,
traditional validation methods aren’t
enough to completely test the grow-
ing complexity of autonomous cars.
Machine learning with situational
adaptations and software updates
and upgrades demand novel regres-
sion strategies.
**COGNITIVE TESTING**

With artificial intelligence and machine learning, we need to satisfy algorithmic transparency. For instance, what are the rules in an obviously not anymore algorithmically tangible neural network 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. Rather, future verification and validation methods and tools will include more intelligence based on big data exploits, business intelligence, and their own learning, to learn and improve about software quality in a dynamic way. Cognitive test procedures are based on a database that transparently depicts scenarios and disruptions, so that a target behavior for critical situations, boundary conditions, etc. is defined. In the signal path, signals are generated from the scenarios for the interfaces of the autonomous system or its components. For example, if a child playing suddenly appears in front of the vehicle, the reaction becomes the overall system or the action of its components, e.g. his steering, tested. These signals can be simulations for camera and radar sensors, but also communication signals, such as Car-to-X, residual bus simulation and the display of disturbances.

By parameterization special cases, such as different lighting conditions, can be displayed. From the behavior of the system under test actual rules are extracted, which are compared with the expected target behavior. The automatically extracted actual rules are compared with known and accepted target rules as to how the system under test should behave in the scenario. The target rules are derived from laws, experiences, human expertise, guidelines from ethics committees but also from simulations. They should be transparent and therefore accessible to human testing. Rules are extracted from the behavior of the autonomous system under test in order to make transparent the learned intransparent behavior stored in implicit rules or neuron links. These now transparent but quite fuzzy rules are compared with the set rules in behavior. The validation and certification is based on the control deviations [5,7,8,9].

**FIGURE 3** Cognitive testing for autonomous vehicles (© Ebert/Weyrich)

**AI-based testing**

1. Develop component model and dependency model
2. Develop dynamic test strategy
3. Identify changes
4. Compose relevant sub-models for regression
5. Automatically analyze change impacts
6. Automatically select test cases for minimum effort and necessary coverage

**FIGURE 4** gives an overview of the cognitive testing we are currently using for networked components of autonomous vehicle systems. Unlike Brute Force, the dependencies between the white box and the black box are considered, bringing efficiency and effectiveness into line. Automotive functions consist of the interaction of many components, such as controllers, sensors and actuators, which are distributed in the system. In a distributed overall system, undesirable behavior and basic malfunctions can arise because there has been a software change at one point that breaks through to other components. This raises numerous questions: How can the function of a system be ensured if changes take place in the subcomponents? How can the safety and reliable behavior be guaranteed if software changes are made to individual components during operation?

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

The potential for an intelligent testing is manifold: On a system level there are questions on which test cases must be executed, and to what extent? This means an intelligent validation is requested to help in terms of selection or even creation of test cases for validation. In a first step an assistance functionality which helps to identify priorities in an existing set of cases. As a result, the validation expert can test quicker and with a better coverage of situational relevant scenarios. On the level of a component or module testing it is also required to identify relevant cases. This can range from a simple support on how to feed the system with adequate inputs and check on the outputs to complex algorithms which
automatically create test cases based on the code or user interface.

**PERSPECTIVES**

With the growing importance, and hence the concerns of users and policymakers regarding the impact of autonomous systems on our lives and society, software engineers must ensure that autonomous functions and systems function reasonably well and properly. To build trust, the quality of the technical system is expected to be at least an order of magnitude higher than that of human-powered systems. Building trust is closely linked to issues of validation. However, such validations depend on many factors. Autonomous vehicle systems provide efficiency and safety by relieving the operator of tedious and error-prone manual tasks. The question “Can we trust autonomous vehicles?” Will continue to grow in the coming years. Public trust in autonomous vehicle systems depends heavily on algorithmic transparency and continuous validation.

An accident caused by software errors is discussed more intensively today than the many accidents caused by alcohol. On the other hand, current software errors with deaths in aviation also show a certain “habituation”. The number of passengers does not decrease because of crashes, as everyone knows that the aircraft are altogether safely developed. This learning curve of acceptance can be seen in all autonomous systems, historically for example in smartphones, bots with automatic speech processing and in social networks. An increasingly informed society accepts that while software is never error-free, so there is a residual risk, there are still many advantages over the past.

With a growing concern of users but also policy-makers on the impact of autonomous systems on our lives and society, software engineers must ensure that autonomy acts better than humans. Clearly, we do not talk here about few percentage points. To build trust we rather need at least one order of magnitude better quality compared to 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**


