Toward a Framework for Highly Automated Vehicle Safety Validation

Philip Koopman & Michael Wagner
Carnegie Mellon University; Edge Case Research LLC

Abstract

Validating the safety of Highly Automated Vehicles (HAVs) is a significant autonomy challenge. HAV safety validation strategies based solely on brute force on-road testing campaigns are unlikely to be viable. While simulations and exercising edge case scenarios can help reduce validation cost, those techniques alone are unlikely to provide a sufficient level of assurance for full-scale deployment without adopting a more nuanced view of validation data collection and safety analysis. Validation approaches can be improved by using higher fidelity testing to explicitly validate the assumptions and simplifications of lower fidelity testing rather than just obtaining sampled replication of lower fidelity results. Disentangling multiple testing goals can help by separating validation processes for requirements, environmental model sufficiency, autonomy correctness, autonomy robustness, and test scenario sufficiency. For autonomy approaches with implicit designs and requirements, such as machine learning training data sets, establishing observability points in the architecture can help ensure that vehicles pass the right tests for the right reason. These principles could improve both efficiency and effectiveness for demonstrating HAV safety as part of a phased validation plan that includes both a "driver test" and lifecycle monitoring as well as explicitly managing validation uncertainty.

Introduction

Wide-scale deployment of Highly Automated Vehicles (HAVs) seems imminent despite facing significant interdisciplinary challenges. [1] At this time, there is no generally agreed upon technical strategy for validating the safety of the non-conventional software aspects of these vehicles. Given NHTSA's "non-regulatory approach to automated vehicle technology safety" [2], it seems that many HAVs will be deployed as soon as development teams think their vehicles are ready – and then they will see how things work out on public roads. Even if pilot deployments yield acceptably low mishap rates, there is still the question of whether a limited scale deployment will accurately forecast the safety of much larger scale deployments and accompanying future software updates.

It is common to see statements to the effect that accumulating on-road miles will validate HAV system safety, especially in the context of attempting to characterize progress of development efforts. (E.g., [3], although this does not necessarily represent the actual safety approach of the company discussed.) More comprehensive discussions of the topic still tend to heavily emphasize the role of testing, even if other forms of validation are mentioned. (E.g., [4][5].) However, even with closed courses and high-fidelity simulation, there are limits to the amount of vehicle-level testing that can be done before deployment.

The scope of this paper is validation required beyond ISO 26262 compliance, with an emphasis on SAE Level 4 autonomy. Level 4 HAVs are only required to operate autonomously within a defined Operational Design Domain (ODD), which defines the specific conditions under which the system is intended to function. [2][6]

A safety validation approach for HAV autonomy that goes beyond mileage accumulation is highly desirable. Preferably, it should also be based on a falsification approach that includes concrete, testable safety goals and requirements. [7] This paper proposes a number of ways to improve HAV validation efficiency, increase effectiveness, and lead to a more defensible safety argument. A layered series of validation steps can help support a conclusion that an HAV system is acceptably safe, even in the absence of a completely specified set of traditional functional requirements for autonomy functions.

Approach

We believe that HAV validation efforts can be significantly strengthened by applying the following ideas:

1. Disentangle the disparate goals of testing by separately managing requirements validation and design validation.
2. Use higher-fidelity simulation and tests to reduce residual risks due to assumptions and gaps in lower fidelity simulations and tests.
3. Provide observability in the HAV architecture to ensure that tests are passed for the right reasons.
4. Explicitly manage uncertainty within the safety argument.

Although these ideas are based on existing practices in some domains, the novelty of HAV technology and the pace at which HAVs are being commercialized motivates a clear, unified description of how these ideas can be applied to manage and reduce the risk of aggressive HAV deployment.

Terminology

Our terminology is generally compatible with ISO 26262. [8] The following terms are defined particularly relevant:

Risk: a combined measure of the probability and consequence of a mishap that could result in a loss event.

Safety: absence of unreasonable risk of a mishap resulting in a loss event. Level 4 HAV loss events can include fatalities potentially attributable to HAV design defects or operational faults. For initial HAV deployment, evaluation of what might constitute a "reasonable risk" will be influenced by public policy decisions.

Safety Validation: demonstrating that system-level safety requirements (safety goals) are sufficient to assure an acceptable level of safety and have been achieved.

Safety Argument (Safety Case): a written argument and evidence supporting safety validation.

Machine Learning (ML): an approach using inductive learning for system design, in which a run-time system uses the results of a learning process to perform algorithmic operations (e.g., running a deep convolutional neural network having precomputed weights). This paper assumes weights are fixed before validation. Validating dynamically adaptive ML systems that modify weights or otherwise learn at run-time is beyond the scope of this paper.

The Role of Vehicle Test and Simulation

Before describing proposed validation strategies, it is helpful to review typical uses of testing and simulation in current HAV safety assessment approaches.

Beyond ISO 26262

Dealing with many potential design and implementation defects can, and should, be done via use of an established safety standard such as ISO 26262. [8] For areas in which even a perfectly working system might not provide completely safe functionality, an emerging standard covering Safety of the Intended Functionality (SOTIF) might be used. [9] A SOTIF standard might provide a way to deal with functions with statistically valid functionality, such as radar-based obstacle detection functions. Other issues specific to ML-based systems must also be addressed, as discussed in [10]. Overall, the problem with validating according to a V model as is typical in functional safety approaches is that ML system functionality can be opaque to humans. [11] This makes traceability problematic to the degree that humans performing traceability analysis can't analyze design artifacts. [12]

Rather than attempt a design-to-test traceability approach according to the V model, we instead explore what can be done with a test-centric approach to areas beyond the obvious scope of practical application of ISO 26262 and SOTIF standards which are not designed for ML validation.

System Test/Debug/Patch as a Baseline Strategy

Historically, on-road testing has been emphasized in prototyping autonomous vehicles. (E.g., [13][14][15][16][17].) The field of robotics relies heavily on "real-world" testing in order to gain an understanding of what features robots need. However, as vehicles transition from prototype to production, the approach to validation must become more comprehensive.

Basing an HAV safety argument solely on accumulating road miles is an impractical way to validate safety. Such a brute force approach takes a huge number of miles to make a credible statistical argument. [18] Beyond that, the validity of accumulated road testing evidence is potentially undermined with each software change, whether it be an update to training data, the addition of new behaviors, or just a security patch.

As a practical matter, what happens if, after billions of miles of road testing and simulation, the data shows that an HAV is not living up to its hoped-for safety goal? Will the development team (or should they) do another billion miles of road testing after fixing any observed defects? Or will the team just patch the readily reproducible bugs, test for a few miles, and declare victory, moving on to deployment? And how will the realities of the intense pressure from the race to market influence a team's interpretation of results and approach to validation?

Essentially all other industries base functional safety validation of software-based systems not on trial deployment, but rather both on testing *and* other validation approaches that can be evaluated by an independent assessor. If the HAV industry wishes to follow those precedents, it will need a way to build a methodical, defensible safety argument that can be evaluated by an independent party despite any unique validation challenges.

Limitations of Vehicle-Level Testing and Simulation

As a practical matter, it is impossible to perform enough ordinary system-level testing to assure the safety of a life-critical system. In general, this is because the exposure of an automotive fleet is so high, and life-critical safety requirements are so stringent, that testing cannot accumulate enough exposure hours to statistically prove safety. [19]

For HAVs, one manifestation of the testing infeasibility problem is that unusual situations must be handled safely, but are comparatively rare in normal driving. Road testing is an inefficient way to observe rare events manifesting by chance. Closed-course testing can accelerate exposure to *known* rare events by setting them up as explicitly designed test scenarios. (E.g., [20].) Evaluation might be further accelerated by skewing distributions of test cases toward the more difficult known scenarios. (E.g., [21].) For example, Waymo uses both closed-

course testing and extensive simulation in addition to its on-road test program. [4]

Even covering known scenarios can be challenging due to resource limitations if it exclusively involves the use physical vehicles. Software-based vehicle simulation can scale up coverage of test scenarios via running simulations on many computers in parallel, but inevitably involves a tradeoff of fidelity vs. run-time cost as well as questions about completeness and accuracy of software models. Simulation suffers from the possibility of not simulating unanticipated scenarios (e.g., *unknown* safety-relevant rare events).

"Shadow mode" driving [22] and SAE Level 3 autonomy deployment [6] can increase exposure to real-world driving scenarios by monitoring a deployed fleet in which human drivers are responsible for safety. However, there is controversy as to whether a human driver can effectively supervise safety in Level 3 systems. [23]

Road testing, closed course testing, simulation, and monitoring of human-tended systems all have an important place in demonstrating HAV safety. However, to be both effective and efficient they should be organized in such a way as to work together in a complementary fashion. (We recognize that many HAV developers have sophisticated but proprietary approaches to validation. In this paper we assume a naïve mileage accumulation baseline approach to illustrate the issues.)

Simulation Realism for Its Own Sake Is Inefficient

When asking why on-road testing with a real vehicle is better than simulation, a typical answer is that it is more "realistic." Ultimately testing a real vehicle in the real world is important. But realism for its own sake is an inefficient, and ultimately unaffordable, use of test resources.

The key to simulation validity is having just the right amount of realism (simulation fidelity) to get the job done. It has famously been said that *all* models are wrong, but *some* are useful. [24] Since simulations involve a model of the system, a model of the environment, and a model of system usage, it follows that no simulation is perfect.

The level of fidelity in a simulation is the degree to which it makes simplifications and assumptions about the behavior of the system. Low-fidelity simulations typically execute quickly by using simplified representations of systems (sometimes called reduced-order models), and hence in some sense are "wrong." High-fidelity simulations typically are more complex and are more expensive to execute, but contain with fewer simplifications and assumptions, and are therefore "less wrong." But both types of models can be useful.

The key to improving testing efficiency is realizing that not all realism is actually useful for all tests. As a simple example, modeling the coefficient of road surface friction is generally irrelevant to determining if a computer vision capability can see a child in the road. (The friction coefficient is likely relevant to determining if the vehicle can stop in time, but is not relevant to whether a particular geometric and environmental scenario will result in detecting a child.) This is true whether testing is done in software simulation (via modeling different road surfaces) or with a simulated test track scenario (via sand or ice on tarmac).

The key to effective and efficient simulation is considering not only the system being validated, but also the assumptions made by the various-fidelity models of the system and operational environments. Accordingly, any practical validation effort should be considered as a hierarchical series of models of varying levels of abstraction and fidelity. Viewed this way, closed-course testing is a form of simulation, because even though obstacles and vehicles involved might be real, the scenarios are "simulated." Validating HAV safety will require not only ensuring that the HAV system model is sufficiently accurate, but also validating both the environmental and usage models used to create test plans and testing simulations.

Clarifying the Goals of Testing

A robust safety validation plan must address at least the following types of defects that encompass potential faults in the system, the environment, and system usage:

- **Requirements defects:** the system is required to do the wrong thing (defect), is not required to do the right thing (gap), or has an ODD description gap.
- **Design defects:** the system fails to meet its safety requirements (e.g., due to implementation defects), or fails to respond properly to violations of the defined ODD.
- **Testing plan defects:** the test plan fails to exercise corner cases in requirements or design, or has other gaps.
- **Robustness problems:** invalid inputs or corrupted system state cause unsafe system behavior or failure (e.g., sensor noise, component faults, software defects), or an excursion beyond the ODD due to external forces.

Among the challenges faced by HAV validation are incomplete requirements and implicit representations of both requirements and design. Non-deterministic system behavior further complicates matters. These challenges will of necessity affect the approach to and goals for system testing. [12] (That previous paper concentrates on identifying the challenges in validating autonomy, run-time monitoring approaches, and fail-operational approaches. We build upon that previous work here by discussing the pieces of a validation approach.)

In general, difficulties in applying traditional functional safety approaches to at least some HAV functionality motivates considering the different possible roles of testing in the overall safety validation process, as well as handling the issue of requirements incompleteness.

HAV Requirements Will Be Incomplete

A key challenge for HAV validation is that a complete set of behavioral requirements needs to be developed before behavioral correctness can be measured to provide pass/fail criteria for testing. For example, while efforts are underway to document vehicle behaviors and scenarios (e.g., the Pegasus Project [25]), there is not a complete, public set of machine-interpretable of traffic laws that includes exception handling rules (e.g., when and how exactly can a vehicle cross a center dividing line, if present, to avoid a lane obstruction?). We use the term "requirements" in this paper primarily to refer to system-level behavioral requirements, although the concepts can apply in other ways as well.

Requirements gaps are a primary motivation for on-road vehicle data-gathering operations, which sometimes are loosely referred to as "vehicle testing." The general strategy of inferring system requirements from road test data also affects the completeness of test plans, in that there will be testing gaps corresponding to gaps in system behavioral requirements (e.g., unknown and therefore missing behavioral scenarios).

It is important to note that, strictly speaking, systems that use on-road data as the basis for training machine learning do not ever identify requirements per se. Rather, the training data set is a proxy for something akin to requirements. [12] In other cases, analysis of on-road data might be used to construct some level of explicitly stated requirements. Successfully validating an HAV requires that test plans capture and exercise the required behaviors, even if expressed implicitly. Regardless of the form, these requirements or proxies for requirements are likely to be incomplete for many initial HAVs deployments.

Vehicle Testing for Debugging Can Be Ineffective

A common view of system-level testing is that it is a way to discover software defects ("bugs") and remove them. However, there is a steep diminishing returns problem for vehicle-level testing. Once the easy bugs have been found that involve typical driving scenarios, it can get dramatically more difficult to find additional defects. This is especially true for defects that require very precisely specified initial conditions, involve timing race conditions, or involve recovery from computational run-time faults that are difficult to induce using ordinary vehicle interfaces. This problem is even worse in robotics, in which we have observed that minute variations in lighting and geometry can trigger unreproducible bugs. In general, it can be expected that many such subtle bugs will escape detection and diagnosis during any reasonable amount of vehicle testing, and will be non-reproducible for practical purposes. However, they will surely show up in the field in high exposure applications such as automotive systems.

Beyond an efficiency problem, any project that uses vehicle testing as its primary mechanism of defect removal has a fundamental problem in its safety world-view. Testing can prove the presence of bugs but not their absence. [26] Moreover, when all the bugs found by test have been fixed, the bugs that are left are ones that the testing procedures are not designed to find (the Pesticide Paradox [27]). Thus, even if vehicle-level testing finds no problems at all, that does not mean the vehicle's software is necessarily safe. This line of reasoning is simply another path to concluding that vehicle-level testing alone is an untenable approach to proving system safety.

Vehicle Testing as Requirements Discovery

Some forms of "vehicle testing" are actually aimed at requirements discovery. Examples of areas in which still-maturing HAV development efforts might well have requirements gaps include:

- Detecting and evading novel road hazards
- Handling of exceptional situations that require violating normal traffic rules
- Unusual vehicle configurations, surfaces, and paint jobs
- Misleading but well-formed map data
- Novel road signs and traffic management mechanisms specific to a micro-location or event
- Unusual road markings and vandalism
- Emergent traffic effects due to HAV behaviors
- Malicious vehicle behavior (humans; compromised HAVs)

While HAV designers should design for known requirements, continual novel operational "surprises" are inevitable in the real world for the foreseeable future. A primary rationale for Level 4 automation rather than full Level 5 autonomy is so that the HAV does not have to handle all possible scenarios. Rather, a significant feasibility benefit of Level 4 autonomy is that it is permitted to exhibit a graceful failure when outside its ODD so long as its failure response is safe. Indeed, it would be no surprise if Level 5 autonomy remains an elusive goal over the long term, with Level 4 autonomy asymptotically approaching – but never actually attaining – complete automation in all possible operating conditions and scenarios.

It is important to point out that Level 4 autonomy does not relieve an HAV safety assurance argument from having to deal with all possible scenarios, including ODD violations and novel scenarios. The general concept of an ODD seems to assume that one of the following two situations must be true: (1) there is some external guarantee that the HAV won't encounter a situation it can't handle well due to a highly reliable ODD constraint (e.g., robustly predicting kangaroo road hazard behavior [28] is generally not required on North America public roadways), and/or (2) the HAV will reliably detect that it is in a situation outside its ODD and bring the vehicle to a safe state (e.g., a vehicle not rated for kangaroo road hazards might be geo-fenced out of a wild animal park and the continent of Australia). In reality, it is possible that the ODD will be violated without being detected due to gaps in understanding the full scope of an ODD (e.g., the designers never considered

kangaroos in the first place), or gaps in the validation plan that omit testing relevant ODD constraints.

An appropriate use of on-road operation is finding requirements gaps. Encountering some unexpected scenarios will result in a requirements update, while others result in a modification either of ODD parameters or ODD violation detection requirements. It is important that the HAV be acceptably safe when it first encounters such an ODD "surprise." Accomplishing this is problematic since, by definition, such a scenario is unexpected and therefore not a designed part of any test plan.

Since no validation approach is perfect, it is likely that some design defects will escape and be found via road tests, or even in deployed vehicles. However, this should be a very small fraction of the total number of defects found in the system, and those defects should result in safe behavior even if that behavior does result in a system safety shutdown or other loss of availability. If an excessive fraction of defects escape detection during the development cycle and aren't seen until road testing, that is indicative of a systemic problem with requirements, test plan, or some other element of the validation approach. As with any safety critical design process, defect escapes to production systems should be cause for a significant response to correct any safety process problems that contributed to the situation.

Separating Requirements Discovery and Design Testing

A crucial perspective regarding the role of on-road testing is that accumulating miles in a search for missing requirements isn't really "vehicle testing" in the traditional sense at all. It is a requirements-gathering and validation exercise. On the other hand, whether on-road data or some combination of simulation, synthesized data, and recorded data are the primary means for testing a particular HAV design is more at the discretion of the design team. So long as the design is validated according to an adequately complete set of requirements, on-road testing need not (and in practice should not) be the only testing performed.

Thus, one way to reduce the time and expense of HAV validation is to separate (1) on-road for requirements gathering from (2) design and implementation validation. There is no obvious way around needing billions of miles of on-road experience to seek out rare but dangerous events that need to be mitigated by system safety requirements. But that doesn't mean that design validation needs to re-do those billions of miles for every design change – at least if a more sophisticated approach is taken beyond brute force system-level testing.

Vehicle Testing to Mitigate Residual Risks

We can generalize upon the notion that on-road testing should primarily emphasize requirements validation, while lower level simulation and testing should emphasize the validation of design and implementation. In general, any level of simulation (including "simulated" aspects of vehicle testing) has a particular level of fidelity as previously discussed. That means that it is also "wrong" – just as all models are wrong – in some aspect due to its simplifications and assumptions.

Improving testing efficiency can be accomplished by focusing the test plans for each level of fidelity on checking the assumptions and simplifications of lower-fidelity levels of simulation. At the same time, pushing as much simulation as possible to lowest practical level of fidelity will decrease simulation costs. For example, simple coding defects should be found in subsystem simulation (or even pre-simulation via traditional software unit test and peer reviews). On the other hand, rare event requirements gaps might be best found in on-road testing if they are due to unforeseeable factors. This leads to an approach based on mitigating residual risks for each level of simulation fidelity, as discussed in the following section.

A Layered Residual Risk Approach

Since complete human-interpretable design and requirements information is unlikely to be available for HAVs in the near term, some approach other than, or in addition to, the traditional V model must be used for validation. To do this, we need to start with at least a (possibly incomplete) set of safety requirements. Then, we must find a way to trace some combination of road testing, closed course testing, and simulation results back to those safety requirements.

Validation According to Safety Requirements

At the highest level, we need some type of system requirements to be able to determine whether tests actually pass or fail. If functional requirements are not fully spelled out, then we need something else. The good news is that optimal performance may not be needed to provide safety. Rather, simpler requirements are likely to be sufficient to define safe operation.

For example, we have found that a list of unsafe behaviors that are forbidden based on safety envelopes can be sufficient for some autonomous vehicle behaviors. [29] In that case, testing can be traced to explicitly stated safety requirements even if the functional requirements themselves are opaque or undocumented. One way to specify safety envelopes is using runtime invariants allocated to a distinct safety checker functional block. [30] As a simple example, a safety envelope for lane-keeping could be that the vehicle stays within its lane boundaries plus some safety margin. This is much simpler to specify and use as a test success oracle than checking perfect implementation of a complex algorithm that optimizes the vehicle's lane position according to road geometry and traffic.

While tracing tests to stated safety requirements can be helpful, we have found via experience that too often safety requirements are poorly understood, or not even written down at a useful level of detail. While a vague notion that mishaps should not occur is a starting point, there must also be a concrete and specific way to determine if a test has shown that a system is safe or not. In practice, we have found that a set of partial runtime invariants

that specifies a combination of safe and unsafe system state space envelopes can be evolved over time in a continuous improvement approach in response to the results of testing and simulation. In other words, one way to approach the problem of missing safety requirements is to start with simple set of rules and elaborate them over time in response to tests that violate those simplistic rules. False positive and false negative rule violations can drive refinement of the rule set. Generally, this evolution works best if it starts with an under-approximation of the safe operating envelope (increasing the high false positive rate) and progressively adds additional envelope area (and accompanying test oracle detail) when analysis shows that doing so is a safe way to increase envelope permissiveness.

If an HAV design team attempts to determine safety requirements via machine learning-based approaches, it will be important for them to express the results in a way that is interpretable to human safety argument reviewers. However, it is unclear how that might be done. At this point we recommend using more traditional engineering approaches to defining safety requirements to avoid the same problem of inscrutability that befalls ML-based functionality.

Basing Validation on Residual Risks

While a safety envelope approach can simplify the complexity of creating a model of requirements to use for pass/fail criteria, HAV testing will still need to run a huge number of scenarios to attain reasonable coverage. Ideally as much as possible will be done with comparatively inexpensive, low-fidelity simulations. Then the approach should add fidelity not just for the sake of undifferentiated "realism," but rather for the sake of *reducing the residual risks* due to simplifications made by low fidelity simulations.

Managing Residual Risks

The important relationship between high- and low-fidelity simulation runs should not be one of "sanity checking" or statistical sampling, but rather one of emphasizing validating the correctness of assumptions and simplifications made at lower fidelity levels. In other words, for each aspect in which a particular level of fidelity model is "wrong" in some respect, a higher fidelity simulation (including potentially various types of physical vehicle testing) should assume the burden of mitigating that residual safety validation risk.

This approach is different than the usual notion of model validation in an important way. Higher fidelity levels of simulation are not only used to validate the correctness of lower fidelity models, but must also be explicitly designed to emphasize checks of the assumptions and simplifications that are known to be present as simulations are run. A primary goal of a higher fidelity model should be to mitigate that residual risk by not only checking the accuracy of lower fidelity simulation results, but also by checking whether assumptions made by lower fidelity models are violated when the higher fidelity simulation is performed. As a simple example, if a simplified model assumes 80% of radar pulses detect a target, a higher fidelity model or vehicle test should flag a fault if only 75% of pulses detect a target – *even if* the vehicle happens to perform safely according to the higher fidelity model. The assumption of 80% detection rates is a residual risk of the lower fidelity simulation that makes that assumption. Violating that assumption invalidates the safety argument, even if a particular test scenario happens to get lucky and avoid a mishap.

This approach fundamentally affects the design of a simulation and test campaign. For example, consider a simulation that explores obstacle placements across the field of view. The simulation arranges obstacles in the environment with very precise resolution, but uses only crude stick-figure simulated pedestrian objects in static positions at a fixed orientation. Doing thousands of additional high-fidelity vehicle tests while varying obstacle placement would be expected to yield a low marginal validation benefit over exhaustive simulation results, especially if the simulation exercises the actual geometry processing code that will be deployed in the HAV. That is because in this example obstacle placement relative to the vehicle is not the primary source of residual risk after simulations are completed. The main residual risk revolves around the pedestrians. The low-fidelity simulation assumes stick figure people, thereby omitting consideration of people carrying large objects, people wearing clothing that significantly distorts sensor signals, different rotational positions with regard to vehicle sensors, and so on.

By the same token, any improvement of simulation capability should not merely strive to make the simulation higher fidelity in every possible dimension. For example, modeling road obstacle placement down to the nanometer rather than the millimeter is not likely to be a generally productive use of simulation resources. Rather, simulation fidelity improvements should be made to replace required system level tests with simulations (e.g., adding surface texture capability as well as a wider variety of geometrical shapes and orientations for the previous stick figure example).

This does not mean that simulation model verification and validation (e.g., as described in [31]) should be neglected. Rather, the point is that even a perfectly validated model at a particular level of abstraction leaves residual risk. Part of the risk is because of the possibility of an incomplete testing campaign, which amounts to not fully mitigating risks inherited from lower fidelity simulation or not fully covering the areas assigned to the level of fidelity in question. Another part of the risk is due to safety considerations that have been intentionally excluded at a particular level of abstraction, which corresponds to risks passed up the line to the next higher level of fidelity.

Thus, the time-honored approach of using runs of varied simulation fidelity [32] still makes sense for HAVs. The art is in making sure that simplifications in lower fidelity tests are explicitly managed and mitigated as validation risks.

The approach of accelerated evaluation via biasing tests towards difficult scenarios [21] is complementary to a residual risk approach. Emphasizing difficult scenarios is intended to winnow redundant nominal path tests from the test set while still covering off-nominal behaviors, edge cases, and complex environmental interactions. On the other hand, residual risk mitigation addresses the potential problem of risks due to simplifications and unchecked assumptions made by lower fidelity layers of a simulation and testing plan.

An Example of Residual Risks

Table 1 shows a simplified example of residual risks that should be considered with an HAV testing and simulation plan. The residual risks at the top of the table tend toward requirements gaps (unexpected scenarios and unexpected environmental conditions). In comparison, the other residual risks tend toward a combination of simplifications driven by speed/fidelity simulation tradeoffs (e.g., sensor data quality) at the mid-level, and potential design issues (e.g., subsystem interactions) at the lowest level.

Validation Activity	**Residual Risks (Threats to Validity)**
Pre-deployment road tests	Unexpected scenarios, environment
Closed course testing	*As above, plus:* Unexpected human driver behavior, degraded infrastructure, road hazards
Full vehicle & environment simulation	*As above, plus:* simulation inaccuracies, simulation simplifications (e.g., road friction, sensor noise, actuator noise)
Simplified vehicle & environment simulation	*As above, plus:* inaccurate vehicle dynamics, simplified sensor data quality (texture, reflection, shadows), simplified actuator effects (control loop time constants)
Subsystem simulation	*As above, plus:* subsystem interactions

Table 1. Hypothetical validation activities and threats to validity.

Revisiting the previous obstacle detection example, this means that higher fidelity levels such as physical vehicle testing should not primarily focus on different sizes and placement of obstacles. Rather, they should focus on things such as dirt on objects and sensors, and other aspects that might not be handled by software-only simulation tools. In other words, vehicle testing should mostly concentrate not on reproducing simulation results, but rather on challenging any known weak points of the simulation methodology. Specifics will vary. The point is that all simulation tools have limitations of some sort that require further validation efforts.

For the example shown in Table 1, closed course testing should not focus on unexpected human driver behavior, degraded infrastructure, or road hazards, because mitigating those threats is the primary reason to do pre-deployment road tests. Expected behaviors, road hazards, and so on should be handled with testing and simulation. It is unexpected problems that can't be addressed, because an unexpected problem is by definition not something that can be explicitly included in a test plan.

It is important to avoid burdening higher level system testing with addressing risks that should properly be dealt with at lower levels. Continuing the example, closed-course testing should not be significantly concerned with normal vehicle dynamics, and ordinary issues of sensor data quality and actuator effects, since those can be taken care of with software-based simulation. Vehicle testing should also not be used to brute force test obstacle placement and geometries that can more be dealt with in a more cost-efficient way with simplified vehicle and environment simulation that exercises just the vehicle's obstacle-handling code. Prototyping tests with a real vehicle on a closed course might make sense when validating the simulation capability. But executing the actual vehicle testing campaign should be done at the lowest practical level of simulation fidelity for each aspect of the test plan as much as possible to reduce time and costs.

The overarching idea is that the *primary* emphasis in each level of validation should be on residual risks inherited from the next lower level, especially when re-running existing simulation test suites on a system that has been modified so as to ensure that the system is still safe. Extensive sampling to exhaustively replicate the results of lower fidelity simulation and testing is wasteful at best, and at worst gives a false sense of security if the random sampling does not cover residual risks.

Improving Observability

Given a thorough simulation- and vehicle-based test plan, sufficient controllability and observability must be provided to yield a credible safety validation outcome.

Controllability and Observability

Controllability is the ability of a tester to control the initial state and the workload executed by a system under test. *Observability* is the ability of the tester to observe the state of the system to determine whether a test passed or failed. [33]

Controlling test scenarios to elicit a particular autonomous system behavior is difficult. [12] This is due to a combination of the use of stochastic methods (e.g., randomized path planners), sensitivity to initial conditions (e.g., exactly repeatable sensor alignment within a test environment), variability in actuator outputs (e.g., unexpected variations in environmental interactions with actuators), and computational timing variations.

A useful approach to improving controllability is to use simulation that can avoid physical world randomness and constraints. Beyond that, a system testing interface can be

provided that forces the system into an initial state for testing. For example, a path planner might be tested in a repeatable manner if its internal pseudo-random number generator can be set to a predetermined seed value. As a practical matter, deterministic testing requires that the HAV software be intentionally designed to provide a deterministic testing capability. It can be difficult to mitigate sources of non-determinism in software after it has been constructed.

Observability can be a more difficult problem. For example, in a vehicle-level obstacle test the vehicle either leaves sufficient clearance as it passes an obstacle or it does not. But, even if the system "passes" a test by not colliding, that could simply be due to the system getting lucky in avoiding an obstacle it did not even know was there. The system might hit the obstacle on the next test run – or perhaps hit it 2000 test runs later. This lack of observability is one facet of the robot legibility problem, which recognizes the difficulty of humans understanding the design, operation, and "intent" of a robotic system. [34] (The additional role of legibility in HAV interaction with human drivers is an important one, but beyond the scope of this paper.)

While one can argue that it is unlikely a system will repeatedly pass tests by dumb luck, the sheer number of test parameters involved makes the "repeatedly" part of that argument expensive. And, regardless of how many tests are run, it is difficult to achieve an extreme level of statistical significance via testing for life-critical assurance levels. (Even a 99.99% confidence level for a system avoiding a detected child in a crosswalk seems problematic if it could result in one out of 10,000 children being hit.) Thus, there will always be a residual risk that some combinations of scenario elements pass tests repeatedly due to a lucky streak rather than due to a safe design.

Software Test Points

Rather than relying only upon system-level behavior and brute force repetition to determine if a test passes, a more efficient testing approach can be to insert software test points into the system to improve observability. For example, if sensor fusion dependability is a residual risk due to simulation limitations, a relevant test point for closed course vehicle simulation would be monitoring the computed certainty level of a sensor fusion results. That would provide information about whether a test obstacle is being avoided with the intended margin of error rather than by luck. (The issue of software test points potentially disturbing the system under test can be resolved by architecting test points in as a permanent part of the system. This will in turn facilitate data collection in the deployed system.)

Software test points also facilitate monitoring for safety argument assumption violations during fleet deployment. The previously discussed 80% detection rate assumption example can be monitored not only during testing, but also during full scale vehicle deployment to detect assumption violation escapes into fielded systems.

Passing Tests for the Right Reason

When a human takes a driver test, the test examiner has a fairly accurate (or at least useful) mental model of the driver behind the wheel. If the driver changes lanes without making eye contact with a rear-view mirror or otherwise checking for vehicles in the destination lane, the examiner knows that the driver got lucky in executing a collision-free lane change instead of behaving properly. With an HAV, this type of assessment is more difficult, because it is unclear what the "tells" are for a machine exhibiting safe behavior vs. getting lucky with unsafe behavior. That is especially true if requirements and design are not traceable via a V-based safety process.

If HAV safety is to be based in part on a driving-test type event, then the examiner must know that the HAV not only behaves the right way, but also behaves the right way for the right reason. Even without a formal driver test, being able to reasonably infer causality of actions from explicit system information can reduce testing costs compared to a brute force statistical approach. Having an HAV self-report regions of saliency [35], bounding boxes on objects, and so on is not a new idea. However, explicitly including such capabilities in a safety argument can reduce testing cost if exploited in the right way. This may motivate further work to verify that self-reporting and explainability mechanisms work reliably.

One way to couple scenarios with behaviors is to have the HAV self-report the scenario it thinks it is in, or the various scenario elements that it thinks are in play. As an example, rather than just performing a vehicle lane change when it can, the vehicle might report: "I want to change lanes … I am checking the next lane and there is a car there but it is sufficiently far behind me that I am clear … I am starting to change lanes … I am continuing to monitor that the lane is still clear ... the car behind me is speeding up to close the gap …" and so on. Some HAV architectures might provide this level of observability already. The question is how formally such information is used by the validation strategy. Moreover, many popular approaches (e.g., end-to-end deep learning) explicitly eschew architectural modularity, which tends to degrade observability. They do so with the goal of achieving higher performance, a tighter implementation, and less development effort. [36] Lack of observability has the potential to exact a high price in terms of validation effort or deployment risk for such systems.

An effective driving test should require not only correct behavior, but also a correct introspective narrative of why the HAV is acting the way it is. That is a good start, but we must then must question the integrity of a machine's explanation for its actions. However, we argue that deciding whether to trust an explicit explanation is an easier to solve problem than having to infer (and then trust) an opaque implicit explanation via behavioral observation. Either way, a decision must be made about whether the vehicle will do the right thing in future circumstances that are not exact matches to training and test data sets. The advantage of an explicit explanation is that the

validity of that mechanism can be made falsifiable if it is required to match the test plan narrative. In designing safety-critical systems, we prefer explicit, verifiable, simple patterns that might be less performant over those that are highly-optimized but opaque. We have reason to believe this trend will hold for HAVs when considering the consequences of attempting to deploy difficult-to-validate systems.

Architecting such a system will require introducing or identifying observability for the purpose of validation. This might be accomplished by having a tool that converts existing data to human-interpretable form, adding a test point to the system architecture, or re-architecting the system to intentionally create new forms of human-interpretable data. (Figure 1)

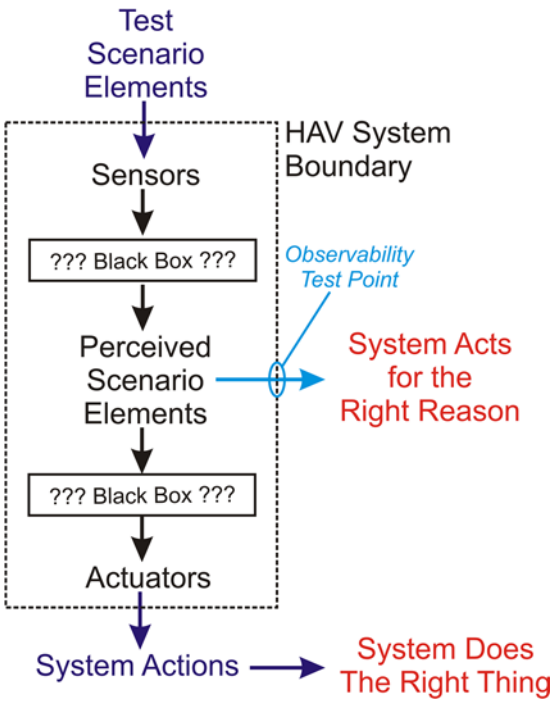

Figure 1. System validation should determine that the system does the right thing for the right reason.

For machine learning systems, this approach suggests a somewhat unusual design strategy. Rather than having an ML system learn its own feature set for achieving an outcome, it must meet two concurrent goals: (1) display the right behavior, and (2) display a set of narrative descriptions or other explanation that matches its behavior. One way to accomplish this is to use models of environments and usage scenarios to define the set of ML outputs that must be learned. While this might be seen as additional design burden and overhead, such might be the price for being able to know whether a vehicle is actually safe enough to deploy.

To avoid a mismatch between behavior and narrative, one possibility is to arrange the ML system so that it operates in two disjoint phases: first creating the narrative, and then using the narrative as inputs for its behavior, as shown in Figure 1. The first phase might build on existing work on creating descriptions of scenarios and hierarchical classification. (E.g., [37][38].) The system actuation should be responsive to the narrative by having the second stage be fully dependent upon the outputs of the first stage. This dependency mitigates the risk of a parallel narrative construction being generated that does not actually match the system's actuation strategy.

Coping with Uncertainty

Knowns and Unknowns

Even with a validated and apparently defect-free system, there is still residual risk from problems due to incomplete understanding of the system and its requirements. These include at least the following potential types of issues:

- Emergent system properties and interactions that are not accounted for at the appropriate validation phase
- Unexpected correlated faults in areas for which safety depends upon implicit independence assumptions
- Scenario and environment exceptions that happen too infrequently to be diagnosed by pre-deployment road tests
- Uncertainty as to the arrival rates of unmitigated hazards that were assumed to be extremely infrequent
- In-range system inputs that activate unexpected defects in ML-based components

There are doubtless other types of defects that are not listed above and are not included in at least some HAV validation plans. Those are the famous "unknown unknowns" [39] that can compromise safety and cause other system failures.

Dealing with Unknown Defects

While approaches such as safety envelopes can help, in the end, there is no way to completely mitigate residual risks from unknown types of defects. However, the arrival of unexpected faults can be monitored to increase confidence over time that the residual risk is sufficiently low. It is essential to recognize unknown problems as a residual risk that must be monitored and mitigated as necessary throughout the life of the fleet. A confidence assessment framework [40] that has been extended to include unknown unknowns is one approach that could provide a way to manage residual risks.

Each time a surprise causes a safety problem, additional steps should be taken to address underlying system and safety argument assumptions that are invalidated by the newly discovered issue (this is in accordance with existing safety practices, e.g., [8]). It is important to do a root-cause analysis of unexpected faults to at least determine if a problem is a known unknown (in which case now you know more about it), or an unknown unknown (in which case you need to add a category of defect type to your validation plan and safety argument to address this new unexpected source of problems).

HAV Maturity

There is substantial intuitive appeal to having a "driving test" as part of HAV validation. However, the analogy of taking an HAV out for a road test similar to a human driving test falls short because there are actually two key elements to a human driving test. The first element is the obvious, overt requirement that the driver must show basic driving knowledge and proficiency, including a driving skills test.

The second and more subtle part of passing a driving exam is that the driver must be approximately 16 years old, depending upon locale. That age requirement serves as a proxy for having reasonably mature judgment that can handle exceptional situations and generally behave in a reasonable manner when encountering a novel unstructured situation. In the real world, correct vehicle operation depends in part upon traffic regulations. However, it also depends upon whether a police officer expertly, though subjectively, thinks the driver behaved in a reasonable and responsible manner for a given situation. ("Plays well with others" is an important HAV characteristic, especially in mixed human/HAV traffic.)

While it is possible (some say certain) that HAV behavior *can* be safer than a person given human frailties, how to measure HAV "maturity" to ensure that this desirable outcome is fully achieved remains an open question.

One way to measure HAV maturity is to deploy vehicles and see how they do. That is one of the arguments for deploying SAE Level 3 automation, which in effect uses a human in the role of an adult supervisor who monitors the junior driver during learner's permit operation. However, there are legitimate concerns that driver supervision will be ineffective over long periods of exposure due to driver dropout, especially when automation fails infrequently. [41]

We propose two different approaches for evaluating HAV maturity beyond developer adherence traditional safety critical software engineering principles. The first way is ensuring that the HAV passes a detailed technical driving skill test *for the right reasons*, and the second way is monitoring whether the HAV *validation assumptions and residual risk monitoring* hold up when it is deployed in the real world. In other words, the system design might be considered to be mature if the vehicle can explain its behavior in a way that makes sense to a human and its safety case assumptions hold true in operation.

HAV Probation: Monitoring Assumptions

Any responsible decision to deploy an HAV must be more sophisticated than simply saying "we fixed all the bugs we found so we must be perfect," because that is never a reflection of reality. There is always one more bug. [42] Rather, a safety argument based on phased validation should at least be made based on measuring rates of defect escapes from each phase of validation. This argues that observability test points should be retained and monitored all the way through to fleet deployment. Doing so permits monitoring system design maturity by ensuring that there are no vehicle operational situations that invalidate assumptions. If a high rate of assumption violations is detected by runtime monitoring, that can provide valuable feedback to the design team of an impaired safety margin. In this manner, issues with the safety argument can be identified even if no actual mishaps have occurred.

As another example beyond the previously discussed assumption violation example, consider the somewhat controversial topic of disengagement reports for HAV road testing. [23] Clearly, not all disengagements are created equal, especially given that various teams are likely to have different false positive rates for triggering disengagements.

Using an approach such as Orthogonal Defect Classification (ODC) [43] might reveal, for example, that some disengagements are due to problems that should have been caught in subsystem simulation, while other disengagements are due to the discovery of a requirement or scenario gap at the highest level. While one expects that HAV development teams do some sort of analysis on disengagements, a methodical analysis that maps defects back to residual risks identified in a validation plan has significant potential benefits, such as providing a health indication for the safety argument and the HAV's overall maturity level.

This approach can support an external assessment of autonomy validation by presenting a well-reasoned set of risk mitigation goals for each phase of validation. Those can be paired with data on defect escapes as measured by relevant observability points during simulation, vehicle testing, and deployment. All this implies that the "driver test" is not actually a one-time event, but rather involves a continual "license" renewal process based on collecting and analyzing field data on defect escapes over the life of the system.

Deploying with Residual Risks

It is important to acknowledge that this discussion has contemplated fielding HAVs that have residual risks, and in particular, potential gaps in requirements and design verification. This is inherent to the domain and the technology being deployed. It will be some time before statistically defensible amounts of data are accumulated to argue that the residual risks fall below the usual safety critical system safety thresholds (for example, below one catastrophic vehicle mishap per 10^9 or 10^{10} operational hours). Given the current HAV market and regulatory climate, it seems likely that public deployment will scale up before such data is collected.

Regardless of the appeal of fielding HAVs, is essential that the deployment be done in a responsible manner. In particular, residual risks should not be accepted blindly. Rather, residual validation risks at all levels should be explicitly understood as well as monitored during deployment. As an example, credible arguments that a particular category of residual risk is likely to

result in low consequence, highly survivable, or extremely infrequent mishaps might be a legitimate motivation to determine they are "acceptable" even if the full extent of the risk is unclear. However, any such argument should be supported by monitoring field feedback data to determine if the assumptions that support the acceptance of such risks are actually true, preferably without waiting for an accumulation of serious loss events.

Ultimately ethical issues arise, such as whether it is better to deploy imperfect technology if there is an expected net savings of life. [44] Safety professionals in particular face a pragmatic choice as to whether they participate in a release of a safety-critical system with unknown (and unknowable, in the short term) but safety risks, or they miss an opportunity to improve the relative safety of HAVs that are bound to be deployed with or without their help. A goal of this paper is to provide a framework for validating such systems before they are deployed that will improve the developers' ability to identify and manage accepted risks.

Conclusions

Summarizing, we describe an approach to HAV validation that includes the following elements:

- A phased simulation and testing approach that emphasizes testing to mitigate residual validation risks from the previous phase while exploiting the speed vs. fidelity scalability properties inherent in testing and simulation.
- Observability points to produce human-interpretable data that both detect defect escapes from lower fidelity simulation phases and demonstrate the system is doing the right thing for the right reason.
- Explicit differentiation of the various roles of testing from checking for requirements gaps to checking for design faults, and matching each type of testing with a relevant portion of a phased validation approach.
- A run-time monitoring approach to managing identified risks, catching assumption violations and unknown unknowns as they arise in fielded systems.

This approach can be expected improve validation effectiveness compared to a brute-force testing campaign because it explicitly links testing and simulation activities to the risks being mitigated. This in turn permits concentrating effort on the sweet spot of defect detection for each particular level of simulation and test fidelity. The approach can also be expected to improve testing efficiency by concentrating each phase of testing on mitigating risks inherited from the preceding phase, without wasting resources revisiting low-risk conclusions or attempting to address out-of-scope risks that belong to other testing phases. (Other forms of validation beyond testing are also important, such as employing ISO 26262 approaches to appropriate portions of system functionality.)

We recognize that, due to the challenges of conclusively establishing the safety of machine-learning functions, the approach presented here will yield an ongoing process of iterative improvement rather than air-tight proofs of safety. However, the approach will serve to underscore where assumptions are being made, and where safety case evidence is missing. One way of validating the approach as well as the system is to create a Goal Structuring Notation-organized safety case (e.g., starting with [45]) and including explicitly stated assumptions to complete the argument. Each assumption identifies the residual risks for a testing or simulation technique. Assumptions that are checked by other validation approaches form part of the safety argument chain. Assumptions that can't be validated at design time are residual risks that are especially important candidates for run-time monitoring in deployed systems.

At some point, designers will have to decide on a responsible deployment plan that might involve taking risks that are judged to be acceptable according some defensible set of technical and social criteria. To minimize unmitigated residual risks, we suggest avoiding architectures in which autonomy that can't be validated using traditional safety approaches is the sole means of ensuring operational safety. One alternative is using a safety checker that can be rated appropriately according to ISO26262, such as a safety envelope monitor. [46]

While it is always better to ensure that all residual risks are known and mitigated to an acceptable level, it is clear that HAVs are going to be deployed even if there are places in which the safety argument contains risks that are not completely understood. The approach discussed in this paper provides a framework for establishing an initial safety argument based on multiple levels of simulation and testing fidelity. It also provides hooks for continuous improvement based on monitoring assumption violations and other residual validation risks during the course of testing and deployment.

Our next steps are refining techniques for establishing traceability from safety requirements to test and simulation plans, and applying this approach to at-scale validation activities.

References

[1] Koopman, P. & Wagner, M., "Autonomous Vehicle Safety: An Interdisciplinary Challenge," IEEE Intelligent Transportation Systems Magazine, Vol. 9 #1, Spring 2017, pp. 90-96.

[2] NHTSA, Automated Driving Systems: a vision for safety, US Dept. of Transportation, DOT HS 812 442, Sept. 2017.

[3] Carson, B., "Uber's self-driving cars hit 2 million miles as program regains momentum." Forbes, Dec. 22, 2017.

[4] Waymo, On the Road to Fully Self-Driving: Waymo safety report, 2017. https://goo.gl/7HUiew

[5] General Motors, 2018 Self-Driving Safety Report, 2018. https://goo.gl/ruLJvV

[6] SAE, Automated Driving (from SAE J3016), http://www.sae.org/misc/pdfs/automated_driving.pdf accessed 10/13/2017.

[7] Wagner & Koopman, "A Philosophy for Developing Trust in Self-Driving Cars," In: G. Meyer & S. Beiker (eds.) Road Vehicle Automation 2, Lecture Notes in Mobility, Springer, 2015, pp. 163-170.

[8] Road vehicles -- Functional Safety -- Management of functional safety, ISO 26262, 2011.

[9] Road vehicles – Safety of the Intended Functionality, ISO/WD PAS 21448. Under development.

[10] Salay, R., Queioz, R., & Czarnecki, K., "An analysis of ISO 26262: Using Machine Learning Safely in Automotive Software," https://arxiv.org/pdf/1709.02435.pdf

[11] Dosovitskiy, A., & T. Brox, "Inverting convolutional networks with convolutional networks," CoRR, vol. abs/1506.02753, 2015.

[12] Koopman, P. and Wagner, M., "Challenges in Autonomous Vehicle Testing and Validation," SAE Int. J. Trans. Safety 4(1):2016.

[13] Urmson, C. et al., "Autonomous driving in urban environments: Boss and the Urban Challenge," Journal of Field Robotics, 2008, pp. 425-466, DOI 10.1002/rob.

[14] Levinson et al., "Towards fully autonomous driving: systems and algorithms," IEEE Intelligent Vehicles Symp., June 5-9, 2011, pp. 163-168.

[15] Broggi et al., "Extensive tests of autonomous driving technologies," IEEE Trans. Intelligent Transportation Systems, 14(3), Sept. 2013, pp. 1403-1415.

[16] Ziegler, J., et al., "Making Bertha drive – an autonomous journey on a historic route," IEEE Intelligent Transportation Systems Magazine, Summer 2014, pp. 8-20.

[17] Aeberhard, M., et al., "Experience, results and lessons learned from automated driving on Germany's highways," IEEE Intelligent Transportation Systems Magazine, Spring 2015, pp. 42-57.

[18] Kalra, N. & Paddock, S., Driving to Safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability? Rand Corporation, RR-1479-RC, 2016.

[19] Butler & Finelli, "The infeasibility of experimental quantification of life-critical software reliability," IEEE Trans. SW Engr. 19(1):3-12, Jan 1993.

[20] Madrigal, A., Inside Waymo's secret world for Training self-driving cars, The Atlantic, Aug. 23, 2017.

[21] Ding, Z., "Accelerated evaluation of automated vehicles," http://www-personal.umich.edu/~zhaoding/accelerated-evaluation.html on 10/15/2017.

[22] Golson, J., Tesla's new autopilot will run in "shadow mode" to prove that it's safer than human driving, The Verge, Oct, 19, 2016.

[23] Davies, A., The very human problem blocking the path to self-driving cars, Wired, Jan 1, 2017.

[24] Box, G., "Robustness in the strategy of scientific model building", MRC Technical Summary Report #1954, University of Wisconsin-Madison, 1979.

[25] Putz, A., Zlocki, A., Bock, J. & Eckstein, L., "System validation of highly automated vehicles with a database of relevant traffic scenarios," 12th ITS European Congress, Strasbourg, June 19-22, 2017.

[26] Bustcon, J., & Randell, B., (Eds.) Software Engineering Techniques: report on a conference sponsored by the NATO Science Committee, April 1970.

[27] Beizer, B., Black-Box Testing: Techniques for functional testing of software and systems, Wiley, 1995.

[28] Zhou, N., "Volvo admits its self-driving cars are confused by kangaroos," The Guardian, June 30, 2017. https://goo.gl/jgA7Ck

[29] Koopman, P., "Challenges in Autonomous Vehicle Validation," SCAV 17, April 2017.

[30] Kane, Chowdhury, Datta & Koopman, "A Case Study on Runtime Monitoring of an Autonomous Research Vehicle (ARV) System," RV 2015.

[31] Sargent, R., "Verifying and Validating Simulation Models," 2014 Winter Simulation Conference, pp. 118-131.

[32] Law, A. & Kelton, W.D., Simulation Modeling and Analysis, 3rd ed., McGraw Hill, 2000.

[33] Freedman, R., Testability of software components, IEEE Trans. Software Engineering, June 1991, pp. 553-564.

[34] Dragan, A., Lee, K. & Srinivasa, S., "Legibility and predictability of robot motion," Human-Robot Interaction (HRI) 2013, pp. 301-308.

[35] M. Bojarski et al., "VisualBackProp: efficient visualization of CNNs", arXiv:1611.05418v3

[36] M. Bojarski et al., "End to End Learning for Self-Driving Cars", arXiv:1604.07316v1

[37] Wang, Y., Lin, Z., Shen, X., Cohen, S. & Cottrell, G., "Skeleton Key: image captioning by skeleton-attribute decomposition," arXiv preprint arXiv:1704.06972

[38] Redmon, J. & Farhadi, A., "YOLO9000: Better, Faster, Stronger," https://arxiv.org/pdf/1612.08242.pdf

[39] Morris, E., "The Certainty of Donald Rumsfeld (Part 2)," NY Times, 26 March 2014, https://goo.gl/Pv7SB7.

[40] Wang, R., Guiochet, J. & Motet, G., "Confidence assessment framework for safety arguments," SAFECOMP 2017, pp. 55-68.

[41] Casner, S., Hutchins, E., & Norman, D., The Challenges of Partially Automated Driving, Comm. ACM, May 2016, pp. 70-77.

[42] Leveson, An investigation of the Therac-25 Accidents, IEEE Computer, July 1993, pp. 18-41.

[43] M. Sullivan, R. Chillarege, "Software Defects and their Impact on System Availability A Study of Field Failures in Operating Systems," FTCS-21, 1991.

[44] Kalra, N. & Groves, D., The Enemy of Good: estimating the cost of waiting for nearly perfect automated vehicles, Rand Corporation, RR-2150-RC, 2017

[45] Burton, S., "Making the case for safety of machine learning in highly automated driving," SAFECOMP, Sept. 2017, pp. 5-16.

[46] Kane, Fuhrman, Koopman, "Monitor Based Oracles for Cyber-Physical System Testing," DSN 2014.

Contact Information

Dr. Philip Koopman is an Associate Professor of Electrical and Computer Engineering at Carnegie Mellon University, where he specializes in software safety and dependable system design. He also has affiliations with the Carnegie Mellon University Robotics Institute, National Robotics Engineering Center (NREC) and the Institute for Software Research. He is CTO and co-founder of Edge Case Research, LLC. E-mail: koopman@cmu.edu

Michael Wagner is CEO and co-founder of Edge Case Research, LLC, which specializes in software robustness testing and high-quality software for autonomous vehicles, robots, and embedded systems. He is also affiliated with the National Robotics Engineering Center. E-mail: mwagner@edge-case-research.com

Definitions/Abbreviations

HAV	Highly Automated Vehicle
ODD	Operational Design Domain
NHTSA	National Highway Traffic Safety Administration
SOTIF	Safety Of The Intended Functionality
V model	A software development model that includes requirements and design on the left side of a "V" with verification and validation on the right side of the "V"