2016-01-0128
Published 04/05/2016

doi:10.4271/2016-01-0128
saetransaf.saejournals.org

Challenges in Autonomous Vehicle Testing and Validation

Philip Koopman
Carnegie Mellon University

Michael Wagner
Edge Case Research LLC

ABSTRACT

Software testing is all too often simply a bug hunt rather than a well-considered exercise in ensuring quality. A more methodical approach than a simple cycle of system-level test-fail-patch-test will be required to deploy safe autonomous vehicles at scale. The ISO 26262 development V process sets up a framework that ties each type of testing to a corresponding design or requirement document, but presents challenges when adapted to deal with the sorts of novel testing problems that face autonomous vehicles. This paper identifies five major challenge areas in testing according to the V model for autonomous vehicles: driver out of the loop, complex requirements, non-deterministic algorithms, inductive learning algorithms, and fail-operational systems. General solution approaches that seem promising across these different challenge areas include: phased deployment using successively relaxed operational scenarios, use of a monitor/actuator pair architecture to separate the most complex autonomy functions from simpler safety functions, and fault injection as a way to perform more efficient edge case testing. While significant challenges remain in safety-certifying the type of algorithms that provide high-level autonomy themselves, it seems within reach to instead architect the system and its accompanying design process to be able to employ existing software safety approaches.

CITATION: Koopman, P. and Wagner, M., "Challenges in Autonomous Vehicle Testing and Validation," *SAE Int. J. Trans. Safety* 4(1):2016, doi:10.4271/2016-01-0128.

INTRODUCTION

While self-driving cars have recently become a hot topic, the technology behind them has been evolving for decades, tracing back to the Automated Highway System project [1], and before. Since those early demonstrations, the technology has matured to the point that Advanced Driver Assistance Systems (ADAS) such as automatic lane keeping and smart cruise control are standard on a number of vehicles. Beyond that, there are numerous different fully autonomous vehicle projects in various stages of development, including extended on-road testing of multi-vehicle fleets.

If one believes pundits, full-scale fleets of autonomous vehicles (often called "self-driving cars") are just around the corner. However, as the traditional automotive industry knows well, there is a huge difference between building a few vehicles to run in reasonably benign conditions with professional safety drivers, and building a fleet of millions of vehicles that have to run in an unconstrained world. Some say that successful demonstrations and a few thousand km (or even a few hundred thousand km) of driving experience means that autonomous vehicle technology is essentially ready to be deployed at full scale. But, it is difficult to see how such testing alone would be enough to ensure adequate safety. Indeed, at least some developers seem to be doing more, but the question is how much more might be required, and how we can know that the resultant vehicles are sufficiently safe to deploy.

In this paper we explore some of the challenges that await developers who are attempting to qualify fully autonomous, NHTSA Level 4 [2] vehicles for large-scale deployment. Thus, we skip past potential semi-automated approaches to address systems in which the driver is not responsible at all for safe vehicle operation. We further limit scope to consider how such vehicles might be designed and validated within the ISO 26262 V framework. The reason for this constraint is that this is an acceptable practice for ensuring safety. It is a well-established safety principle that computer-based systems should be considered unsafe unless convincingly argued otherwise (i.e., safety must be shown, not assumed). Therefore, autonomous vehicles cannot be considered safe unless and until they are shown to conform or map to ISO 26262 or some other suitable, widely accepted software safety standard.

Infeasibility of Complete Testing

Vehicle-level testing won't be enough to ensure safety. It has long been known that it is infeasible to test systems thoroughly enough to ensure ultra-dependable system operation.

For example, consider a hypothetical fleet of one million vehicles operated one hour per day (i.e., 10^6 operational hours per day). If the safety target is to have about one catastrophic computing failure in this fleet every 1,000 days, then the safety goal is a mean time between catastrophic failures of 10^9 hours, which is comparable to

aircraft permissible failure rates. [3] Note that this admits to the likelihood that several such catastrophic failures due to computer defects or malfunctions will happen during the life of the fleet of cars. However, such a goal might be justifiable if accompanied by a much larger reduction in catastrophic mishaps due to driver error compared to manually driven vehicles. (This is just an example failure rate. Arguments might be made for this rate to be higher or lower, but it has been selected as a defensible rate that illustrates some of the difficulties in achieving safety.)

In order to validate that the catastrophic failure rate of a vehicle fleet is in fact one per 10^9 hours, one must conduct at least 10^9 vehicle operational hours of testing (a billion hours) [4], and in fact must test several times longer, potentially repeating such tests multiple times to achieve statistical significance. Even this assumes that the testing environment is highly representative of real-world deployment, and that circumstances causing mishaps arrive in a random, independent manner. Building a fleet of physical vehicles big enough to run billions of hours in representative test environments without endangering the public seems impractical. Thus, alternate methods of validation are required, potentially including approaches such as simulation, formal proofs, fault injection, bootstrapping based on a steadily increasing fleet size, gaining field experience with component technology in non-critical roles, and human reviews. (Component level testing also plays a role, but it is still impractical to accumulate 10^9 hours of pre-deployment testing for a physical hardware device.) Things get even worse when one considers that testing is even more difficult for autonomy systems than for everyday software systems, as will be discussed below.

That having been said, for relatively non-critical computing systems it may be possible to use testing as a primary basis for validating an appropriate level of safety. This is because failures involving low severity and low exposure may be permissible at a higher occurrence rate than catastrophic failures. For example, if a failure of a particular type once every 1,000 hours is acceptable (because such failures result in a minimal-cost incident or slight disruption), then validation of that failure rate could be credibly achievable by testing for several thousand hours. This is not to say that all software quality process can be abandoned for such systems, but rather that a suitable testing and failure-monitoring strategy might make it possible to validate that a component with suitable quality has actually attained an acceptably low failure rate if the mean-time-between-failure requirement is relatively lenient.

The V Model as a Starting Point

Because system-level testing can't do the job, more is required. And that is precisely the point of having a more robust development framework for creating safety critical software.

The "V" software development model has been applicable to vehicles for a long time. It was one of the development reference models incorporated into the MISRA Guidelines more than 20 years ago [5, 6]. More recently, it has been promoted to be the reference model that forms the basis of ISO 26262 [7].

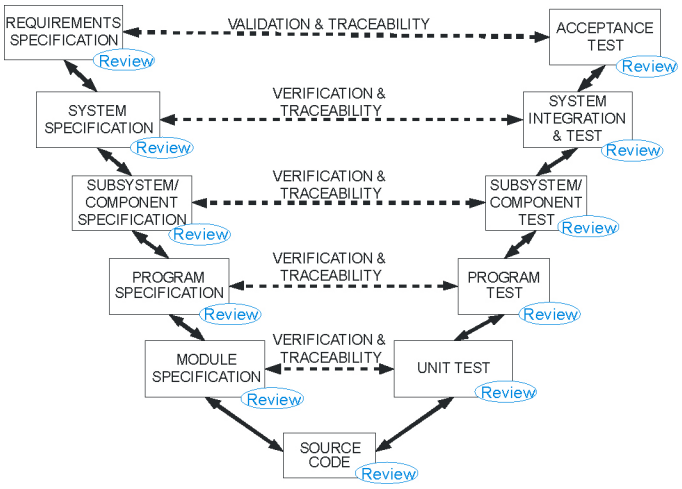

Figure 1. A generic V model.

In general, the V model (Figure 1) represents a methodical process of creation followed by verification and validation. The left side of the V works its way from requirements through design to implementation. At each step it is typical for the system to be broken into subsystems that are treated in parallel (e.g., there is one set of system requirements, but separate designs for each subsystem). The right side of the V iteratively verifies and validates larger and larger chunks of the system as it climbs back up from small components to a system-level assessment. While ISO 26262 has a detailed elaboration of this model, and much more, we keep things generic so as to discuss the high level ideas.

Although ISO 26262 and its V framework generally reflect accepted practices for ensuring automotive safety, fully autonomous vehicles present unique challenges in mapping the technical aspects of the vehicle to the V approach.

DRIVER OUT OF THE LOOP

Perhaps the most obvious challenge in a fully autonomous vehicle is that the whole point is for the driver to no longer be actually driving the vehicle. That means that, by definition, the driver can no longer be counted on to provide control inputs to the vehicle during operation. [2]

Controllability Challenges

Typical automotive safety arguments for low-integrity devices can hinge upon the ability of a human driver to exert control. For example, with an Advanced Driver Assistance System (ADAS), if a software fault causes a potentially dangerous situation, the driver might be expected to over-ride that software function and recover to a safe state. Drivers are also expected to recover from significant vehicle mechanical failures such as tire blow-outs. In other words, in human-driven vehicles the driver is responsible for taking the right corrective action. Situations in which the driver does not have an ability to take corrective action are said to lack controllability, and thus must be designed to a higher Automotive Safety Integrity Level, or ASIL. [8]

With a fully autonomous vehicle, the driver can't be counted on to handle exceptional situations. Rather, the computer system must assume that role as the primary exception handler for faults, malfunctions, and beyond-specified operating conditions. Putting the computer in charge of exception handling seems likely to dramatically increase automation complexity compared to ADAS systems. Combinations of ADAS systems such as lane-keeping and smart cruise control seem tantalizingly close to fully autonomous operation. However, a fully autonomous vehicle must have significant additional complexity to deal with all the ways things might go wrong because there is no driver to grab the wheel and hit the brakes when something goes awry.

Autonomy Architecture Approaches

In the context of ISO 26262, putting the computer in charge suggests one of two strategies for assessing risk. One strategy is that the controllability portion of risk evaluation [8] should be set to "C3 Difficult to control or uncontrollable." This might be a viable option if the severity and exposure are very low, and thus a low ASIL can be assigned. However, in cases that have moderate or high severity and exposure, the system must be designed to a high Automotive Safety Integrity Level (ASIL). (Some might argue that there should be an even higher controllability classification C4 because of the potential of an automation system to take proactively dangerous positive actions rather than simply failing to deliver a safety function. But we assume here that the existing C3 suffices.)

Another way to handle a potentially high-ASIL autonomy function is to use ASIL decomposition [9] via a combination of a monitor/actuator architecture and redundancy. A monitor/actuator architecture is one in which the primary functions are performed by one module (the actuator), and a paired module (the monitor) performs an acceptance test [5, 10] or other behavioral validation. If the actuator misbehaves, the monitor shuts the entire function down (both modules), resulting in a fail-silent system (i.e., any failure results in a silent component, sometimes also known as fail-stop, or fail-safe).

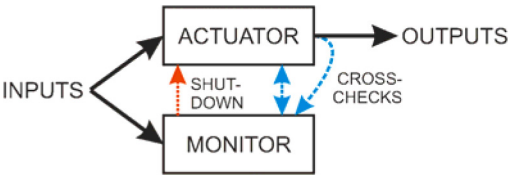

Figure 2. Monitor/actuator pair conceptual diagram.

If the monitor/actuator pair (Figure 2) is designed properly, the actuator can be designed to a low ASIL so long as the monitor has a sufficiently high ASIL and detects all possible faults in the monitor. (There is also a requirement to detect latent faults in the monitor to avoid a broken monitor failing to detect an actuator fault.) This architectural pattern can be especially advantageous if the monitor can be made substantially simpler than the actuator, reducing the size of the high-ASIL monitor, and permitting the majority of the functional complexity to be placed into a lower-ASIL actuator.

Both the strength and weakness of a monitor/actuator pair is that it creates a fail-silent building block (i.e., one that shuts down if there is a fault). The use of heterogeneous redundancy (two modules: the monitor and the actuator) is intended to prevent a malfunctioning actuator from issuing dangerous commands. However, it also causes loss of the actuator function if something goes wrong, which is a problem for a function that must fail operational, such as steering in a moving vehicle.

At the very least, providing fail operational behavior requires even more redundancy (more than one monitor/actuator pair), and very likely design diversity so that common-mode software design failures do not cause a systemic failure. This is important to avoid situations such as the loss of Arianne 5 Flight 501, which was caused by both a primary and a backup system that failed the same way due to experiencing the same un-handled exceptional (unanticipated by the component design) operating condition. [11]

It should be noted that achieving diversity is not necessarily simple, due to issues such as vulnerability to defects in the same set of high-level requirements used to implement the diverse components (e.g., [12]). However this is a situation that is also true for non-autonomous software. It should also be noted that a monitor/actuator pair's fail-silent requirement is based on an assumption of failure independence, but again this is also true of non-autonomous systems.

A key high level point is that regardless of the approach, it seems likely that there will need to be a way to detect when autonomy functions are not working properly (whether due to hardware faults, software faults, or requirements defects), and to somehow bring the system to a safe state when such faults are detected via a fail-operational degraded mode autonomy capability.

COMPLEX REQUIREMENTS

An essential characteristic of the V model of development is that the right side of the V provides a traceable way to check how the left side turned out (verification and validation). However, this notion of checking is predicated on an assumption that the requirements are actually known, are correct, complete, and unambiguously specified. That assumption presents challenges for autonomous vehicles.

Requirements Challenges

As mentioned earlier, removing the driver from the control system means that software has to handle exceptions, including weather, environmental hazards, and equipment failures. There are likely to be very many different types of these, from bad weather (flooding, fog, snow, smoke, tornados), to traffic rule violations (wrong-direction cars on a divided highway, other drivers running red lights, stolen traffic signs), to local driving conventions (parking chairs, the "Pittsburgh Left" [13]), to animal hazards (deer, armadillos, and the occasional plague of locusts).

Anyone who has driven for a long time is likely to have stories to tell of freak events they've seen on the road. A large fleet of vehicles will, in aggregate, be likely to experience all such types of events, and perhaps more. Worse still is that combinations of adverse events and

driving conditions can occur that are simply too numerous to enumerate in a classical written requirements specification. Perhaps not all such extremely rare combinations need to be covered if results are likely to be innocuous, but the requirements should be clear about what is within the scope of system design, as well as what is not. Thus, it seems unlikely that a classical V process that starts with a document that enumerates all system requirements will be scalable to autonomous vehicle exception handling in a rigorous way, at least in the immediate future.

Operational Concept Approaches

One way to manage the complexity of requirements is to constrain operational concepts and engage in a phased expansion of requirements. This is already being done by developers who might concentrate on-road testing in particular geographic regions (for example only performing daytime driving on divided highways in Silicon Valley, which has limited precipitation and little freezing weather). However, the idea of employing an operational concept can be scaled in many directions.

Examples of axes that can be exploited for limiting operational concepts include:

- Road access: limited access highways, HOV lanes, rural roads, suburbs, closed campuses, urban streets, etc.
- Visibility: day, night, fog, haze, smoke, rain, snow, etc.
- Vehicular environment: self-parking in a closed garage with no other cars moving, autonomous-only lanes, marker transponders on non-autonomous vehicles, etc.
- External environment: infrastructure support, pre-mapped roads, convoying with human-driven cars
- Speed: lower speeds potentially lead to lower consequences of a failure and larger recovery margins

While there are still a great many combinations of the above degrees of freedom (and more that can no doubt be imagined), the purpose of selecting from possible operational concepts is not to increase complexity, but rather to reduce it. Mitigation of requirement complexity can be achieved via only enabling autonomy in a certain limited set of situations for which requirements are fully understood (and ensuring that the recognition of those valid operational conditions is correct).

Limiting operational concepts therefore becomes a bootstrapping strategy for deploying successively more sophisticated technical capabilities in a progressively more complex operational context. (e.g., [14, 15]) Once confidence is gained that requirements for a particular operational concept are well understood, additional similar operational concepts can be added over time to expand the envelope of allowable automation scenarios. This will not entirely eliminate the issue of complex requirements, but it can help mitigate the combinatorial explosion of requirements and exceptions that would otherwise occur.

Safety Requirements and Invariants

Even with the use of restricted operational concepts, it seems likely that it will be impractical to use a traditional safety-related requirements approach. Such an approach more or less proceeds as follows. First the functional requirements are created. Then the requirements that are safety-relevant are annotated after some risk assessment process has been performed. Then, these safety-relevant requirements are allocated to safety critical subsystems. Then, safety critical subsystems are designed to satisfy allocated requirements. Finally, unanticipated emergent subsystem interactions are identified and mitigated via repeating the cycle.

Annotation of safety-critical requirements can be impractical for autonomy applications for at least two reasons. One reason is that many requirements might be only partially safety related, and are inextricably entwined with functional performance. For example, the many conditions for operating a parking brake when the car is moving could be a starting set of requirements. However, only some aspects of those requirements are actually safety critical, and those aspects are largely emergent effects of the interaction of the other functions. In the case of the parking brake, a deceleration profile when the parking brake is applied at speed is one of the desired functions, and is likely to be described by numerous functional requirements. But, simplifying, the only safety critical aspect in the deceleration mode might be that the emergent interaction of the other requirements must avoid locking up the wheels during the deceleration process.

The second reason that annotation of requirements to identify safety-relevant requirements may fail is that this may not even be possible when machine learning techniques are used. That is because the requirements, such as they are, take the form of a set of training data that enumerates a set of input values and correct system outputs. These tend not to be in the form of traditional requirements, and therefore require a different approach to requirements management and validation. (See the section on machine learning later in this paper).

Rather than attempting to allocate functional requirements among safety and non-safety subsystems, it can be helpful to create a separate, parallel set of requirements that are strictly safety related. [16] These requirements tend to be in the form of invariants that specify system states that are required for safety (both things that must be true to be safe, and things that must be false to be safe). This approach can disentangle issues of performance and optimization ("What is the shortest traveling path?" or "What is the speed for optimal fuel consumption?") from those of safety ("Are we going to hit anything?").

Using this approach would divide the set of requirements into two parts for the V model. The first set of requirements would be a set of non-safety-related functional requirements, which might be in traditional format or an untraditional format such as a machine learning training set. However, by definition those potentially nontraditional requirements are not safety-related, so it might be acceptable if traceability and validation have ample but imperfect coverage.

The second set of requirements would be a set of purely safety requirements that completely and unambiguously define what "safe" means for the system, relatively independent of the details of optimal system behavior. Such requirements can take the form of safe operating envelopes for different operational modes, with the system free to optimize its performance within the operating envelope. [17] It is clear that such envelopes can be used in at least some situations (e.g., enforcing a speed limit or a setting a minimum following distance). This concept promises to be rather general, but proving that remains future work.

A compelling reason to adopt a set of safety requirements that is orthogonal to functional requirements is that such an approach cleanly maps onto monitor/actuator architectures. Functional requirements can be allocated to a low-ASIL actuator functional block, while safety requirements can be allocated to a high-ASIL monitor. This idea has been used informally for many years as part of the monitor/actuator design pattern. We are proposing that this approach be elevated to a primary strategy for architecting autonomous vehicle designs, requirements, and safety cases rather than being relegated to a detailed implementation redundancy strategy.

NON-DETERMINISTIC AND STATISTICAL ALGORITHMS

Some of the technologies used in autonomous vehicles are inherently statistical in nature. In general, they tend to be non-deterministic (non-repeatable), and may give answers that are only correct to some probability - if a probability can be assigned at all. Validating such systems presents challenges not typically found in more deterministic, conventional automotive control systems.

Challenges of Stochastic Systems

Non-deterministic computations include algorithms such as planners that might work by ranking the results of numerous randomly selected candidates (e.g., probabilistic roadmap planners [18]). Because the core operation of the algorithm is based on random generation of candidates, it is difficult to reproduce. While techniques such as using a reproducible pseudo-random number stream in unit test can be helpful, it may be impractical to create completely deterministic behavior in an integrated system, especially if small changes in initial conditions lead to diverging system behaviors. This means that every vehicle-level test could potentially result in a different outcome despite attempts to exercise nominally identical test cases.

Successful perception algorithms also tend to be probabilistic. For example, the evidence grid framework [19] accumulates diffuse evidence from individual, uncertain sensor readings into increasingly confident and detailed maps of a robot's surroundings. This approach yields a *probability* that an object is present, but never complete confidence. Furthermore, these algorithms are based on prior models of sensor physics (e.g., multipath returns) and noise (e.g., Gaussian noise on LIDAR-reported ranges) which are themselves probabilistic and sensitive to small changes in environmental conditions.

Beyond modeling the geometry of surroundings, other algorithms extract labels from perceived data. Prominent examples of these include pedestrian detection. [20] Such systems can exhibit potentially unpredicted failure modes even with largely noise-free data. For example, vision systems might have trouble disambiguating color variations due to shadows, and experience difficulties determining object positions in the presences of large reflective surfaces. (In all fairness, these present challenges for humans as well.) Moreover, any classification process exhibits a tradeoff between false negatives and false positives, with fewer of one necessarily incurring more of the other. The testing implications of this are that such algorithms won't "work" 100% of the time, and that depending on construction they might report a particular situation as being "true" when it is only a moderately high probability of that situation actually being true.

Non-Determinism in Testing

Handling non-determinism in testing is difficult for at least two reasons. The first is that it can be difficult to exercise a particular specific edge-case situation. This is because the system might behave in a way that activates that edge case only if it receives a very specific sequence of inputs from the world. Due to factors discussed earlier, such as the potentially dramatic differences in planner response to small changes of inputs, it can be difficult to contrive a situation in which the world will reliably offer up just the right conditions to run a particular desired test case.

As a simple example, a vehicle might prefer to drive a more circuitous route on a wide roadway rather than a shortcut through a narrow alley. To evaluate the performance navigating the narrow alley, testers would need to contrive a situation that makes the wide roadway unappealing to the planner. But, doing this requires additional attention to test planning, and perhaps (manually) moving the vehicle into a situation it would not normally enter to force the desired response. Testing the vehicle's ability to consistently choose the better of two almost equally unattractive paths without vacillating might be even more difficult.

A second difficulty with non-determinism in testing is that it can be difficult to evaluate whether test results are correct or not, because there is no unique correct system behavior for a given test case. Thus, correctness criteria are likely to have to take a form similar to the safety envelopes previously discussed, in which a test passes if the end system state is within an acceptable "test pass" envelope. In general, multiple tests might be required to build confidence that the system will always end up in the test pass envelope.

Probabilistic system behaviors present a similar challenge to validation, because passing a test once does not mean that the test will be passed every time. In fact, with a probabilistic behavior it might be expected that at least some types of tests will fail some fraction of the time. Therefore, testing might not be oriented toward determining if behaviors are correct, but rather to validating that the statistical characteristics of the behavior are accurately specified (e.g., that the false-negative detection rate is no greater than the rate assumed in an accompanying safety argument). This is likely to take

a great many more tests than simple functional validation, especially if the behavior in question is safety critical and expected to have an extremely low failure rate.

Achieving extremely high performance from a probabilistic system is likely to require multiple subsystems that in composite are assumed to provide a low aggregate failure rate due to having completely independent failures. For example, a composite radar and vision system might be combined to assure no missed obstacles to within some extremely low probability. This approach applies not only to sensing modalities, but also to other diverse algorithmic schemes in planning and execution. If such an approach is successful, it might well be that the resulting probability of failure is so low that testing to verify the composite performance is infeasible. For example, if obstacles must be missed by both systems once in a billion detections, then billions of representative tests must be run to validate this performance.

Validating very low failure rates for composite diverse algorithms might be attempted by separately validating the more frequent permissible failure rates of each algorithm in isolation. But that is insufficient. One must also validate the assumption of independence between failures, which might well have to be based on analysis in addition to testing.

MACHINE LEARNING SYSTEMS

Proper behavior for autonomous vehicles is only possible if a complex series of perception and control decisions are made correctly. Achieving this usually requires proper tuning of parameters, including everything from a calibrated model of each camera lens to the well-tuned weighting of the risks of swerving versus stopping to avoid obstacles on a highway. The challenge here is to find the calibration model or the ratio of weights such that some error function is minimized. In recent years, most robotics applications have turned to machine learning to do this [21, 22], because the complexities of the multi-dimensional optimization are such that manual effort is unlikely to yield desired levels of performance.

The details of approaches to machine learning are many, e.g., the use of learning from demonstration, active learning, and supervised vs. unsupervised approaches. However, all such approaches involve inductive learning, in which training examples are used to derive a model.

For example, consider the case of detecting pedestrians in monocular images. Using a large training set of images, a classifier can learn a decision rule that minimizes the probability that pedestrians are detected in a separate validation set of images. For our purposes, an essential element is that the training set is effectively the set of requirements for the system, and the rules are the resultant system design. (The machine learning algorithm itself and the classifier algorithm are both more amendable to traditional validation techniques. However, these are general-purpose software "engines" and the ultimate system behaviors are determined by what training data is used for learning.)

One could attempt to skirt the issue of training set data forming de facto requirements by instead creating a set of requirements for collecting the training data. But this ends up simply pushing the same challenge up one level of abstraction. The requirements are not in the typical V format of a set of functional requirements for the system itself, but rather in the form of a set of training data or a plan for collecting the set of training data. How to validate training data is an open question that might be addressed by some combination of characterizing the data as well as the data generation or data collection processes.

Challenges of Validating Inductive Learning

The performance of inductive learning methods can be tested by holding back some samples from the overall data set that has been collected and using those samples for validation. The presumption is that if the training set is used as the system requirements (the left-hand side of the V) an independent set of validation data can be used to ensure that the requirements have been met (forming the corresponding right-hand side of the V). Training data must not have accidental correlations unrelated to the desired behavior, or else the system will become "over-fitted." Similarly, the validation data must be independent and diverse from the training data in every way except the desired features, or else overfitting will not be detected during validation. It is unclear how to argue that a machine learning system has not been over-fitted as part of a safety argument.

A significant limitation of machine learning in practice is that if labelled data is used, each data point can be expensive. (Creating labels has to be done by someone or something. Unsupervised learning techniques are also possible, but require a clever mapping to solving a particular problem.) Moreover, if a problem with the training set (i.e., a requirements defect) or the learned rules (i.e., a design defect) is found and corrected, then more validation data has to be collected and used to validate the updated system. This is necessary because even a small change to the training data could produce a dramatically different learned rule set. Thus, complete revalidation would normally be required for any training set "bug fix," no matter how small.

Because of the complexity of requirements for an autonomous system, it seems likely that rare, edge cases will be where learning problems would be expected to occur. However, because of their rarity, collecting data depicting such unusual circumstances can be expensive and difficult to scale. (Simulation and synthetic data can help with this, but come with the risk of bias in simulated data, as well as overfitting to simulation artifacts.)

Another issue with validating machine learning is that, in general, humans cannot intuitively understand the results of the process. For example, the internal structure of a convolutional neural network [23] may not tell a human observer much intuitive about the decision rules that have been learned. While there might be some special cases, in general the problem of "legibility" [24, 25] of machine learning in terms of being able to explain in human terms how the system behaves is unsolved. This makes it difficult to predict how techniques other than expensive brute force testing can be applied for validation of machine learning systems. (Perhaps some organizations do have

the resources to do extensive brute force testing. But even in this case the accuracy, validity, and representativeness of the training data must be demonstrated as part of any safety argument based on the correctness of a machine learning system.)

Because legibility for machine learning systems is generally poor, and because the danger of overfitting is real, there are failure modes in such a system that can significantly affect safety. Of particular concern are accidental correlations that are present in training set data but not noticed by human reviewers. For example, consider the method of detecting pedestrians in imagery using trained deformable-part models, which has been shown to be quite effective in real-world data sets. [26] If no (or few) images of pedestrians in wheelchairs were present in the training data set, it is likely that such a system would incorrectly correlate the label of "pedestrian" with "people who walk on two legs."

Solutions to Inductive learning

Validating inductive learning is notoriously difficult due to the "black swan" problem [27], which is in general the susceptibility of a person (or system) to believe that common observations are true, and draw potentially incorrect conclusions due to an abundance of confirming data points. The story goes as follows. Before the late 1700s, all observed swans in Europe were white, and thus an observer using inductive logic would have concluded that all swans are white. However, this observer would experience a brittle failure of this belief when visiting Australia, where there are plenty of black swans. In other words, if there is a special case the system has not seen, it cannot learn that case. This is an essential limitation to inductive learning approaches that is not readily cured. [28] Moreover, with machine learning this problem is compounded by the lack of legibility, so it can be difficult or impossible for human reviewers to imagine what form a black swan-like bias in such a system might take.

Validating an inductive learning system seems to be an extremely challenging problem. Extensive testing might be used, but would require validating an assumption of random independent arrival rates of "black swan" data and testing on data sets sized accordingly. This might be feasible given enough resources, but there will always be new black swans, so a probabilistic assessment of huge numbers of operational scenarios and input values would have to be made to ensure an acceptably low level of system failures. (If resources were available to do this in a defensible way, this might suffice to form the right-hand-side of a V process.)

An alternative to validating inductive learning systems to high ASIL levels would be to pair a low-ASIL inductively-based algorithm that sends commands to an actuator with a high-ASIL deductively-based monitor. This would sidestep the majority of the validation problem for the actuation algorithm, since failures of the inductive algorithm controlling the actuator would be caught by a non-inductive monitor based on a concept such as a deductively-generated safety envelope. Thus, actuator algorithm failures would be an availability problem (the system safety shuts down, assuming an adequate failover capability) rather than a safety problem.

MISSION CRITICAL OPERATIONAL REQUIREMENTS

As a final technical area, we return to the previously discussed point that the computer is ultimately in control of the vehicle rather than the person being in control. That means that at least some portion of the vehicle has to be fail-operational rather than fail-stop.

Challenges of Fail-Operational System Design

Fail operational system design has been done successfully in aerospace and other contexts for decades, but is still difficult for several reasons. The first reason is the obvious one that redundancy has to be provided so that when one component fails another one can take over. Achieving this requires at least two independent, redundant subsystems for fail-stop behavior.

Achieving a fail-operation system in turn requires at least three redundant fail-arbitrary components so that it can be determined which of the three failed in the event that it issues incorrect outputs rather than failing silent at the component level. [29] For systems that have to tolerate arbitrarily bad faults, a Byzantine fault tolerant system with four redundant components might be required [30], depending on the relevant fault model.

The structure of the redundancy varies depending on the design approach, and might include configurations such as a triplex redundant system with a voter (in which case the voter must be ensured not to be a single point of failure), or a dual two-of-two system that uses four computers in fail-silent pairs. [29] Beyond the obvious expense such approaches introduce, there is also an issue of testing to make sure that failure detection and recovery works, assuring independence of failure, and ensuring that all redundant components are fault-free at the start of a driving mission. It seems unlikely that redundancy can be avoided, but it may be possible to reduce the complexity and expense of providing sufficient redundancy to ensure safety.

Failover Missions

In typical fail-operational system such as aircraft, all the redundant components are essentially identical and capable of performing an extended mission. For example, commercial aircraft are commonly configured with two jet engines, and each jet engine has at least a dual-redundant computer control. If the pair of computers on one engine shuts down due to a fault detected via continual crosschecking, there is a second independent engine to keep the aircraft flying. Even so, the requirements on engine dependability are very stringent, because aircraft might potentially have to fly several hours after a first engine failure to reach the nearest airport without having the second engine fail. This puts significant reliability requirements on each engine, and therefore increased component costs.

While cars are notoriously cost-sensitive, they do have an advantage in that failover missions can be short (e.g., pull over to the side of the road, or if necessary come to a stop in a travel lane), with failover mission durations measured in seconds rather than hours. Additionally, a failover mission to stop the vehicle might be able to

operate with significantly less functionality than fully autonomous operation. This can simplify requirements complexity, computational redundancy, sensor requirements, and validation requirements. (As a simple example, a failover mission control system might not support lane changes, greatly simplifying sensor requirements and control algorithms. More sophisticated approaches that are still simpler than full autonomy might be possible.) Therefore, designing an autonomous vehicle with a fail-stop primary controller and a simpler fail-operational failover controller might be attractive both in terms of hardware cost and in terms of design/validation cost.

It might also be that a safety argument can be created not based on the full autonomy system being perfect, but rather on the full autonomy system having a detector that realizes when it is malfunctioning or has encountered a gap in its requirements. This would make the fault detector itself high-ASIL, but might permit normal autonomy functions to be low-ASIL. Such an approach would map well onto a monitor/actuator architecture for the primary autonomy system. The failover autonomy would also have to be designed in a safe manner, with an appropriate architectural approach depending on its complexity and calculated reliability requirements. It might even be possible to use a single-channel failover system if the probably of failure during a short failover mission lasting only seconds is sufficiently low.

NON-TECHNICAL FACTORS

Some challenges in deploying autonomy are non-technical, such as the frequently mentioned liability problem (who pays when there is a mishap?) and how laws generally treat the ownership, operation, maintenance, and other aspects such vehicles.

A deep dive into this topic is beyond the scope of this paper. However, resolutions to non-technical challenges will very likely have an impact on technical solutions. For example, there may be forensic requirements imposed on autonomy systems for accident reconstruction data. Careful analysis of the provenance of such data will need to be performed to ensure that the data is used properly. As a simple example, if a radar has a hypothetical detection probability of 95%, its output might still be recorded in the system in terms of whether an obstacle was or was not detected, superficially implying detection certainty. It is important to ensure that forensic analysis takes into account that just because the radar didn't detect a pedestrian does not mean the pedestrian was not there (e.g., a 95% detection probably implies that 1 out of 20 pedestrians will not actually be detected).

It seems likely that with the inherent complexity of an autonomous vehicle and the clear inability to demonstrate anything close to perfection via testing, it will be important for developers to create a safety assurance argument in the form of an assurance case (e.g., according to [31]). Such an assurance argument will be necessary to defend and explain the integrity of their system and be able to credibly explain the system's responses to events surrounding the inevitable mishaps that will occur. A particular point that should be addressed is ensuring the integrity of evidence to establish whether a mishap was reasonably unavoidable due to its circumstances. Other important points will be whether or not a mishap was arguably caused by a defect in system requirements (e.g., a gap in training data), a reasonably foreseeable and avoidable design defect, an implementation defect, or other cause attributable to the vehicle manufacturer.

FAULT INJECTION

As is apparent from the preceding discussion, traditional functional testing will have trouble exercising a complete system, and especially will find it difficult to exercise combinations of exceptions occurring during unusual operational conditions. While testers can define some off-nominal test cases, scalability of that testing is questionable due to the combinatorial explosion of exceptions, operational scenarios, and other factors involved. Additionally, it has been shown that even very good designers often have blind spots and miss exceptional situations in comparatively simple software systems. [32]

Fault injection and robustness testing are relatively mature technologies for assessing the performance of a system under exceptional conditions [33], and can help avoid designer and tester blind spots when testing exceptional condition responses. Traditional fault injection involves inserting bit flips into memory and communication networks. More recent techniques have increased the level of abstraction to include data-type-based fault dictionaries [32], and ensuring fault representativeness [33]. Such techniques have already been used successfully to find and characterize defects on autonomous vehicles. [35]

A promising approach to helping validate autonomy features is to perform fault injection at the level of abstraction of the component, as part of a strategy of attempting to falsify claims of safety. [36] This involves not only simulating objects for primary sensor inputs, but also inserting exceptional conditions to test the robustness of the system (e.g., inserting invalid data into maps). The point of doing such fault injection is not to validate functionality, but rather to probe for weak spots that might be activated via unforeseen circumstances. Such fault injection can be performed across the range of layers in the ISO 26262 V process. [37]

CONCLUSIONS

The challenges of developing safe autonomous vehicles according to the V process are significant. However, ensuring that vehicles are safe nonetheless requires following the ISO 26262 V process, or demonstrating that a set of process and technology practices equally rigorous has been applied. Assuming that the V process is applied, there are three general approaches that seem promising.

Phased Deployment

It appears impractical to develop and deploy an autonomous vehicle that will handle every possible combination of scenarios in an unrestricted real-world environment, including exceptional situations, all at once. Rather, as is common in automotive systems, a phased deployment approach building on current developer practice seems likely to be a reasonable approach.

Tying phased deployment to the V process can be done by identifying well-specified operational concepts to limit the scope of operations and therefore the necessary scope of requirements. This would include limitations in environment, system health, and operational constraints that must be satisfied to enable autonomous operation. Validating that such operational constraints are enforced will be an essential part of ensuring safety, and will have to show up in the V process as a set of operational requirements, validation, and potentially run-time enforcement mechanisms. For example, run-time monitoring might be required to monitor not only whether system state is in a permissible autonomy regime, but also that assumptions made about the operational scenario in the safety argument are actually being satisfied, and whether the system is actually in the operational scenario it thinks it is in.

An aspect of restricted operational concepts that will require particular attention is ensuring that safety is maintained when an operational scenario suddenly becomes invalidated, due to for example an unexpected weather event or an infrastructure failure. Such exceptional transitions out of an acceptable operational concept regime will require that system recovery or a failover mission be executed successfully even when there is a system excursion outside the assumptions of permissible autonomy operational scenarios.

It is unclear whether a phased deployment approach will provide a path all the way to complete autonomy. But at least such an approach provides a way to make progress and gain some benefits of autonomy while gaining greater understanding of the difficult edge cases and unanticipated scenarios that will arise as systems see more exposure to real world conditions.

Monitor/Actuator Architecture

A common approach that might help mitigate many of the challenges of autonomous vehicle safety is the use of a monitor/actuator architecture. As discussed, this architectural style can help with requirements complexity (only the monitor needs to be essentially perfect), and deployment of inductive algorithms (by limiting use of induction to the actuator, and using a deductively-based monitor).

Additionally, the use of a failover mission strategy can allow a primary autonomy system monitor to detect a primary system failure without having to ensure fail-operational behavior. A simpler, high-integrity failover autonomy system can bring the vehicle to a safe state. Such a system might have a failover mission short enough that minimal redundancy for failover operation is required, so long as it can be assured that the system is fault-free when it is time to start a failover mission.

Fault Injection

Testing alone is infeasible to ensure ultra-dependable systems. Autonomous vehicles only make this problem harder by automating responses to highly complex environmental situations, and introducing technology such as machine learning that is difficult and expensive to test. Moreover, because much of the autonomy capability must have a high ASIL due to the lack of human driving oversight, it seems difficult to do enough ordinary system testing to gain even a reasonable level of assurance.

Fault injection can play a useful role as part of a validation strategy that also includes traditional testing and non-test-based validation. This is especially true if fault injection is applied at multiple levels of abstraction rather than just at the level of stuck-at electrical connectors.

Future Work

This paper discusses ways to fit autonomous vehicle safety assurance within an ISO 26262-based V framework. However, it is expected that using architectural patterns such as the monitor/actuator approach and the practical limits of validation possible via fault injection will place constraints on operational performance. In other words, the functionality of autonomous vehicles might need to be limited to fit the constraints of feasible validation techniques. Relaxing those constraints will require advances in areas such as characterizing the coverage of machine learning training data compared to the expected operational environment, gaining confidence in safety requirements with regard to exceptional driving conditions, and being able to validate the independence of failures in redundant inductive-based systems.

REFERENCES

1. Transportation Research Board, National Automated Highway System Research Program: a review, TRB Special Report 253, National Academy Press, Washington DC, 1998.
2. NHTSA, Preliminary Statement of Policy Concerning Automated Vehicles, May 2013, http://www.nhtsa.gov/staticfiles/rulemaking/pdf/Automated_Vehicles_Policy.pdf, accessed Oct. 2015.
3. US Department of Transportation, FAA, Advisory Circular, System design and analysis, AC 25.1309-1A, June 21, 1988.
4. Butler & Finelli, “The infeasibility of experimental quantification of life-critical software reliability,” IEEE Trans. SW Engr. 19(1):3-12, Jan 1993.
5. Motor Industry Software Reliability Association, Development Guidelines for Vehicle Based Software, November 1994.
6. Motor Industry Software Reliability Association, Report 6: Verification and Validation, February 1995.
7. Road vehicles -- Functional Safety -- Part 2: Management of functional safety, ISO 26262-2:2011, Nov. 15, 2011.
8. Road vehicles -- Functional Safety -- Part 3: Concept Phase, ISO 26262-3:2011, Nov. 15, 2011.
9. Road vehicles -- Functional Safety -- Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses, ISO 26262-9:2011, Nov. 15, 2011.
10. Randell, B., “System structure for software fault tolerance,” IEEE Trans. SW Engineering, SE-1:2, June 1975, pp. 1-18.
11. Lions, J., ARIANE 5 Flight 501 Failure, Report by the Inquiry Board, Paris, July 1996.
12. Yeh, Y.C.; “Design considerations in Boeing 777 fly-by-wire computers,” HASE 1998.
13. Wereschagin, M., “'Pittsburgh Left' seen by many as a local right,” TribLive News, June 14, 2006, http://triblive.com/x/pittsburghtrib/sports/s_457936.html
14. Bayouth, M. & Koopman, P., “Functional Evolution of an Automated Highway System for Incremental Deployment,” Transportation Research Record, #1651, Paper #981060, pp. 80-88.
15. Shladover, S. et al., Development and performance evaluation of AVCSS deployment sequences to advance from today's driving environment to full automation, California PATH Research Report UCB-ITS-PRR_2001-18, August 2001.
16. Black, J. & Koopman, P., “System safety as an emergent property in composite systems,” DSN 2009, pp. 369 - 378.

17. Kane, Chowdhury, Datta & Koopman, "A Case Study on Runtime Monitoring of an Autonomous Research Vehicle (ARV) System," RV 2015.
18. Geraerts, R., Overmars, M. H., "A comparative study of probabilistic roadmap planners," Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02), 2002, pp. 43-57.
19. Martin C., Moravec H., Robot Evidence Grids, tech. report CMU-RI-TR-96-06, Robotics Institute, Carnegie Mellon University, March, 1996.
20. Dollár, P. et al., "Pedestrian Detection: An Evaluation of the State of the Art," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 34, No. 4, 2012.
21. Silver, D., Bagnell, J, Stentz, A., "Active Learning from Demonstration for Robust Autonomous Navigation," IEEE Conference on Robotics and Automation, May, 2012.
22. Dima, C., Active Learning for Outdoor Perception, doctoral dissertation, tech. report CMU-RI-TR-06-28, Robotics Institute, Carnegie Mellon University, May, 2006
23. Krizhevsky, A., Sutskever, I., Hinton, G.E., "ImageNet classification with deep convolutional neural networks." In NIPS, pages 1106-1114, 2012.
24. Dosovitskiy A., Brox, T., "Inverting convolutional networks with convolutional networks," CoRR, vol. abs/1506.02753, 2015.
25. Zeiler, M. D., Fergus, R., "Visualizing and understanding convolutional networks." In ECCV, 2014.
26. Felzenszwalb, P., Girshick, P., McAllester, D., Ramanan, D., "Object Detection with Discriminatively Trained Part Based Models," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, No. 9, Sep. 2010.
27. Taleb, N., The Black Swan: the impact of the highly improbable, Random House, 2007.
28. Hume, D. (1910) [1748]. An Enquiry concerning Human Understanding. P.F. Collier & Son. ISBN 0-19-825060-6.
29. Hammet, "Design by extrapolation: an evaluation of fault-tolerant avionics," IEEE Aerospace and Electronic Systems, 17(4), 2002, pp. 17-25.
30. Lamport, L., Shostak, R., Pease, M., "The Byzantine Generals Problem," Trans. Prog. Lang. and Sys. 4(3):382-401, ACM, July 1982.
31. Systems and Software Engineering -- Systems and Software Assurance -- Part 2: Assurance Case, ISO/IEC 15026:2011.
32. Koopman, P., DeVale, K. & DeVale, J., "Interface robustness testing: experiences and lessons learned from the Ballista Project," In: Kanoun, K. & Spainhower, L., Eds., Dependability Benchmarking for Computer Systems, IEEE Press, 2008, pp. 201-226.
33. Kanoun, K. & Spainhower, L., Eds.,Dependability Benchmarking for Computer Systems, IEEE Press, 2008, pp. 201-226.
34. Natella, R., Cotroneo, D., Duraes, J., Madeira, H., "On fault representativeness of software fault injection," IEEE Trans. SW Engineering, 39:1, pp. 80-96, Jan. 2013.
35. Vernaza, P., Guttendorf, D., Wagner M., Koopman, P., "Learning Product Set Models of Fault Triggers in High-Dimensional Software Interfaces," IROS 2015.
36. Wagner M., Koopman, P., "A Philosophy for Developing Trust in Self-Driving Cars," In: Meyer G. & Beiker S. (eds.) Road Vehicle Automation 2, Lecture Notes in Mobility, Springer, 2014, pp. 163-170.
37. Pintard, L., Fabre, J.-C., Kanoun, K., Roy, M., Leeman, M. "Fault Injection And Automotive Development Process," Embedded Real-Time Software And Systems (ERTS2), Fev. 2014, Toulouse, France.

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 National Robotics Engineering Center (NREC) and the Institute for Software Research. E-mail:

koopman@cmu.edu.

Michael Wagner is the 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 has an affiliation with the National Robotics Engineering Center. E-mail:

mwagner@edge-case-research.com

DEFINITIONS/ABBREVIATIONS

ADAS - Advanced Driver Assistance System

ASIL - Automotive Safety Integrity Level

HOV - High Occupancy Vehicle

LIDAR - Light Detection and Ranging

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"