

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 10⁶ 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].

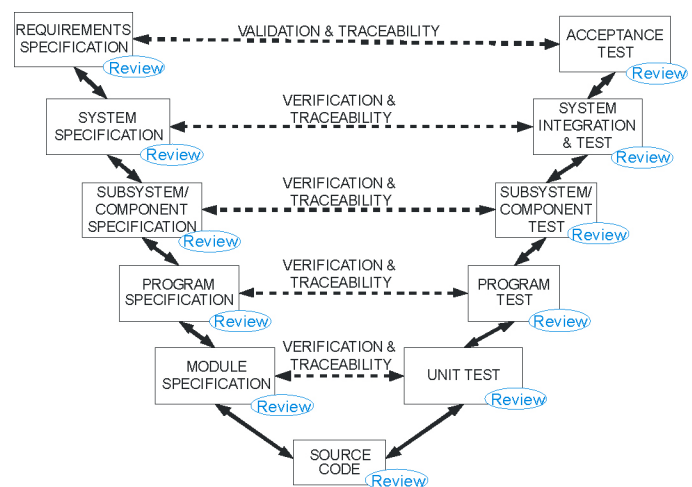


Figure 1. A generic V model.

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).

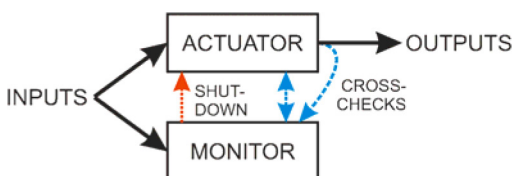


Figure 2. Monitor/actuator pair conceptual diagram.

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 Ariane 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