Fault Injection Testing for Automotive Software

Introduction

Car systems are becoming ever more complex, while also taking more responsibility for safety-related tasks, such as adaptive cruise control and the widely-predicted advent of autonomous vehicles. To provide the necessary confidence in such automotive systems, they are designed to be fault tolerant and fail-safe. Positive testing should provide confidence that the system's functional requirements are met, but for safety-related systems we also need to be sure that when the unexpected does occur the system is truly fault tolerant or will fail to a safe state. One way we address this is to perform negative testing in the form of fault injection testing - we deliberately inject faults into the system to assure ourselves that it reacts in the way we expected (i.e. safely).

Sometimes these scenarios *should* never occur (e.g. a software task should never 'die' or get stuck in an infinite loop) and cannot be simulated by normal system testing, but with fault injection testing, we create the fault scenario and measure the subsequent system behaviour to ensure it is safe.

Fault injection testing has been around for well over 30 years, and tests the dependability or robustness of a system in the presence of faults in the system's environment (e.g. a faulty power supply, badly-formed input messages) and faults in the system itself (e.g. a flipped bit caused by cosmic radiation, poor design and bad coding). It can be applied throughout the lifecycle (from early modelling to the testing of the complete system) and typically supplements the more normal test techniques by targeting that code that is only there to handle exceptional situations and so increases test coverage. When applied early in the lifecycle it is typically used to improve the design to cope with unexpected faults, while later in the lifecycle it is used to obtain confidence that mechanisms for achieving dependability are working, and to find defects in the implementation of the fault tolerant design.

Fault Injection Testing in ISO 26262

The ISO 26262 set of standards was first published in 2011, and applies to the full safety lifecycle (concept, development and operation) of electrical and electronic systems in production-line passenger cars. These standards are part of the family of safety standards based on the generic IEC 61508 standard, which was first published in 1998.

Fault injection testing is recommended by ISO 26262 as shown in Appendix A. At the system and hardware levels (parts 4 and 5), these requirements are explicitly focused on the implementation of the safety requirements and the effectiveness of fault detection mechanisms. For the software (part 6), the focus on safety is less obvious, and at the unit level it is implied (perhaps through the requirement to demonstrate robustness). At the software integration level, the requirement is similarly hazy, but the note on fault injection testing explicitly states that it is for testing safety mechanisms.

Fault Detection and Fault Tolerance

There are several possible outcomes when a fault occurs in a safety-related system, as shown in Figure 1. In the first scenario, a fault occurs and goes undetected for the duration of the 'fault detection time'. Once detected, the system reacts and recovers to the safe state, which could be a return to normal operation or some degraded, but safe, level of operation – this system is demonstrating a level of fault tolerance (if we had injected the fault, then it would have passed the test). In the second scenario, the fault goes undetected until the fault tolerant time interval is exceeded. At this point the system enter the hazardous state as nothing was done to manage the fault. In the third scenario, the fault is detected, but the system fails to react in a way that prevents the system entering the hazardous state (this could be because the recovery takes too long or the recovery was not correctly implemented). In the latter two scenarios, the system's fault tolerance appears to be lacking – and, if we had injected the fault, the system would have failed the tests.

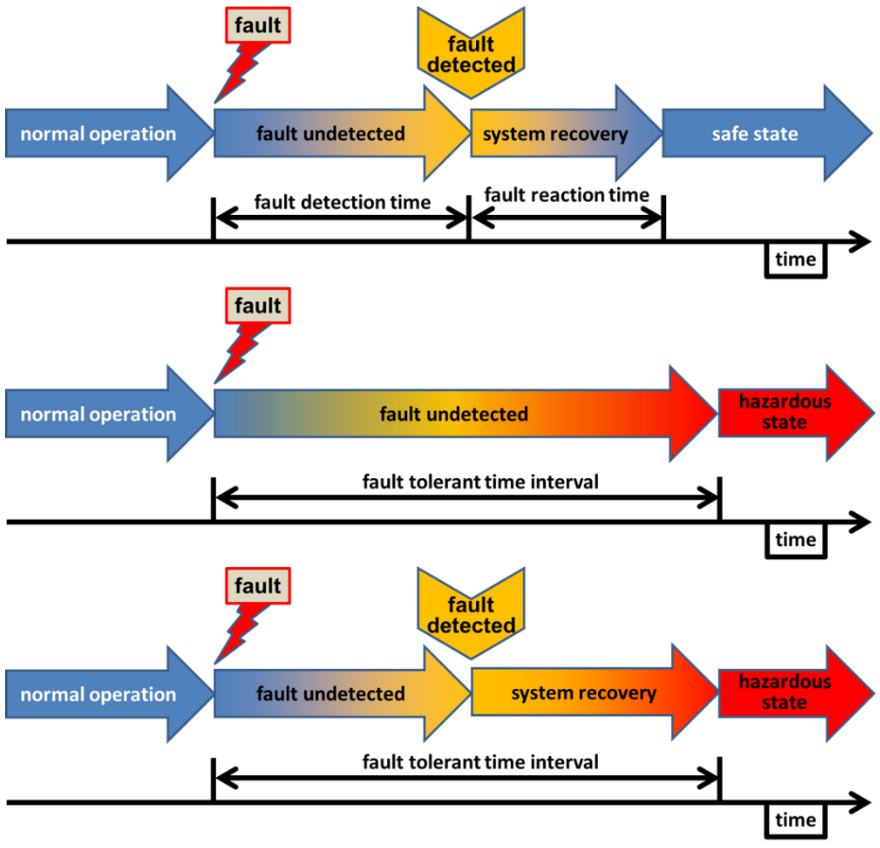

Figure 1: Moving to a safe state (or not)

It is clear from these scenarios that fault injection testing is focused on both the fault detection and recovery attributes (robustness) of the system.

Fault Injection Testing Process

A fault injection process aligned with ISO 26262 is shown in Figure 2. The first three activities correspond to activities that are performed as part of the Concept Stage of the safety lifecycle defined in ISO 26262-2. For fault injection, we need to know which safety requirements are allocated to which parts of the architecture because our fault injection testing is primarily aimed at checking that the safety requirements are correctly implemented.

Once we know our safety requirements and the architecture of the system, we can more easily identify fault types that correspond to these safety requirements (those faults that could possibly occur and cause the system to fail).

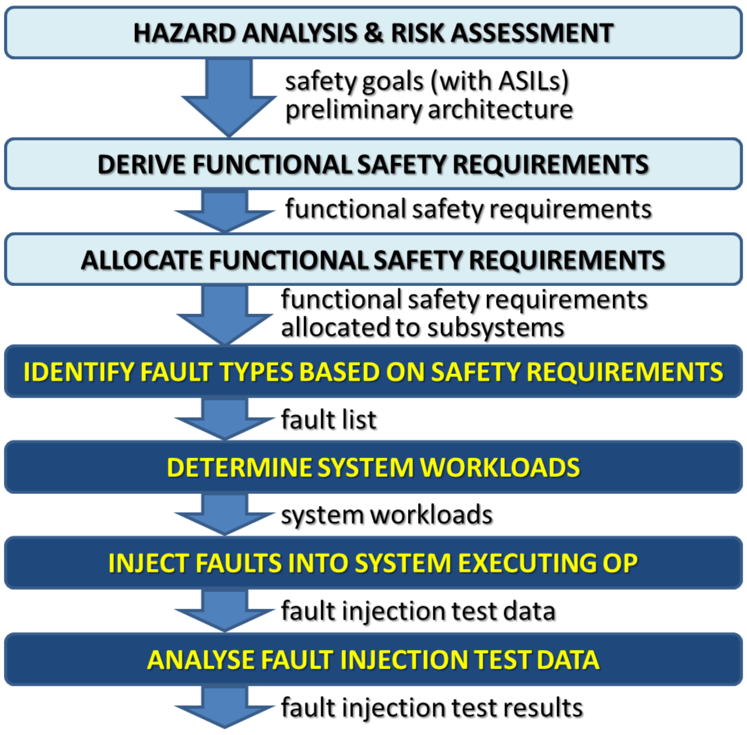
HAZARD ANALYSIS & RISK ASSESSMENT
safety goals (with ASILs)
preliminary architecture
DERIVE FUNCTIONAL SAFETY REQUIREMENTS
functional safety requirements
ALLOCATE FUNCTIONAL SAFETY REQUIREMENTS
functional safety requirements
allocated to subsystems
IDENTIFY FAULT TYPES BASED ON SAFETY REQUIREMENTS
fault list
DETERMINE SYSTEM WORKLOADS
system workloads
INJECT FAULTS INTO SYSTEM EXECUTING OP
fault injection test data
ANALYSE FAULT INJECTION TEST DATA
fault injection test results

Figure 2: High-Level Fault Injection Process

To run the fault injection tests, we need to know the typical workloads that the system will be running once operational (a workload is a set of tasks or functions that reflects the processing capacity and demand on the system), so that we can ensure our tests are run in a representative situation. Otherwise we waste time running tests in unrealistic situations that provide little useful information.

With a fault list and the workloads defined, we can now prepare the test environment and run the fault injection tests. We will collect and analyse the test data from these tests to determine whether the implemented safety mechanisms met the safety requirements. Sometimes, before we inject any faults, we will first perform a 'golden run' and record data representing the system running with no defects – and we can then compare this 'golden run' data with the data that comes from running the system with injected faults. This allows us to measure how well the system recovers from the injected faults.

Fault Injection Test Environment

To perform fault injection testing, we need a special form of test environment, as shown in Figure 3 – this supports the last two activities shown in Figure 2. Before any testing can occur, the fault list and system workloads need to be available. The controller manages the fault injection tests by activating the Fault Injector and the Workload Generator, which inject faults into the System under Tests from the Fault List and run the System under Tests under the pre-defined workloads. The

Monitor tracks the test execution and records test data, as appropriate. The Analyzer is used to perform test data processing and analysis.

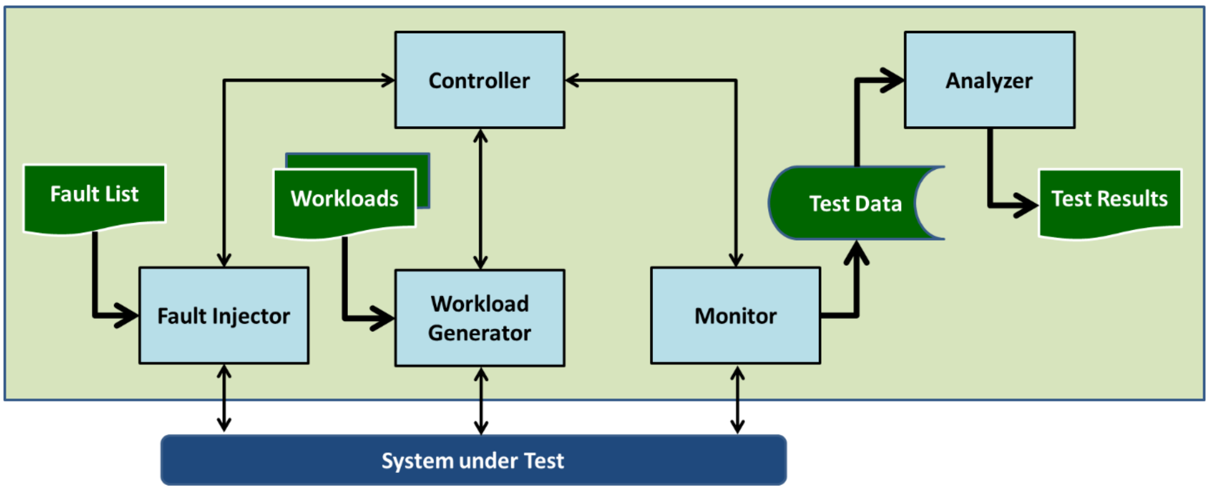

Figure 3: Fault Injection Test Environment

Fault Injection Approaches and Techniques

At a high level, we can consider two main approaches to fault injection. In the first, as shown in Figure 4, we inject faults into the environment of the system under test (typically into the input stream). The goal is to evaluate the ability of the system to handle unexpected inputs, caused by a fault in the environment of the system under test, such as a poor communication channel. This approach is especially useful for the fault injection testing of COTS software, where we don't have access to the source code. Many of the traditional software testing techniques, such as Equivalence Partitioning, Boundary Value Analysis, Syntax Testing and Fuzz Testing are useful with this approach.

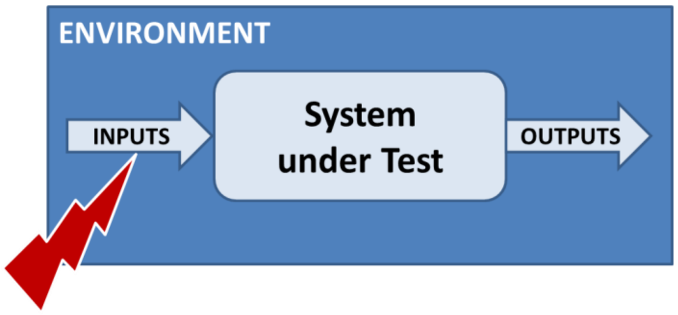

Figure 4: Corrupting the Environment

In the second, shown in Figure 5, we inject faults directly into the system under test. The goal here is focused on testing the internal fault tolerance mechanisms.

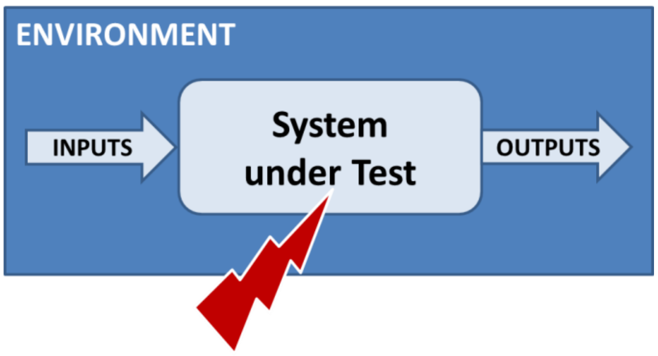

Figure 5: Corrupting the System under Test

There are several different techniques used to inject the faults into the system under test, as shown in Figure 6.

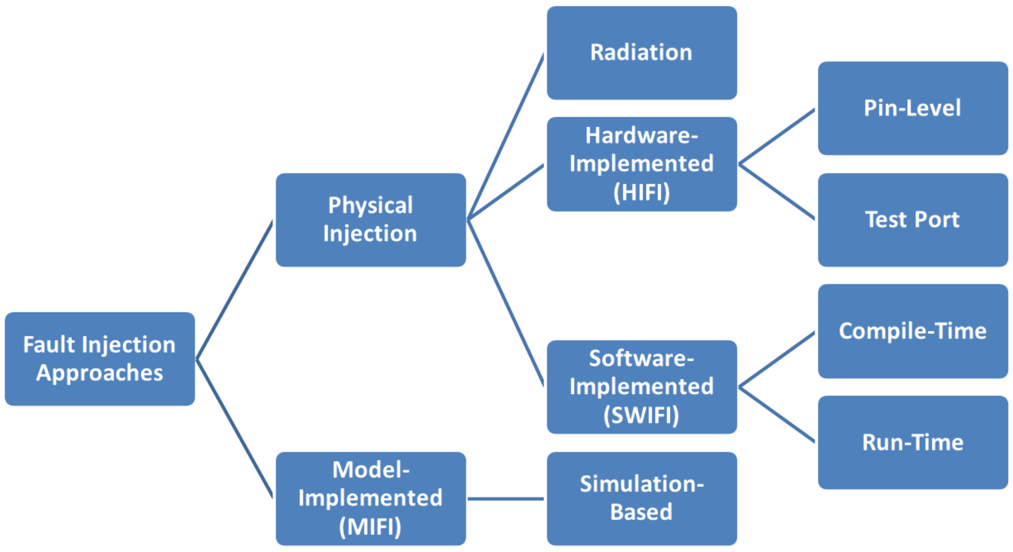

Figure 6: Fault Injection Techniques

These techniques can be characterized by their:

- controllability (ability to control when and where faults are injected);
- repeatability (ability to obtain the same result from the same test);
- fault representativeness (how accurately the injected fault represents real-world faults);
- workload representativeness (how accurately the workloads represent real-world workloads);
- observability (ability to observe and measure the effects of an injected fault);
- intrusiveness (impact on the target system);
- reachability (how much of the system is accessible for fault injection).

Each of the techniques is briefly described below:

Radiation-Based Fault Injection

This is used to test the sensitivity (and the devices low-level error detection and correcting ability) of the integrated circuits in the system to various forms of external radiation, such as electromagnetic interference and particle radiation. It is not used very often due the drawbacks of low controllability and lack of repeatability (it is difficult to control which bits of the system are affected).

Pin-Level Fault Injection

Faults are injected on the electrical contacts ('pins') of integrated circuits or hardware components using probes. This may be done with the original contact still connected (you may get two signals overlapping and there is greater potential for damaging the integrated circuit) or with it disconnected (you only get the injected signal, but requires more effort).

Faults injected using this approach are typically stuck-at, bridging contacts, noise, reduced/increased voltage, etc. With power supply disturbance fault injection, we can test the possibility of common cause failures due to power supply problems.

Test Port Fault Injection

Many microprocessors include test ports (or test access ports) that can be used to inject faults – these are also referred to as on-chip debugging (OCD) ports. These test ports are well-defined by standards (e.g. the Nexus standard or IEEE-ISO 5001-2003). Typically, faults can be injected into microprocessor instruction set registers and into main memory.

Injecting a fault via a test port typically involves four steps:

1. Setting a breakpoint in the software on the microprocessor via the test port and waiting for the program to reach the breakpoint;
2. Reading the value of the target location (a register or word in memory) via the test port;
3. Changing this value to a new, faulty value;
4. Resuming program execution via a command sent to the test port.

Software-Implemented Fault Injection (SWIFI)

SWIFI techniques are widely used, as they are relatively easy to deploy. They can be divided into compile-time techniques (the fault is injected into the software before it is deployed to the system under test) and run-time techniques, where the faults are injected during execution of the system under test.

Compile-Time SWIFI

With compile-time fault injection, faults are put into the source, assembly or binary code of the software under test to simulate the effects of hardware or software faults (e.g. the original code is mutated). When compiled, this results in a faulty software executable, which, when it runs, activates the fault. Test execution with compile-time SWIFI is nearly always faster than with run-time SWIFI, however, the overall time taken for this form of fault injection testing can often be longer due to the effort required to decide and perform the mutation. There is also the problem that the software being tested is (necessarily) different from the software that is eventually delivered.

Run-Time SWIFI

Run-Time SWIFI requires a mechanism that:

1 stops the execution of the system under test;

2 calls a fault injection routine; and

3 restarts the system under test.

There are two basic mechanisms (or triggers) for deciding when to inject a fault into the software during run time:

- time-based triggers - when the timer reaches a specified time, an interrupt is generated and the corresponding interrupt handler associated with the timer injects the fault;
- interrupt-based triggers - when a specific memory location is reached or a specific event occurs then hardware exceptions and software trap mechanisms are used to generate an interrupt (which injects the fault).

Simulation-Based Fault Injection or Model-Implemented Fault Injection (MIFI)

This technique is used earlier in the lifecycle to ensure that the models used as part of model-based development (MBD) correctly implement the required safety mechanisms (and so allows them to be fixed early). The faults are injected into models (e.g. Simulink) from which the source code is automatically generated to test the behaviour of the model in the presence of faults.

Fault Injection Tools

As fault injection testing has been used for many years, there are many tools (commercial, open source and research-based) available to support it in its different forms. The following list provides examples with a mix of technologies and techniques, but is, by no means, an exhaustive list and you should be sure to do your own systematic selection before adopting a tool.

- BTC EmbeddedPlatform, by BTC Embedded Systems
- Cadence® Incisive® functional verification platform by Cadence Design Systems, Inc.
- csXception by Critical Software
- Mx-Suite™ by Danlaw
- EHOOKS & LABCAR by ETAS
- CodeSonar by GrammaTech
- OneSpin Safety Critical Verification Solution & Fault Injection App (FIA™) by OneSpin
- Fault Injection Engine (FIE™), CosmicASIC and CosmicFault by Optima
- Development Testing Platform (DTP) by Parasoft

- SafeTI Hitex Safety Kit by Texas Instruments
- Fault Injection Tester by Suresoft Technologies, Inc
- Z01X Fault Simulation Solution by Synopsys
- Simics by WindRiver
- Safety Verifier by Yogitech
- CANoe & VectorCAST/Probe by Vector
- ESACS, FAIL, FERRARI, FIAT, FISCADE, GOOFI-2, Grinder, ISAAC, MESSALINE, MODIFI, RIFLE, (Research/Open Source tools)

ISO 26262 and Fault Injection Testing

Many fault injection approaches are based on using tools that randomly generate faults and then monitor whether (or by how much) the resultant faulty system differs from the 'golden' system with no faults. The ISO 26262 standard suggests the injection of arbitrary faults for software unit testing and software integration testing, which suggests that this random generation of faults is required. A problem with this approach is that even when many faults are injected into systems under many different workloads only a small proportion of all combinations are tested – and those that may cause a safety problem can easily be missed. Alternatively, the faults can be systematically generated based on knowledge of the safety goals, but this can be expensive if done manually and needs tool support. At the software unit and integration testing levels, fault injection based on corruption the software under test is not really possible, so for these two levels the fault injection testing should be restricted to approaches based on corrupting the environment of the software under test.

For the testing of the complete embedded software system, ISO 26262 does not specifically require any test methods, but, instead, suggests that tests previously created for unit and integration testing be used. For fault injection testing this could be seen as limiting, as the safety requirements at this level are different and covering these safety requirements would often require the use of new, different fault injection test cases. At this level, it is also possible to use the approach of corrupting the system under test alongside corrupting the environment.

ISO 26262 suggests that fault injection testing for hardware-software integration testing should use "special means to introduce faults into the test object during runtime". It is unclear why the standard restricts the fault injection to runtime approaches when other approaches, such as compile time fault injection, are also available (and appropriate in the right conditions).

ISO 26262 suggests that fault injection testing be used for improving the coverage of safety requirements (at various stages in the safety lifecycle), and seems to ignore the possibilities of it being used during the design stages to validate and improve the design of the safety mechanisms.

Conclusions

Fault injection testing has been used for the verification of safety requirements and safety mechanisms in embedded systems for many years. It provides a necessary addition to the more normal positive tests as it allows test coverage of safety mechanisms that cannot otherwise be tested.   The effectiveness of this test method is highly-dependent on the selection of faults to be injected and the workloads that are applied to the system under test during the testing.

The specification of fault injection testing in ISO 26262 could be read as being unnecessarily restrictive. However, ISO 26262 does allow the selection and use of techniques that are not explicitly specified in the tables when appropriate and this allows the use of some of the other forms of fault injection testing described above to ensure the robustness of the systems and coverage of all the safety requirements.

Appendix A – ISO 26262 Fault Injection Testing

Part 4 - Product development at the system level - Hardware-Software Level	8.4.2.2.2 The correct implementation of the technical safety requirements at the hardware-software level shall be demonstrated using feasible test methods given in Table 5.
	Recommended for ASIL A. Highly-recommended for ASIL B, C and D.
	A fault injection test uses special means to introduce faults into the test object during runtime. This can be done within the software via a special test interface or specially prepared hardware. The method is often used to improve the test coverage of the safety requirements, because during normal operation safety mechanisms are not invoked.
	8.4.2.2.5 This requirement applies to ASIL (A), (B), C, and D, in accordance with 4.3: the effectiveness of the hardware fault detection mechanisms' diagnostic coverage at the hardware-software level, with respect to the fault models, shall be demonstrated using feasible test methods given in Table 8.
	Recommended for ASIL A and B. Highly-recommended for ASIL C and D.
	A fault injection test uses special means to introduce faults into the test object during runtime. This can be done within the software via a special test interface or specially prepared hardware. The method is often used to improve the test coverage of the safety requirements, because during normal operation safety mechanisms are not invoked.
Part 4 - Product development at the system level - System Level	8.4.3.2.2 The correct implementation of the functional and technical requirements at the system level shall be demonstrated using feasible test methods given in Table 10.
	Recommended for ASIL A and B. Highly-recommended for ASIL C and D.

	A fault injection test uses special means to introduce faults into the system. This can be done within the system via a special test interface or specially prepared elements or communication devices. The method is often used to improve the test coverage of the safety requirements, because during normal operation safety mechanisms are not invoked.
	8.4.3.2.5 This requirement applies to ASIL (A), (B), C, and D, in accordance with 4.3: the effectiveness of the safety mechanisms' failure coverage at the system level shall be demonstrated using feasible test methods given in Table 13.
	Recommended for ASIL A and B. Highly-recommended for ASIL C and D.
	A fault injection test uses special means to introduce faults into the system. This can be done within the system via a special test interface, specially prepared elements, or communication devices. The method is often used to improve the test coverage of the safety requirements, because during normal operation safety measures are not invoked.
Part 4 - Product development at the system level - Vehicle Level	8.4.4.2.2 The correct implementation of the functional safety requirements at the vehicle level shall be demonstrated using feasible test methods given in Table 15.
	Highly-recommended for ASIL A, B, C and D.
	A fault injection test uses special means to introduce faults into the item. This can be done within the item via a special test interface or specially prepared elements or communication devices. The method is often used to improve the test coverage of the safety requirements, because during normal operation safety mechanisms are not invoked
	8.4.4.2.5 This requirement applies to ASIL (A), (B), C, and D, in accordance with 4.3: the effectiveness of the safety mechanisms' failure coverage at the vehicle level shall be demonstrated using feasible test methods given in Table 18.
	Optional for ASIL A. Recommended for ASIL B. Highly-recommended for ASIL C and D.
	A fault injection test uses special means to introduce faults into the vehicle. This can be done within the vehicle via a special test interface, specially prepared hardware or communication devices. The method is often used to improve the test coverage of the safety requirements, because during normal operation safety measures are not invoked.

Part 5 - Product development at the hardware level	10.4.5 The hardware integration and testing activities shall verify the completeness and correctness of the implementation of the safety mechanisms with respect to the hardware safety requirements. To achieve these objectives, the methods listed in Table 11 shall be considered.
	Recommended for ASIL A and B. Highly-recommended for ASIL C and D.
	Fault injection testing aims at introducing faults in the hardware product and analysing the response. This testing is appropriate whenever a safety mechanism is defined. Model-based fault injection (e.g. fault injection done at the gate-level netlist level) is also applicable, especially when fault injection testing is very difficult to do at the hardware product level. For example, showing the response of safety mechanisms to transient faults inside hardware parts, such as a microcontroller, is very difficult to do with fault insertion at the hardware product level since it would require irradiation tests.
Part 6 - Product development at the software level	9.4.3 The software unit testing methods listed in Table 10 shall be applied to demonstrate that the software units achieve: a) compliance with the software unit design specification (in accordance with Clause 8); b) compliance with the specification of the hardware-software interface (in accordance with ISO 26262-5:2011, 6.4.10); c) the specified functionality; d) confidence in the absence of unintended functionality; e) robustness; and f) sufficient resources to support their functionality.
	Recommended for ASIL A, B and C. Highly-recommended for ASIL D.
	This includes injection of arbitrary faults (e.g. by corrupting values of variables, by introducing code mutations, or by corrupting values of CPU registers).
	10.4.3 The software integration test methods listed in Table 13 shall be applied to demonstrate that both the software components and the embedded software achieve:

	a) compliance with the software architectural design in accordance with Clause 7; b) compliance with the specification of the hardware-software interface in accordance with ISO 26262-4:2011, Clause 7; c) the specified functionality; d) robustness; and e) sufficient resources to support the functionality.
	Recommended for ASIL A and B. Highly-recommended for ASIL C and D.
	This includes injection of arbitrary faults in order to test safety mechanisms (e.g. by corrupting software or hardware components).
	11.4.2 To verify that the embedded software fulfils the software safety requirements, tests shall be conducted in the test environments listed in Table 16. NOTE Test cases that already exist, for example from software integration testing, can be re-used. 11.4.4 The results of the verification of the software safety requirements shall be evaluated with regard to: a) compliance with the expected results; b) coverage of the software safety requirements; and c) pass or fail criteria.