Fault Injection in the Automotive Standard ISO 26262: An Initial Approach

Ludovic Pintard, Jean-Charles Fabre, Karama Kanoun, Michel Leeman, Matthieu Roy

Fault Injection in the Automotive Standard ISO 26262: An Initial Approach

Ludovic Pintard[1,4], Jean-Charles Fabre[1,2], Karama Kanoun[1,3], Michel Leeman[4], and Matthieu Roy[1,3]

[1] CNRS, LAAS, 7 avenue du colonel Roche, BP 54200, F-31031 Toulouse, France
firstName.lastName@laas.fr
[2] Univ de Toulouse, INPT, LAAS, F-31400 Toulouse, France
[3] Univ de Toulouse, LAAS, F-31400 Toulouse, France
[4] VALEO, 2 rue André Boulle, 94046 Créteil cedex, France
fistName.lastName@valeo.com

Abstract. Complexity and criticality of automotive electronic embedded systems is steadily increasing today. A new standard —ISO 26262— recommends methods and techniques, such as fault injection, to improve safety. A first goal is to use fault injection earlier at the design stage, particularly on models providing an appropriate level of abstraction, to identify errors in the handling of safety requirements. A second objective is to use the results of these model-based analyzes to efficiently identify targets and check their implementation by fault injection. Hence, a verification approach, based on fault injection, has to be defined to complement conventional testing methods and analyzes traditionally used in automotive development process. The paper discusses the various steps of this approach, the link between abstraction and implementation, and gives a brief illustration on a real automotive application.

Keywords: fault injection, automotive systems, ISO 26262, development process

Introduction

As safety is a non-negotiable requirement for automotive critical embedded systems, the development process is evolving to assure that they should not lead to severe hazards. To respond to this trend a new standard have been proposed. ISO 26262, published in November 2011, defines the safety aspects of the development of electric and electronic automotive systems. A significant aspect of ISO 26262 is that it recommends fault injection to verify if systems are safe. To this end, this paper explores the integration of fault injection techniques throughout the development process, to perform efficient fault removal activities.

To illustrate our approach described in this paper, we use an electronic automotive component, the *Electronic Steering Column Lock* (ESCL), that has strong

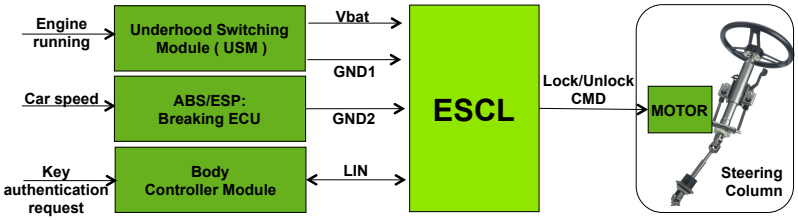

Fig. 1. ESCL Component and its Environment at System Level

safety requirements. Indeed, according to a conventional automotive scale of criticality, the highest Automotive Safety Integrity Level (ASIL D) is allocated to this component.

The ESCL, as shown in Fig. 1, is a component intended to manage the locking/unlocking of the steering column of a vehicle, so that if a thief tries to steal the vehicle, he cannot turn the vehicle wheels. However, this component may endanger the safety of the driver, since a spurious blocking of the steering column when the vehicle is at high speed could obviously threaten passengers safety.

The article is structured as follows. Section 1 describes the problem statement. In Section 2, we discuss the motivation and the benefits of using fault injection at the design stage, particularly on models. We briefly describe conventional fault injection on implementation in Section 3, and conclude on the lessons learnt.

1 Problem Statement

The new ISO 26262 standard highly recommends the use of fault injection techniques throughout the development process to verify safety requirements and safety mechanisms. Requirements of ISO 26262 highlight several targets for fault injection into the V-Cycle represented in Fig. 2.

The possible targets can be classified in two categories, namely (1) during the design steps down to the implementation, and (2) up to the verification and the validation of the integrated system. In the left side of the cycle, targets are *models* or representations of the system before the implementation. The right side of the cycle corresponds to an implementation of the system *components*, their integration and their validation.

Models: The standard recommends fault injection on models of system level and hardware level. There are two goals: *i*) to check that specifications related to the behavior in the presence of faults do not contain any omission or error, and *ii*) to ensure if the system implements appropriate mechanisms to prevent the violation of safety properties.

Components: These are effective targets linked to the verification of a system implementation, from the unit tests, through the integration phase, to the veri-

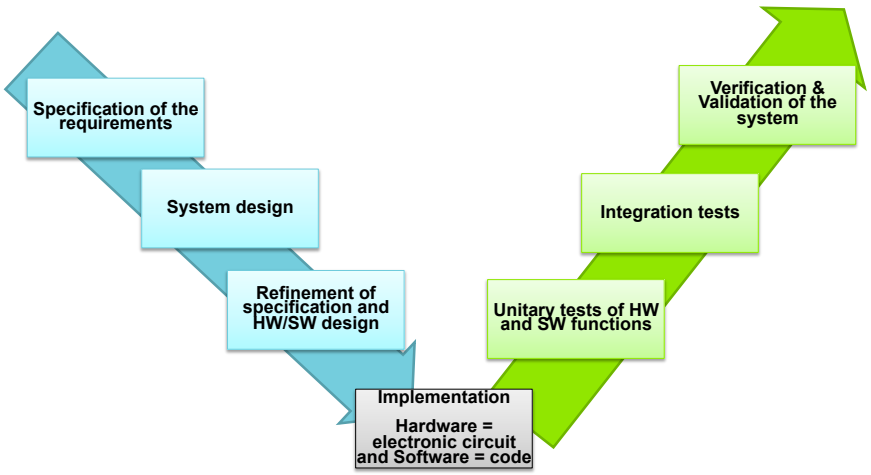

Fig. 2. ISO 26262 V-cycle Development Process.

fication and validation of the complete system. At this stage, we seek to characterize the effectiveness of fault tolerance mechanisms (detection and recovery of errors) that have been implemented to increase safety as well as reliability and availability.

A first challenge is to define fault injection methods throughout the development process of an electronic/electrical system and particularly explore the possible contribution of fault injection at each stage of the process to improve the quality of the design. In a certain sense, we introduce the notion of multi-level fault injection and aim at analyzing possible links between targets, objectives and results at various development stages. This paper reports on our initial approach to tackle this problem.

Concerning integration, our second objective is to analyze how fault injection experiments related to a system can benefit from the results obtained to its components by fault injection. However, the composition of fault injection experiments in a hierarchical way is out of the scope of this paper, but will be highly explored during the project.

Fault injection [1] is a mature technology that has been successfully applied using several techniques on different targets [3–5, 7, 8, 14], that are usually components implementation. However, to the best of our knowledge, fault injection has not been studied throughout a development process at various development or abstraction levels in cooperation with usual testing methods.

2 Fault Injection During System Design

2.1 Fault Injection before Implementation

The high-level specifications of the system are progressively refined to provide detailed specifications of the hardware and of the software before implementation. During this refinement process, the form in which the specifications are

written may evolve. These specifications can be expressed in natural language, or in the form of a well-structured and formalized model (e.g., in an enterprise proprietary language), or in the form of a formal model, based on a standard formal language. Two kinds of requirements are usually distinguished: functional requirements and safety requirements.

At the initial step, the requirements are related to very high-level functions of the system, without addressing the system structure. A functional model may exist or can be built. Faults can be defined only at the same level of abstraction. Fault injection consists in assuming (or simulating) a failure of a basic function, and analyzing the impact of this failure on the other system functions and on the overall system functions. In this case, fault injection constitutes a way *i*) to check the impact of the failure of each basic function on the other function(s), and *ii*) to ensure that the safety requirements are satisfied. Hence, fault injection can be seen as a method very similar to the well-known and widely used approach, usually referred to as Failure Mode and Effect Analysis, FMEA [2], or Failure Mode and Effect and Criticality Analysis, FMECA (depending on whether the criticality is analyzed or not).

The primary benefit of fault injection is the same as for FMECA: the early identification of all critical system failure modes so they can be eliminated or minimized through design modification at the earliest phase in the development process. Another benefit is to identify the parts of the system and functions that require error detection and/or fault-tolerance mechanisms.

As for FMECA, the results of the fault injection analysis become more precise when more details about system functions are available (i.e., when the abstraction level of the system functional description becomes lower).

From the system requirements, a functional model is created. A fault injection "experiment" consists in selecting a failure mode of a function (or a component) and analysing its resultant effects on system operation, taking into account the overall safety requirements. The effects are usually defined with respect to the impact of the failures on the safety properties; they are referred to as "system failure modes". Each fault injection experiment corresponds to a single failure mode of one basic function (or of one component). A function or a component, with several potential failure modes, requires one experiment per failure mode. Indeed, each fault injection experiment corresponds to one line of a FMECA worksheet (or spreadsheet). Examples of information items that can be included in one line (or provided after a fault injection experiment) are: identification of the basic function (or component) analyzed, its potential failure mode addressed, the potential causes of this failure mode, the local effect of the failure mode, the next assembly layer effect, system level effect, the risk level, detection mechanisms to put in place, actions for further investigation.

Even though, the analyses can be performed manually for high-level preliminary design, the support of a modelling formalism and a tool becomes mandatory as soon as detailed information becomes available.

From a modelling point of view, at a high-level, the analyses can be performed based on data flow diagrams and/or state charts, to help the analyst to propagate

errors between the components of the system. When more details are available, languages such as UML (Unified Modelling Language) or AADL (Architecture Analysis Design Language) can be used. Their main advantage is that they have been extended to perform quantitative dependability assessment of critical systems (see e.g., [9, 10]).

Finally, several simulation modelling languages have been used for fault injection. At these levels, the model can be very similar to the real implementation of the system (see e.g., VHDL [6], SystemC [11], Matlab/Simulink [12] or SCADE [13]).

2.2 Illustration on a Case Study

The first objective of our method is to determine a high-level abstraction of our system. Fig. 1 presents the relation between ESCL and its environment. Indeed, a component has dependencies with other components. The dependencies are related to the inputs and outputs, because they link the components. Hence, fault injection at this level consists in propagating component failures and errors through the relationships between blocks.

Then, we have to define the safety requirements this system has to verify. There are two safety requirements, called safety goals according to ISO 26262, that must be ensured:

- SG1 = The ESCL must not lock the steering column when the vehicle speed is greater than 10 km/h. (ASIL D)
- SG2 = If the steering column is locked, the ESCL must prevent to start the engine of the vehicle. (ASIL A)

For example, SG2 will be specified at this level as follow: the ESCL should not send erroneous messages, via the LIN bus, stating that the steering column is unlocked while it is locked.

All safety requirements, at each level, are important because they define a set of invariants that must be satisfied by the system. Would an invariant be violated, a hazard may occur. To satisfy these properties, the design should explicitly exhibit each safety mechanism that deals with a property. Here, the criticality of a mechanism can be defined according to the ASIL level of the safety requirement, and so, the targeted safety mechanisms must be evaluated by fault injection techniques.

Then, the error model has to be defined. At this level, we can identify the hardware architecture of our case study with five components, and the associated five links. There are four electrical links, with four failures modes: *i*) there is no power when it is required, *ii*) there is power while it is not required, *iii*) oversupply, *iv*) and under-supply. The failure modes of the fifth link, a bidirectional LIN bus, are the following: no message transmitted, erroneous message transmitted, corrupted message.

Following the approach described in Section 2.1, the fault injection applied to the functional description of the ESCL enables the identification of critical

blocks and their effect on the system. Considering the link between them, we can identify whether a safety property may be violated. A critical path is the propagation of an error through the link that violates the safety properties.

Let's take the example of a corrupted message from the Body Controller Module ordering to lock the column when the vehicle is at high-speed. This could violate SG1, and the study of the architecture allows to check whether a safety mechanism exists — here, the ABS/ESP switches off the ESCL when the speed is larger than a threshold and hence the ESCL will not lock the column. The system level architecture shown on figure Fig. 1 can be considered as the first modeling level. However, this description is at a too high level to verify the real hardware architecture or the software architecture of the ESCL.

Considering the hardware, a more detailed model should be used to describe the architecture in terms of subcomponents: sensors, micro-controllers, memories, power supply units. This detailed architecture is represented on Fig. 3.

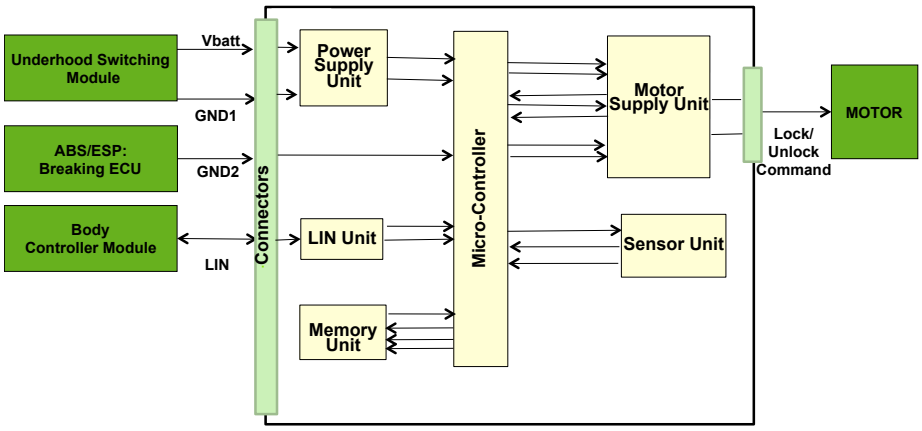

Fig. 3. Architectural Model of ESCL Component.

Considering the software, modules and their dependencies can be modeled through function and procedure calls on a static view with a communication diagram, and then with sequence diagrams for the dynamic representation of the interactions. The robustness of implemented components running on the hardware must be evaluated according to a fault model:

- Hardware errors: errors on the inputs of the micro-controller, errors at the interfaces of software module due to a corruption of the memory, or error in sequences and with timing constraints.
- Software errors: coding faults should be represented by malfunctions on the interface of each software module.

3 Fault Injection during Verification

On the right-hand side of the V-cycle, an implementation of the system is available, in the form of a hardware support system, a set of software components, and later on in the form of a global system in which the software components are integrated on the hardware system. All conventional fault injection methods are applicable.

Our recommendation is to use the results of the analyses carried out during the design phase to guide fault injection experiments in the verification phase. For example, fault injection campaigns will focus on critical components identified earlier. In our case study, we concentrate our analysis on the internal architecture of an ESCL component. The goal could be to activate error detection and error recovery mechanisms according to a fault model and evaluate their robustness. The aim of the experiments will be for example to:

- Check the correct implementation of the system together with the associated error detection and fault-tolerance mechanisms.
- Assess the error detection coverage and error recovery coverage of safety mechanisms.

In addition to the conventional aims of fault injection experiments targeting concrete components, the objective is also to find complementarities with usual testing methods applied in the automotive industrial domain in order to optimize the validation process.

Conclusion

The use of fault injection in the development of safety critical embedded automotive systems is explicitly mentioned in the ISO 26262 development standard. One can easily understand that similar types of techniques are nowadays used during the testing phase of embedded automotive systems. Such techniques include in particular FMECA analyses. However, advocating fault injection at various levels of the development of the system poses several challenges, not only with respect to the ISO 26262 application in the automotive domain, but raises more general scientific challenges. In particular, how to handle the complementarities between FMECA and fault injection at various stages of the development process.

We observed that FMECA is similar to fault injection when targeting models. Indeed, these concepts should mutually enrich each other because they share the same objectives. Yet, in practice, FMECA is often applied to coarse grain models. The short-term objective of this work is to show that the concept of FMECA can be applied to more fine grain structural and behavioral models, reaching finally the implementation. Conventional fault injection is the major technique on implemented components, but one can understand that FMECA and Fault Injection are overlapping concepts as models become more and more detailed.

Conversely, fault injection automatized on a detailed model can produce similar results as those expected with FMECA.

In this paper, we have shown how model-based fault injection could be of interest to identify drawbacks in the handling of safety requirements but also to guide lower layers fault injection experiments, for instance with the identification of key targets for conventional verification by SWIFI.

References

1. A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004.
2. A. Bouti and D. A. Kadi. A state-of-the-art review of fmea/fmeca. *Int. Journal of reliability, quality and safety engineering*, 1(04):515–543, 1994.
3. J. Carreira, H. Madeira, and J. Silva. Xception: A technique for the experimental evaluation of dependability in modern computers. *Software Engineering, IEEE Transactions on*, 24(2):125–136, 1998.
4. D. Cotroneo, A. Lanzaro, R. Natella, and R. Barbosa. Experimental analysis of binary-level software fault injection in complex software. In *2012 Ninth European Dependable Computing Conference*, pages 162–172. IEEE, 2012.
5. M. Hsueh, T. Tsai, and R. Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, 1997.
6. E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson. Fault injection into vhdl models: the mefisto tool. In *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*, pages 66–75. IEEE, 1994.
7. J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo. Using heavy-ion radiation to validate fault-handling mechanisms. *Micro, IEEE*, 14(1):8–23, 1994.
8. P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz. Comparing operating systems using robustness benchmarks. In *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on*, pages 72–79. IEEE, 1997.
9. A.-E. Rugina, K. Kanoun, and M. Kaaniche. The adapt tool: From aadl architectural models to stochastic petri nets through model transformation. In *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, pages 85–90, 2008.
10. A.-E. Rugina, K. Kanoun, and M. Kaaniche. Software dependability modeling using aadl (architecture analysis and design language). *International Journal of Performability Engineering*, 7(4):313, 2011.
11. R. A. Shafik, P. Rosinger, and B. M. Al-Hashimi. Systemc-based minimum intrusive fault injection technique with improved fault representation. In *14th IEEE Intl. On-Line Testing Symposium. IOLTS'08*, pages 99–104. IEEE, 2008.
12. R. Svenningsson. Model-implemented fault injection for robustness assessment, 2011. QC 20111205.
13. J. Vinter, L. Bromander, P. Raistrick, and H. Edler. Fiscade-a fault injection tool for scade models. In *Automotive Electronics, 2007 3rd Institution of Engineering and Technology Conference on*, pages 1–9. IET, 2007.
14. S. Winter, C. Sârbu, B. Murphy, and N. Suri. The impact of fault models on software robustness evaluations. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 51–60. IEEE, 2011.