Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model based Development

Rakesh Rana[1], Miroslaw Staron[1], Christian Berger[1], Jörgen Hansson[2], Martin Nilsson[3] and Fredrik Törner[3]
[1]*Computer Science & Engineering, University of Gothenburg, Gothenburg, Sweden*
[2]*Computer Science & Engineering, Chalmers University of Technology, Gothenburg, Sweden*
[3]*Volvo Car Corporation, Gothenburg, Sweden*

Keywords: Fault Injection, Mutation Testing, ISO 26262, Simulink, Model based Development, Automotive Domain, Safety Critical Software.

Abstract: The rapid growth of software intensive active safety functions in modern cars resulted in adoption of new safety development standards like ISO 26262 by the automotive industry. Hazard analysis, safety assessment and adequate verification and validation methods for software and car electronics require effort but in the long run save lives. We argue that in the face of complex software development set-up with distributed functionality, Model-Based Development (MBD) and safety-criticality of software embedded in modern cars, there is a need for evolving existing methods of MBD and complementing them with methods already used in the development of other systems (Fault Injection and Mutation Testing). Our position is that significant effectiveness and efficiency improvements can be made by applying fault injection techniques combined with mutation testing approach for verification and validation of automotive software at the model level. The improvements include such aspects as identification of safety related defects early in the development process thus providing enough time to remove the defects. The argument is based on our industrial case studies, the studies of ISO 26262 standard and academic experiments with new verification and validation methods applied to models.

1 INTRODUCTION

Nowadays, a typical premium car has up to 70 ECUs which are connected by several system buses to realize over 2000 functions (Broy, 2006). As around 90% of all innovations today are driven by electronics and software, the complexity of car's embedded software is already high and expected to grow further. The growth is fuelled by cars beginning to act more proactively and provide more assistance to its drivers, which requires software to interact with hardware more efficiently and making more decisions automatically (e.g. collision avoidance by braking, brake-by-wire or similar functions). In total with about 100 million lines of code (SLOC), premium segment vehicles carry more software code than in modern fighter jets and airliners (Charette, 2009).

Software for custom functionality in modern cars is usually developed by multiple suppliers although it is largely designed by a single OEM (Original Equipment Manufacturer) like Volvo Cars. The distributed development and use of standards like AUTOSAR aims to facilitate reuse of software and hardware components between different vehicle platforms, OEMs and suppliers (Fennel et al., 2006). However, testing of such systems is more complex and even today testing of software generally accounts for almost 50% of overall development costs (Boehm and Basili, 2001).

ISO-26262 in automotive domain poses stringent requirements for development of safety critical applications and in particular on the testing processes for this software. These requirements are intended to increase the safety of modern cars, although they also increase the cost of modern cars.

Rana R., Staron M., Berger C., Hansson J., Nilsson M. and Törner F..
Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model based Development.
DOI: 10.5220/0004592002510257
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-EA-2013), pages 251-257
ISBN: 978-989-8565-68-6

The position for which we argue in this paper is that *efficient verification and validation of safety functions requires combining Model Based Development (MBD) with fault injection into models with mutation testing*. This position is based on the studies of the ISO 26262 standard (mainly chapter 6 that describes requirements on software development but also chapter 4, which poses requirements on product development (ISO 26262 - 2011, 2011)). It is also based on previous case studies of the impact of late defects on the software development practices in the automotive domain (e.g. (Mellegård et al., 2013))

The requirements from the ISO 26262 standard on using fault injection techniques is challenging since it relates to the development of complete functions rather than components or sub-components of software. The current situation in the automotive sector is that fault injection is used, but it is used at the level of one electronic component (ECU) or one software system and rarely at the function level (Hillenbrand et al., 2010; Schätz, 2010).

The current state of art testing is not enough for detecting safety defects early in the automotive software development process since fault injection is done late in the development (when ECUs are being developed), which usually makes the detection of specification-related defects difficult and costly (Mellegård et al., 2013). As much possible this detection should be done at the model level when the ECUs' functionality is still under design and thus, it is relatively cheap to redesign/reconfigure. The evidence from literature on successful use of fault injection shows that the techniques are indeed efficient in finding dependability problems of hardware and software systems when applied to compute (Hsueh et al., 1997). Finally, to be able to increase the effectiveness of the fault injection strategies and identify whether the faults should be injected at the model, software or ECU level - Mutation testing should be applied to verify the adequacy of test cases and finally how the combination of these approaches when applied at the model level will enhance the detection of safety defects right at the design stage.

In this paper, we provide a roadmap, which shows how to introduce fault injection and mutation testing to modelling of automotive software in order to avoid costly late defects and increase the safety of modern and future cars.

The remaining of the paper is structured as follows: In the next section (2) we provide an overview of software development in automotive domain and associated concepts. This is followed by brief discussion on related work in section 3 and our position is presented and discussed in section 4. Section 5 concludes our work.

2 BACKGROUND

In this section we take a brief overview on the current state of automotive software development process and environment, how safety is important in safety critical applications and overview of theoretical background on fault injection techniques and mutation testing.

2.1 Automotive Software Development & ISO 26262

Various software functions/applications developed within the automotive industry today are classed as safety critical, for example Volvo's City Safety feature consists of components that are safety critical.

Figure 1: Volvo Cars city safety function, image provided by Volvo Car Corporation.

(Broy, 2006) gives examples of functions/areas within automotive domain with recent development which includes crash prevention, crash safety, advanced energy management, adaptable man-machine interface, advanced driver assistance, programmable car, car networking etc., much of these also fall within the safety critical functionality and thus demands high quality and reliability. Also a number of on-going projects are directed towards the goal of self-driving cars.

Software development in automotive sector in general follows the 'V' process, where OEMs take the responsibility of requirement specification, system design, and integration/acceptance testing. This is followed by the supplier, which develops the actual code that runs on ECUs. Although the code is tested at the supplier level (mainly unit testing), the

OEMs are responsible for the final integration, system and acceptance testing to ensure that the given implementation of a software (SW) meets its intended functional and safety goals/demands.

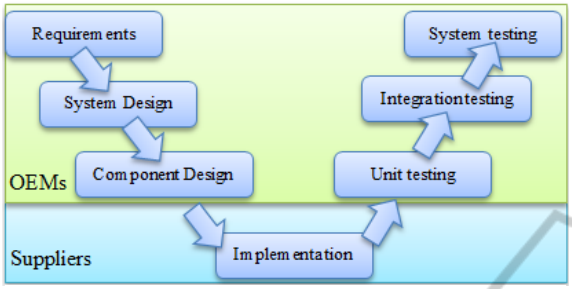

Figure 2: The V-model in the automotive industry with distinction between the OEM and supplier contributions.

In this model of software/product development (see Figure 2) testing is usually concentrated in the late stages of development, which also implies that most of the defects are discovered late in the development process. In a recent study using real defect data from an automotive software project from the industry (Mellegård et al., 2012) showed that late detection of defects is still a relevant problem and challenge yet to overcome. The defect inflow profile presented in this study is reproduced in Figure 3 for reference, which exhibits a clear peak in number of open defects in the late stages of function development/testing.

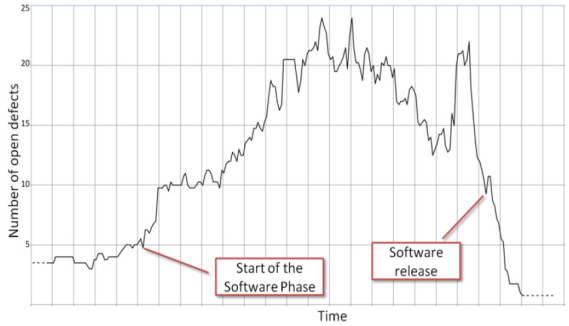

Figure 3: Defect inflow profile for automotive software project, as given in (Mellegård et al., 2012).

Testing the software is an important tool of ensuring correct functionality and reliability of systems but it is also a very resource intensive activity accounting for up to 50% of total software development costs (Jones, 2001) and even more for safety/mission critical software systems. Thus having a good testing strategy is critical for any industry with high software development costs. It has also been shown that most of the defects detected during testing do not depend on actual implementation of code, about 50% of defects detected during testing in the study by (Megen and Meyerhoff, 1995), were found during the test preparation, an activity independent of the executable code. And since automotive sector has already widely adopted MBD for the software development of embedded systems, a high potential exists for using the behavioural modes developed at the early stages of software development for performing some of the V&V (Verification & Validation). Early V&V by helping to detect defects early will potentially save significant amount of cost for the projects and reduce the cycle time.

2.2 ISO 26262

ISO/IEC 26262 is a standard describing safety requirements. It is applied to safety-related systems that include one or more electrical and/or electronic (E/E) systems. The overview of safety case and argumentation is represented in Figure 4, based on (ISO 26262 - 2011, 2011).

Figure 4: Overview of ISO-26262 safety case & argumentation process.

Written specifically for automotive domain/sector, the ISO-26262 standard is adapted for the V-model of product development corresponding to the current practice in the industry. The guidelines are laid out for system design, hardware and software design & development and integration of components to realize the full product. ISO-26262 includes specifications for MBD and provides recommendations for using fault injection techniques for hardware integration and testing, software unit testing, software integration testing, hardware-software integration testing, system

integration testing and vehicle integration testing, for overview on fault injection recommendations in ISO-26262 see (Rana et al., 2013). Although the functional safety standard specifies clearly the recommendations for using fault injection during various stages of testing but it does not recommend anything with respect to using mutation testing. This also reflects the current standard practice within the automotive industry where mutation testing is not widely adopted yet.

2.3 Fault Injection

Fault injection techniques are widely used for experimental dependability evaluation. Although these techniques have been used more widely for assessing the hardware/prototypes, the techniques are now about to be applied at behavioural models of software systems (Svenningsson et al., 2010) - thus enabling early verification of intended functionality as well as enhancing communication between different stakeholders. Fault injection techniques applied at models level offer distinct advantages especially in an industry using MBD for its software development, but use of these techniques at model level in automotive industry is currently at its infancy. Figure 5 shows a mind map of classification of fault injection techniques based on how the technique is implemented; some of the tools which are developed based on given approach are also listed for reference. For a good overview of fault injection techniques readers are referred to (Hsueh et al., 1997; Ziade et al., 2004).

2.4 Mutation Testing

Mutation testing is technique for assessing the adequacy of given test suite. Mutation testing includes injection of systematic, repeatable seeding of faults in large number thus generating number of copies of original software artefacts with artificial fault infestation (called a mutant). Percentage of mutations detected by the given test cases/suite is a metrics (called "mutation adequacy score" (Jia and Harman, 2011)) used for measuring effectiveness of the given test suite. The variants of code (faults) can be introduced by hand or auto-generated using tools like Insure++, Plextest, Certitude, ESPT for C/C++ codes. It has been shown that the use of mutants

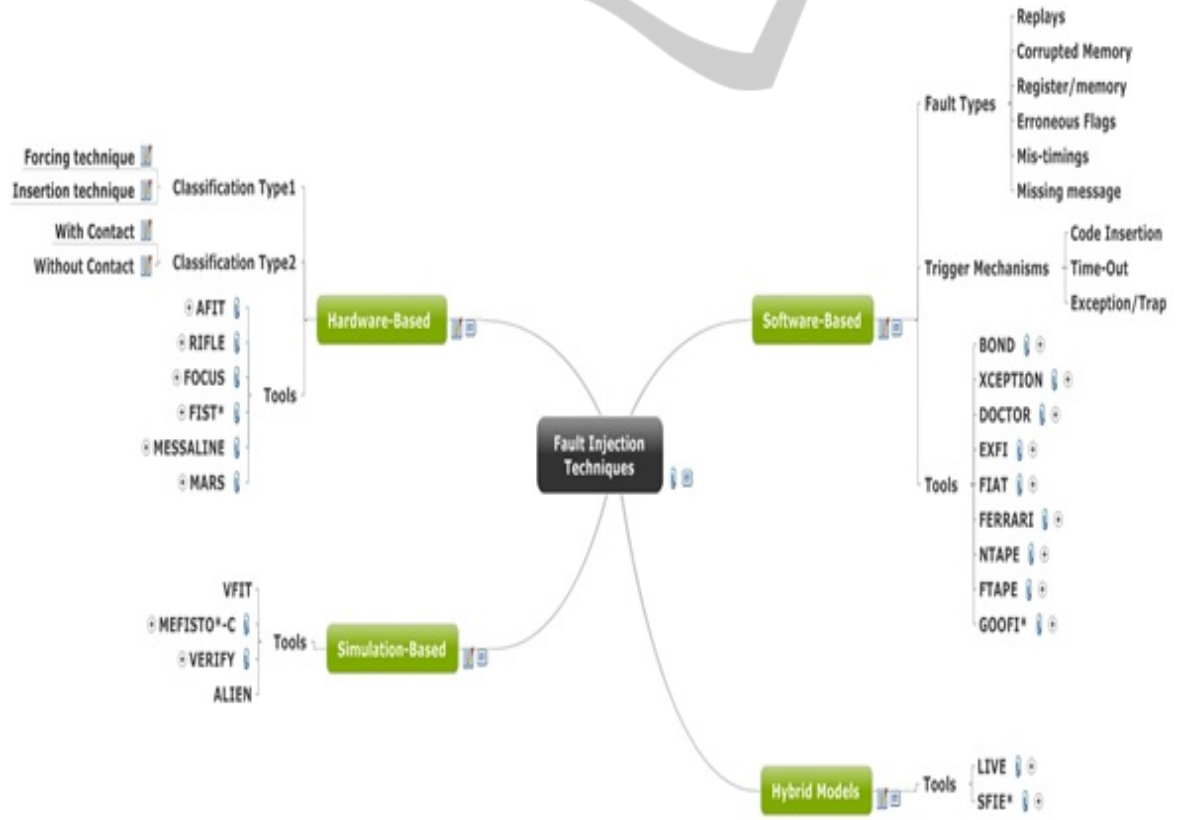

Figure 5: Common classification of fault injection techniques and implementation tools, description available in (Ziade et al., 2004; Hsueh et al., 1997).

yields trustworthy results (Andrews et al., 2005), i.e. mutants do reflect characteristics of real faults.

Mutation theory is based on two fundamental hypotheses namely Competent Programmer Hypothesis and the Coupling Effect, both introduced by (DeMillo et al., 1978). The Competent Programmer hypothesis reflects the assumption that programmers are competent in their job and thus would develop programme close to correct version (although making a number of mistakes) while the Coupling Effect hypothesis means that complex mutants are coupled to simple mutants in such a way that a test data that detects large percent of simple faults is also effective in detecting high percentage of the complex defects" (Offutt, 1992).

3 RELATED WORK

A number of European Union sponsored projects, within the area of embedded software development and safety critical systems have looked at and developed techniques to effectively use fault injection for safe and reliable software development. The examples include the ESACS (Enhanced Safety Assessment for Complex Systems) (Joshi and Heimdahl, 2005) and the ISAAC (Kakade et al., 2010)(Improvement of Safety Activities on Aeronautical Complex systems). These projects have used the SCADE (Safety-Critical Application Development Environment) modelling environment to simulate hardware failure scenarios to identify fault combinations that lead to safety case violations.

A model-implemented fault injection plug-in to SCADE called FISCADE is introduced in (Vinter et al., 2007). The plug-in tool utilizes approach similar to mutation based testing, where it replaces the original model operators by their equivalent fault injection nodes. The derived models are then used to inject the fault during execution and log the results which are analysed later. Dependability evaluation of automotive functions using model based software implemented fault injection techniques have also been studied in (Plummer, 2006).

A generic tool capable of injecting various types of faults on the behavioural/functional Simulink models is also developed and introduced in (Svenningsson et al., 2010). The tool called MODIFI (or MODel-Implemented Fault Injection tool) can be used to inject single or multiple point faults on behavioural models, which can be used to study the effectiveness/properties of fault tolerant system and identify the faults leading to failure by studying the fault propagation properties of the models.

Another work (Brillout et al., 2010) with its root in the European CESAR (Cost-efficient methods and processes for safety relevant embedded systems) project provides a good theoretical overview of how fault and mutation based test coverage can be used for automated test case generation for Simulink models. We provide a practical framework on how fault injection combined with mutation testing within an MDB environment can be used in the industry. And how will this practice enhance the verification and validation of software under development, its functional validation that would generates statistics for the effective argumentation of ISO 26262 compliance.

4 ROAD MAP FOR EARLY DEFECT DETECTION

We contend that fault injection can be effectively used at the model level to verify and validate the attainment or violation of safety goals. We also propose that it should be complemented with mutation testing approach at the model level to provide enough statistical evidence for argumenting the fulfilment of safety goals as per the ISO-26262 safety standard requirements.

A major challenge in successful argumentation of ISO-26262 compliance is to provide statistical evidence that safety goals (SGs) would not be violated during operation and collecting the evidence for this argumentation within reasonable testing efforts.

If we are able to differentiate early between defects that can cause the violation of SGs and those that cannot cause the violation, the amount of testing required will be manageable. With MBD the functional testing could be done using fault injection techniques and this can be complemented with later system testing of the actual code using the mutation testing approach.

The framework on how this could be achieved in practice is as follows:

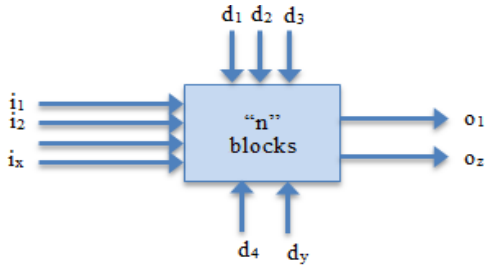

Figure 6: MBD based representation of a general system with inputs, outputs and dependencies.

As illustrated in Figure 6, a given system/function generally have following common features (in context of model based development): firstly it will have x inputs ($i_{1,2...x}$); it would have dependencies to other y components/ functions ($d_{1,2...y}$); it will have z outputs ($o_{1,2...z}$); and it will have a number of sub-units/modules within it that implement the intended functionality, let us assume that this part contains n basic blocks in the modelling environment corresponding to n statements for a hand written code. To verify and validate the correct functionality and ISO-26262 compliance of this generic function using fault and mutation testing approach we can follow the steps as:

a. Assign or define the technical safety requirements (TSRs) corresponding to the functional safety requirements (FSRs) for the given system/function to its z outputs.
b. Use fault injection techniques to inject faults which are similar to commonly occurring defects and other possible fault conditions at the x inputs of the function.
c. Fault scenarios that leads to violation of TSRs/FSRs are identified, statistics are built on what percentage of total faults lead to such failures and fault propagation properties of such cases are studied to build the fault tolerance within the system for given fault conditions.
d. Repeat steps (b) & (c) to test, correct and validate the given system/function for its dependencies on other functions/components.
e. Cause mutations to the n basic blocks of given functional model and asses the detection effectiveness of test suite/cases for possible implementation bugs.
f. Examine the mutants which are not killed by given set of test cases/suits for their effect on FSRs. If a given mutation violates the FSRs then a suitable test case is created to detect/kill such mutants, i.e. detect such bugs in actual code.

By following the above mentioned steps we not only ensure that the given function holds the FSRs and TSRs under faulty inputs, but we can also prevent potential implementation defects and ensure that we have test cases ready to catch such faults that can potentially violate the FSRs/TSRs already at the design (model) level.

It is also worthwhile to note here that steps (a) to (e) can be easily automated using the currently available testing methodologies, which makes the usability and industrial viability much higher that testing frameworks requiring high manual interventions.

Further to make this framework/approach more effective in industrial practice we identify a number of best practices that will have positive impact on detecting defects early in the development process and thus have effective V&V of ISO-26262:

a. The best practice is to build and maintain models corresponding to each abstraction layer of software architecture.
b. The next best practice is to specify and test these models for FSRs and TSR at the appropriate abstraction level.
c. Also identification of different types of defects/faults and at what stage they could be modelled/injected in the behavioural models would ensure that models are tested for these faults at the earliest - leading to models being build that are robust right from the start instead of adding fault tolerance properties in the later stages of development.

5 CONCLUSIONS

The development of software in the automotive domain has widely adopted the paradigm of model based development to allow for easier integration of functionality usually developed by multiple suppliers. By the nature of the domain much of the functionality developed and implemented in cars is safety critical; the criticality that requires observation of stringent quality assessment and adherence to functional safety standards such as ISO 26262.

Development of behavioural models in MBD offers significant opportunity to do functional testing early in the development process. Fault injection and mutation testing approach in combination can be used to effectively verify and validate the functional properties of a software system/function. The approach also provides required statistics for the argumentation of safety standards compliance. In this paper the need for such validation and a framework on how this could be achieved in practice is discussed. The results are a roadmap for further research and tool support to bring this approach into wider industrial adoption.

By detecting defects early and being able to do much of verification and validation of intended functionality, robustness and compliance to safety standards on the models – the quality and reliability of software in automotive domain can be significantly enhanced. Effective approaches and tools support reduce the V&V costs and lead to shorter development times.

ACKNOWLEDGEMENTS

The work has been funded by Vinnova and Volvo Cars jointly under the FFI programme (VISEE, Project No: DIARIENR: 2011-04438).

REFERENCES

Andrews, J. H., Briand, L. C., Labiche, Y., 2005. Is mutation an appropriate tool for testing experiments? [software testing], in: Software Engineering, 2005. ICSE 2005. *Proceedings. 27th International Conference On*. pp. 402–411.

Boehm, B., Basili, V. R., 2001. Defect Reduction Top 10 List. Computer 135–137.

Brillout, A., He, N., Mazzucchi, M., Kroening, D., Purandare, M., Rümmer, P., Weissenbacher, G., 2010. *Mutation-based test case generation for simulink models, in: Formal Methods for Components and Objects*. pp. 208–227.

Broy, M., 2006. Challenges in automotive software engineering, in: *Proceedings of the 28th International Conference on Software Engineering*. pp. 33–42.

Charette, R. N., 2009. This Car Runs on Code. http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

DeMillo, R. A., Lipton, R. J., Sayward, F.G., 1978. Hints on test data selection: Help for the practicing programmer. Computer 11, 34–41.

Fennel, H., Bunzel, S., Heinecke, H., Bielefeld, Jü, rgen, Fü, rst, S., Schnelle, K.-P., Grote, W., Maldener, N., Weber, T., Wohlgemuth, F., others, 2006. Achievements and exploitation of the AUTOSAR development partnership. Convergence 2006, 10.

Hillenbrand, M., Heinz, M., Adler, N., Müller-Glaser, K.D., Matheis, J., Reichmann, C., 2010. ISO/DIS 26262 in the context of electric and electronic architecture modeling, in: Architecting Critical Systems. *Springer*, pp. 179–192.

Hsueh, M. C., Tsai, T. K., Iyer, R. K., 1997. Fault injection techniques and tools. Computer 30, 75–82.

ISO 26262 - 2011, 2011. Road vehicles -- Functional safety -- Part 1-10.

Jia, Y., Harman, M., 2011. An analysis and survey of the development of mutation testing. Softw. Eng. *IEEE Trans*. 37, 649–678.

Jones, E. L., 2001. Integrating testing into the curriculum—arsenic in small doses, in: *ACM SIGCSE* Bulletin. pp. 337–341.

Joshi, A., Heimdahl, M. P. E., 2005. Model-based safety analysis of simulink models using SCADE design verifier.

Kakade, R., Murugesan, M., Perugu, B., Nair, M., 2010. Model-Based Development of Automotive Electronic Climate Control Software. Model. Found. Appl. 144–155.

Megen, R., Meyerhoff, D. B., 1995. Costs and benefits of early defect detection: experiences from developing client server and host applications. Softw. Qual. J. 4, 247–256.

Mellegård, N., Staron, M., Tö, rner, F., 2012. A light-weight defect classification scheme for embedded automotive software and its initial evaluation.

Mellegård, N., Staron, M., Törner, F., 2013. A Light-Weight Defect Classification Scheme for Embedded Automotive Software Development.

Offutt, A. J., 1992. Investigations of the software testing coupling effect. Acm Trans. Softw. Eng. Methodol. Tosem 1, 5–20.

Plummer, A., 2006. Model-in-the-loop testing. Proc. Inst. Mech. Eng. Part J. Syst. Control Eng. 220, 183–199.

Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M., Törner, F., 2013. Improving Fault Injection in Automotive Model Based Development using Fault Bypass Modeling. Submitted To: *2nd Workshop on Software-Based Methods for Robust Embedded Systems*, Informatik 2013, Koblenz, Germany.

Schätz, B., 2010. Certification of Embedded Software–Impact of ISO DIS 26262 in the Automotive Domain, in: Leveraging Applications of Formal Methods, Verification, and Validation. *Springer*, pp. 3–3.

Svenningsson, R., Vinter, J., Eriksson, H., Törngren, M., 2010. MODIFI: a MODel-implemented fault injection tool. Comput. Saf. Reliab. Secur. 210–222.

Vinter, J., Bromander, L., Raistrick, P., Edler, H., n.d. FISCADE - A Fault Injection Tool for SCADE Models, in: Automotive Electronics, 2007 *3rd Institution of Engineering and Technology Conference On*. pp. 1–9.

Ziade, H., Ayoubi, R. A., Velazco, R., others, 2004. A survey on fault injection techniques. *Int Arab J Inf Technol 1*, 171–186.