Towards Formal Fault Injection for Safety Assessment of Automated Systems *

Ashfaq Farooqui, Behrooz Sangchoolie
Dependable Transport Systems, RISE Research Institutes of Sweden, Borås, Sweden
{ashfaq.farooqui, behrooz.sangchoolie}@ri.se

Reasoning about safety, security, and other dependability attributes of autonomous systems is a challenge that needs to be addressed before the adoption of such systems in day-to-day life. *Formal methods* is a class of methods that mathematically reason about a system's behavior. Thus, a correctness proof is sufficient to conclude the system's dependability. However, these methods are usually applied to abstract models of the system, which might not fully represent the actual system. *Fault injection*, on the other hand, is a testing method to evaluate the dependability of systems. However, the amount of testing required to evaluate the system is rather large and often a problem. This vision paper introduces *formal fault injection*, a fusion of these two techniques throughout the development lifecycle to enhance the dependability of autonomous systems. We advocate for a more cohesive approach by identifying five areas of mutual support between formal methods and fault injection. By forging stronger ties between the two fields, we pave the way for developing safe and dependable autonomous systems. This paper delves into the integration's potential and outlines future research avenues, addressing open challenges along the way.

1 Introduction

Safety- and security-critical systems continue to be integrated into our daily lives. Ensuring the safety and security is a multi-disciplinary challenge, where design, development, and evaluation play a crucial role. Thus a strong emphasis on updating current engineering practices to create an end-to-end verification and validation process that integrates all safety and security concerns into a unified approach is key to adopt these systems [23]. Already, *formal methods* and *fault injection* are used in different parts of the development lifecycle to ensure the system is safe and dependable.

Formal methods refers to mathematically rigorous techniques for specifying and verifying software and hardware systems. To many researchers, the necessity of formal methods is now a given [41]. However, this has yet to be the case from an industrial perspective. Several reasons have been suggested for this situation, including a lack of accessible tools, high costs, incompatibility with existing development techniques, and the fact that these methods require a certain level of mathematical sophistication [21, 22, 7].

Fault injection(FI), on the other hand, is an established method used for the measurement, test, and assessment of dependable computer systems in extreme stress or faulty conditions. Functional safety standards such as IEC 61508 [9] and ISO 26262 [38] recommend the use of FI to prove that malfunctions in electrical and/or electronic systems will not lead to violations of safety requirements. In comparison to

*This work was partly supported by the VALU3S project, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey. This work has also been partly financed by the CyReV project, which is funded by the VINNOVA FFI program – the Swedish Governmental Agency for Innovation Systems (Diary number: 2019-03071).

M. Farrell, M. Luckcuck, M. Schwammberger, and
M. Gleirscher (Eds): Fifth International Workshop on
Formal Methods for Autonomous Systems (FMAS 2023)
EPTCS 395, 2023, pp. 153–161, doi:10.4204/EPTCS.395.11

formal methods, FI could be done after the complete system is built. This way, FI could be used to study the impact of a fault in one system and its propagation and impact in the complete end to end system. FI is also used to evaluate security properties of computer systems by means of *attack injection*. Avizienis et al. [4], consider an attack to be a special type of fault which is human made, deliberate and malicious, affecting hardware or software from external system boundaries and occurring during the operational phase.

Formal methods for safety analysis and fault injection are complementary techniques, and the choice of approach depends on the specific system being evaluated and the goals of the evaluation. Each of these methods has its benefits and shortcomings. While formal methods are commonly employed in the early design phase, fault injection testing is performed towards the later stages of development, where the system or its simulation exists. Unfortunately, the knowledge gained from the formal methods at the design phase is rarely reused in other development lifecycle phases when conducting fault injection experiments. Conversely, feedback from fault injection analysis is rarely used to improve the formal design of the system. The underlying problem is the need for common semantics and knowledge sharing across the different communities. It is clear that to deal with modern autonomous systems, formal methods and fault injection will have to be integrated in a smart way to be able to specify, verify, and validate systems, and be understandable by people without a background in formal methods. This last point is essential for autonomous systems, where they must be certified before they can be used.

By looking at the current state of applied research within both fields this paper introduces *formal fault injection* to help develop dependable autonomous systems.

2 Integrating Formal Methods and Fault Injection

The following section highlights the different research directions leveraging the existing state-of-the-art. Additionally, we highlight the new possibilities that will open up as a consequence of this integration and our research plans.

1. **Design level fault analysis:** Some studies focus on analyzing fault impacts during the design phase, using formal methods. Automating Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) for system safety analysis [6, 37, 30], are well-established for within the formal community. Other studies concentrate on developing control strategies to safeguard systems against cyber-security attacks [39, 17, 27]. These approaches primarily target system design and are implemented during early development phases. Despite the industry standards of FMEA and FTA techniques, integrating formal methods into the development cycle has encountered limited adoption, mainly due to the scarcity of practical tools. Furthermore, these methods often face computational challenges, restricting their applicability to only a small portion of the system. Consequently, they typically address a fixed set of faults. It is valuable to explore how insights from the fault injection phase could be incorporated at a more generic abstraction level to streamline analysis process early in the design phase.
2. **Learning the behavior of the faulty system via model learning/learning-based testing:** Model learning [13, 42, 26] seeks to devise techniques for acquiring discrete formal models of systems by observing or interacting with them. These techniques engage in iterative testing of the system, or its simulation, to progressively learn the behavior. Model learning techniques are often applied in combination with other tools and methodologies: model checking [8] for model verification, testing methods [26] to rigorously assess system behavior, and supervisory synthesis [13] to derive supervisory controllers for system control.

we envision integrating fault analysis into the model learning phase. This analysis approach offers insights into a system's fault-handling capabilities. The resultant model encompasses both nominal and faulty behaviors, enabling offline safety analysis. Moreover, these models hold the potential to serve as authoritative proof of system safety, offering a resource for regulatory authorities.

Existing model learning tools, already applied in select industrial contexts [14, 19], provide interfaces external systems. However, an investigation is warranted to devise methods for introducing fault models into these tools and subsequently evaluating their utility. An overarching challenge lies in the scale of the resulting models. Nominal models themselves often attain considerable complexity, leading to challenges associated with state-space explosion. Augmenting these models with fault scenarios is sure to encounter state-space limitations, even for relatively modest systems.

3. **Using formal models for reducing the fault space:** Executing exhaustive fault injection campaigns is not practical. In most cases, such an approach would result in executions that do not contribute significantly to safety analysis. The challenge lies in identifying the optimal set of test instances that effectively analyze a system's dependability. Numerous testing methods have been proposed to address this challenge, including probabilistic approaches [20], coverage-based techniques [10], and heuristic as well as machine learning-based methods [29, 28, 35, 36]. However, most of these methods rely on some level of prior knowledge about the target system. The availability of such knowledge poses limitations in practical applications [28, 20].

   Yet, when formal specifications exist for a specific system, they inherently contain valuable information that can be leveraged. These specifications offer insights into critical faults and their configurations, which are most likely to lead to system failures. This knowledge about fault configurations proves invaluable when designing fault campaigns. Unfortunately, such utilization is often overlooked.

   To address this gap, we propose an investigation into the development of common semantics that can harmonize formal specifications and fault injection techniques. Additionally, we recommend the creation of tools to facilitate the seamless integration of results from both methods. This integration holds the potential to enhance the effectiveness of fault analysis while leveraging the rich information contained within formal specifications.

4. **Falsification for fault analysis:** Falsification methods typically come into play once the system has been implemented, typically in the later stages of the development lifecycle. Since falsification and fault injection share similar approaches and setups, integrating both these methods represents one of the most straightforward way towards a formal fault injection analysis. Both these methods operate with limited knowledge of the system under test. Falsification primarily focuses on testing the input space, while fault injection adopts a broader perspective by also assuming the presence of fault(s) within the system.

   Several generalized methods and tools have proven effective in the formal community for these purposes. For instance, tools like Scenic and VerifyAI [15, 12] take a probabilistic approach to generate scenarios intelligently and test the system. Tools like Breach [11] and HyConf [1] interface with MATLAB/Simulink models for falsification. In the realm of fault injection, AV-fuzzer [24] aligns closely with falsification approaches.

   While most falsification approaches aim to find input values that lead to violations of the system's specifications, they typically do not distinguish between valid and faulty inputs. To assess a system's safety, falsification engines can be employed to identify boundaries within the state-space.

Subsequently, fault injection analysis focuses on these boundary values and faulty inputs to analyze the impact of faults. In this context, exploring techniques to augment the falsification engine with both nominal and faulty behavior represents a promising avenue for further investigation.

5. **A formal specification language for fault injection:** To bridge the envisioned integration of formal methods and fault injection into practical application, a critical missing element is a well-defined *formal specification language* that can act as an interface between the two domains. Therefore, it becomes imperative to delve into the realm of formal specification languages from both theoretical and practical standpoints. A precedent is set by Bessayah et al. [5], who successfully demonstrated the use of Hoare Logic [18] as a specification language for implementing fault injection in communication systems. However, there have been limited efforts to evaluate the suitability of such a language for autonomous systems. Hence, we propose a comprehensive exploration of available formal specification languages to assess their compatibility with fault injection methodologies and to pinpoint areas of research inquiry apart from studies to develop such a language.

3 Formal Fault Injection

In the past few years, a paradigm called "shift-to-left" has inspired researchers to go towards simulation-based and model-based verification and validation of automated systems. The rise of simulation-based development is a key reason we believe this to be the right time to start investigating the integration of formal methods and fault injection-based testing. Notably, simulation-based methodologies have firmly established themselves in both the formal methods and fault injection communities. This shared foundation provides a common ground for the implementation of the proposed integration methods.

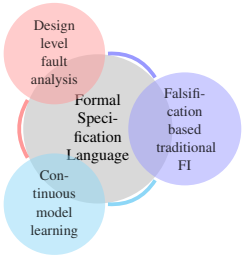

Figure 1: Conceptual overview of the proposed methodology.

Figure 2: Generalized overview of the development lifecycle.

Figure 1 provides a birds-eye view of the proposed integration. Several methodologies exist in literature and practice that define the development lifecycle, such as V-method [16], Waterfall [33], and Agile [32], to name a few. Most of these methodologies include the three phases: design, implementation, and testing phases, in an iterative manner and are depicted in Figure 2. This cycle of development is valid at various abstraction levels of the product lifecycle from the initial conceptual design, feature development, simulation and the final product development. During each phase, a particular set of methods as discussed in Section 2 can be mapped to the phases and are depicted using the similarly colored bubbles in Figure 1. These different phases are never isolated and are continuously updated with feedback from one another. The formal specification language makes it possible for this feedback to be easily translatable between the different phases.

It is crucial to recognize that there is no universal solution applicable to all scenarios. Therefore, the proposal does not revolve around creating a singular specification language and its corresponding

toolkit. Instead, it presents a high-level methodology for evaluating and constructing dependable systems. It acknowledges that specific system characteristics may demand different formalisms. Hence, the idea is to establish a collection of formal specification languages, each supported by its toolset, to facilitate this methodology. Furthermore, these languages, tools, and application approaches can vary across industries, necessitating a multifaceted strategy to address existing limitations and explore new possibilities. Above all, the aim is to establish a consistent and reproducible framework for assessing system dependability.

The assertion of a system's dependability must always be substantiated by the possibility of reproducing the results of all conducted tests. By formally specifying the entire injection methodology, it becomes possible to perform analysis, and potential replication or extension of the results by interested parties. The overarching vision of this work is to enable a standardized interface for all stakeholders involved in ensuring system safety. For instance, developers and companies can employ formal proofs to demonstrate a product's dependability, governmental certification agencies can utilize available data to enhance certification processes, and third-party auditors can scrutinize systems from security and safety perspectives using existing information. Aligning different phases of the development lifecycle with a common language paves the way for formalizing safety evaluations.

Furthermore, ongoing national and international efforts within the autonomous driving domain aim to define and develop a safety assurance framework [40, 31, 34] for verification and validation of autonomous systems. These methods aim to develop a database of testing scenarios to validate a system. Therefore, in addition to developing tools and techniques, we propose investigating the feasibility of recommending formal fault injection as a best practice through responsible standardization organizations.

4 Insights from early experiments

In this section we share our initial experiences and insights from applying formal techniques in simulation based fault injection. Although these experiences are common within the formal community, we believe they offer valuable insights to those interested in bridging the formal and fault injection domains.

4.1 The case study

Maleki and Sangchoolie [25] investigated the effects of faults on Advanced Driver Assistance Systems using the Simulation of Urban Mobility (SUMO) [2]. We use this work as a basis to study the integration of formal techniques and fault injection. The scenario used in that work [25] revolves around a three-lane road, spanning 750 meters. Two vehicles, a leader and a follower, navigate this road. The overarching requirements mandate that these vehicles not collide, successfully traverse the road, and maintain a speed not exceeding 36 m/s–the maximum permissible speed. Furthermore, these requirements should endure even when additional vehicles are introduced to the scenario, thus preserving safety and functionality amidst traffic.

To enhance the above with formal methods, we explore the following approaches.

- *Utilizing SAT Techniques for Vehicle Controller Modeling:* This approach involves modeling the vehicle models from SUMO into a SAT solver. By doing so, we enable the solver to identify potential (faulty) parameters that could lead to the violation of requirements. This approach effectively narrows down the fault space that needs to be tested. Subsequently, these identified inputs can be verified within SUMO to assess their impact on the system's behavior.

- *Applying Model Learning for Faulty Model Generation:* Here, we connect SUMO to a model learning tool controlled over TCP/IP and allow the system to learn an abstracted model that closely describes the behavior of the simulation.

4.2 Insights

Below we provide some of our insights from early experiments on the previously mentioned case study.

1. **Finding suitable abstractions:** One of the most significant takeaways from our study was the challenge of achieving a suitable abstraction level that aligns with the use case. As the system was implemented within the SUMO framework, we encountered limitations in deriving valuable insights from higher levels of abstraction. Operating at elevated levels of abstraction led to limited applicability of formal analysis to the fault injection process. Specifically, our attempts at model learning within the SUMO context resulted in excessively intricate models, often never terminating.

   For instance, to learn a faulty model, employing a model learning approach akin to Angluin's L^* algorithm [3] necessitated the establishment of a system *alphabet*. In this context, the alphabet signifies the set of symbols providing context to the system states and transitions. Ensuring that this alphabet adeptly captures both faulty and non-faulty state transitions demanded creative thinking and held a pivotal role in shaping the effectiveness of the acquired model.

2. **The state-space explosion problem:** Both the SAT-based and model learning approaches encountered state space explosion. This challenge emerged due to the logical abstraction and simulation granularity, both significantly influencing the efficiency and effectiveness of the formal analysis. A key disparity between fault injection analysis and formal analysis became evident. While fault injection focuses solely on inputs and their corresponding outputs, formal models encompass all inputs and potential outputs, unveiling the system's comprehensive behavior. Consequently, what initially appears as the challenge of examining a finite set of input parameters in fault injection transforms into the state space issue within formal methods. Striking a balance between these two extremes emerges as the sought-after equilibrium.

3. **Non-deterministic behavior:** SUMO is a deterministic simulator. While constructing the formal model, however, this determinism is lost when augmented with faulty parameters. Additionally, the fault injection community employs randomness within faulty inputs to assess system dependability. This introduces complexity in constructing models that can effectively accommodate such randomness, necessitating techniques like abstraction or alternative formalisms. The non-deterministic aspect might not be immediately evident, potentially yielding unfavorable outcomes if appropriate analysis methods are not selected.

5 Conclusions

In summary, this paper introduces the concept of formal fault injection, an approach that synergizes formal methods and fault injection techniques to enhance the dependability and safety of autonomous systems. By harmonizing development and evaluation phases, this approach facilitates cross-phase knowledge sharing, fostering a unified approach to ensuring dependability attributes. This alignment not only benefits certification bodies and developers but also strengthens the foundation for proving system reliability.

References

[1] Arend Aerts, Mohammad Reza Mousavi & Michel Reniers (2015): *A Tool Prototype for Model-Based Testing of Cyber-Physical Systems*. In Martin Leucker, Camilo Rueda & Frank D. Valencia, editors: *Theoretical Aspects of Computing - ICTAC 2015*, Lecture Notes in Computer Science, Springer International Publishing, Cham, p. 563–572, doi:10.1007/978-3-319-25150-9_32.

[2] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner & Evamarie Wiessner: *Microscopic Traffic Simulation using SUMO*. doi:10.1109/ITSC.2018.8569938.

[3] Dana Angluin (1987): *Learning regular sets from queries and counterexamples*. *Information and Computation* 75(2), pp. 87 – 106, doi:10.1016/0890-5401(87)90052-6.

[4] A. Avizienis, J.-C. Laprie, B. Randell & C. Landwehr (2004): *Basic concepts and taxonomy of dependable and secure computing*. *IEEE Transactions on Dependable and Secure Computing* 1(1), pp. 11–33, doi:10.1109/TDSC.2004.2.

[5] Fayçal Bessayah, Ana Cavalli & Eliane Martins (2009): *A formal approach for specification and verification of fault injection process*. In: *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, Association for Computing Machinery, New York, NY, USA, p. 883–890. Available at `https://doi.org/10.1145/1655925.1656086`.

[6] Marco Bozzano, Alessandro Cimatti, Cristian Mattarei & Stefano Tonetta (2014): *Formal Safety Assessment via Contract-Based Design*. In Franck Cassez & Jean-François Raskin, editors: *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, Springer International Publishing, Cham, p. 81–97, doi:10.1007/978-3-319-11936-6_7.

[7] Holloway C. Michael (1997): *Why Engineers Should Consider Formal Methods*. Technical Report, NASA Langley Technical Report Server, doi:10.1109/DASC.1997.635021.

[8] Sofia Cassel, Falk Howar, Bengt Jonsson & Bernhard Steffen (2016): *Active learning for extended finite state machines*. *Formal Aspects of Computing* 28(2), pp. 233–263, doi:10.1007/s00165-016-0355-5.

[9] International Electrotechnical Commission (2010): *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems*. Available at `https://webstore.iec.ch/publication/5515`.

[10] M. Cukier, D. Powell & J. Ariat (1999): *Coverage estimation methods for stratified fault-injection*. *IEEE Transactions on Computers* 48(7), p. 707–723, doi:10.1109/12.780878.

[11] Alexandre Donzé (2010): *Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems*. In Tayssir Touili, Byron Cook & Paul Jackson, editors: *Computer Aided Verification*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, p. 167–170, doi:10.1007/978-3-642-14295-6_17.

[12] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte & Sanjit A. Seshia (2019): *VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems*, p. 432–442. *Lecture Notes in Computer Science* 11561, Springer International Publishing, Cham, doi:10.1007/978-3-030-25540-4_25. Available at `http://link.springer.com/10.1007/978-3-030-25540-4_25`.

[13] Ashfaq Farooqui (2021): *On supervisor synthesis via active automata learning*. Chalmers Tekniska Hogskola (Sweden). Available at `https://research.chalmers.se/en/publication/523934`.

[14] Ashfaq Farooqui, Fredrik Hagebring & Martin Fabian (2021): *MIDES: A Tool for Supervisor Synthesis via Active Learning*. In: *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 792–797, doi:10.1109/CASE49439.2021.9551435.

[15] Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli & Sanjit A. Seshia (2020): *Scenic: A Language for Scenario Specification and Data Generation* (arXiv:2010.06580). doi:10.48550/arXiv.2010.06580. Available at `http://arxiv.org/abs/2010.06580`. ArXiv:2010.06580 [cs].

[16] Iris Graessler & Julian Hentze (2020): *The new V-Model of VDI 2206 and its validation*. at - Automatisierungstechnik 68(5), p. 312–324, doi:10.1515/auto-2020-0015. Available at https://www.degruyter.com/document/doi/10.1515/auto-2020-0015/html.

[17] Christoforos N. Hadjicostis, Stéphane Lafortune, Feng Lin & Rong Su (2022): *Cybersecurity and Supervisory Control: A Tutorial on Robust State Estimation, Attack Synthesis, and Resilient Control*. In: *2022 IEEE 61st Conference on Decision and Control (CDC)*, p. 3020–3040, doi:10.1109/CDC51059.2022.9992966.

[18] C A R Hoare (1969): *An axiomatic basis for computer programming* 12(10). doi:10.1145/363235.363259.

[19] Malte Isberner, Falk Howar & Bernhard Steffen (2015): *The open-source LearnLib: A Framework for Active Automata Learning*. In: *International Conference on Computer Aided Verification*, Springer International Publishing, Cham, pp. 487–495, doi:10.1007/978-3-319-21690-4_32.

[20] Saurabh Jha, Subho Banerjee, Timothy Tsai, Siva K. S. Hari, Michael B. Sullivan, Zbigniew T. Kalbarczyk, Stephen W. Keckler & Ravishankar K. Iyer (2019): *ML-Based Fault Injection for Autonomous Vehicles: A Case for Bayesian Fault Injection*. In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, p. 112–124, doi:10.1109/DSN.2019.00025.

[21] John C. Knight (1998): *Challenges in the Utilization of Formal Methods*. In Anders P. Ravn & Hans Rischel, editors: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 1–17, doi:10.1007/BFb0055331.

[22] John C. Knight, Colleen L. DeJong, Matthew S. Gibble & Luís G. Nakano (1997): *Why Are Formal Methods Not Used More Widely?* In: *Fourth NASA Formal Methods Workshop*, pp. 1–12.

[23] Philip Koopman & Michael Wagner (2017): *Autonomous vehicle safety: An interdisciplinary challenge*. *IEEE Intelligent Transportation Systems Magazine* 9(1), pp. 90–96, doi:10.1109/MITS.2016.2583491.

[24] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk & Ravishankar Iyer (2020): *AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems*. In: *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, Coimbra, Portugal, p. 25–36, doi:10.1109/ISSRE5003.2020.00012. Available at https://ieeexplore.ieee.org/document/9251068/.

[25] Mehdi Maleki & Behrooz Sangchoolie (2021): *SUFI: A Simulation-based Fault Injection Tool for Safety Evaluation of Advanced Driver Assistance Systems Modelled in SUMO*. In: *2021 17th European Dependable Computing Conference (EDCC)*, pp. 45–52, doi:10.1109/EDCC53658.2021.00014.

[26] Karl Meinke & Muddassar A Sindhu (2013): *LBTest: a learning-based testing tool for reactive systems*. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, IEEE, pp. 447–454, doi:10.1109/ICST.2013.62.

[27] Rômulo Meira-Goes, Eunsuk Kang, Raymond H. Kwong & Stéphane Lafortune (2020): *Synthesis of sensor deception attacks at the supervisory layer of Cyber–Physical Systems*. *Automatica* 121, p. 109172, doi:10.1016/j.automatica.2020.109172. Available at https://linkinghub.elsevier.com/retrieve/pii/S0005109820303708.

[28] Mehrdad Moradi, Bentley Oakes & Joachim Denil: *Machine Learning-assisted Fault Injection*. Available at https://hal.laas.fr/hal-02931709v1.

[29] Mehrdad Moradi, Bentley James Oakes, Mustafa Saraoglu, Andrey Morozov, Klaus Janschek & Joachim Denil (2020): *Exploring Fault Parameter Space Using Reinforcement Learning-based Fault Injection*. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, p. 102–109, doi:10.1109/DSN-W50199.2020.00028.

[30] Frank Ortmeier & Gerhard Schellhorn (2007): *Formal Fault Tree Analysis - Practical Experiences*. *Electronic Notes in Theoretical Computer Science* 185, pp. 139–151, doi:10.1016/j.entcs.2007.05.034. Available at https://www.sciencedirect.com/science/article/pii/S1571066107004549. Proceedings of the 6th International Workshop on Automated Verification of Critical Systems (AVoCS 2006).

[31] PEGASUS: *PEGASUS project*. Available at https://www.pegasusprojekt.de/en/home.

[32] D.J. Reifer (2002): *How good are agile methods?* *IEEE Software* 19(4), p. 16–18, doi:10.1109/MS.2002.1020280.

[33] Dr Winston W Rovce: *MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS*. Available at `https://dl.acm.org/doi/10.5555/41765.41801`.

[34] SAKURA: *SAKURA project*. Available at `https://www.sakura-prj.go.jp/project_info/`.

[35] Behrooz Sangchoolie, Karthik Pattabiraman & Johan Karlsson (2022): *An Empirical Study of the Impact of Single and Multiple Bit-Flip Errors in Programs*. *IEEE Transactions on Dependable and Secure Computing* 19(3), pp. 1988–2006, doi:10.1109/TDSC.2020.3043023.

[36] Ali Sedaghatbaf, Mehrdad Moradi, Jaafar Almasizadeh, Behrooz Sangchoolie, Bert Van Acker & Joachim Denil (2022): *DELFASE: A Deep Learning Method for Fault Space Exploration*. In: *2022 18th European Dependable Computing Conference (EDCC)*, pp. 57–64, doi:10.1109/EDCC57035.2022.00020.

[37] Yuvaraj Selvaraj, Zhennan Fei & Martin Fabian (2020): *Supervisory Control Theory in System Safety Analysis*. In António Casimiro, Frank Ortmeier, Erwin Schoitsch, Friedemann Bitsch & Pedro Ferreira, editors: *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops*, Lecture Notes in Computer Science, Springer International Publishing, Cham, p. 9–22, doi:10.1007/978-3-030-55583-2_1.

[38] International Organization for Standardization (2018): *ISO 26262: Road vehicles — Functional safety*. Available at `https://www.iso.org/standard/68383.html`.

[39] Rong Su (2018): *Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations*. *Automatica* 94, p. 35–44, doi:10.1016/j.automatica.2018.04.006. Available at `https://www.sciencedirect.com/science/article/pii/S0005109818301912`.

[40] SUNRISE: *SUNRISE project*. Available at `https://sunrise-europe.eu/`.

[41] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui & John Fitzgerald (2009): *Formal methods: Practice and experience*. *ACM Computing Surveys* 41(4), p. 1–36, doi:10.1145/1592434.1592436. Available at `https://dl.acm.org/doi/10.1145/1592434.1592436`.

[42] H. Zhang, L. Feng & Z. Li (2018): *A Learning-Based Synthesis Approach to the Supremal Nonblocking Supervisor of Discrete-Event Systems*. *IEEE Trans. on Automatic Control* 63(10), pp. 3345–3360, doi:10.1109/TAC.2018.2793662.