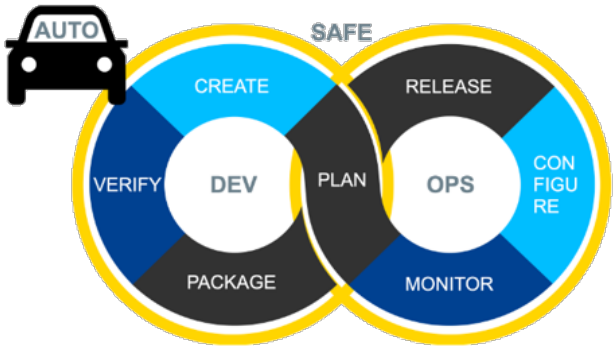


MANNHEIM - AutoDevSafeOps

Schlussbericht

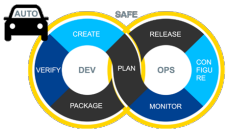
5. November 2025

Projektnummer	16IS22087
Projektkürzel	MANNHEIM-AutoDevSafeOps
Projektstart	2022/10/01
Projektdauer	36 Monate
Förderprogramm	BMFTR - MANNHEIM - Thema C : Methoden und Werkzeuge für die Softwareentwicklung in automobilen „Systems of Systems“
Verantwortliche Organisation	TTTech Auto
Editor	Mohammed Abuteir
Klassifikation	Öffentlich
Revision	1.0

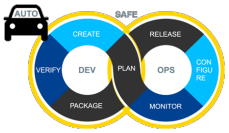
GEFÖRDERT DURCH

Bundesministerium
für Forschung, Technologie
und Raumfahrt

Dieses Projekt wurde durch das Bundesministerium für Forschung, Technologie und Raumfahrt (BMFTR) unter dem Förderkennzeichen 16IS22087 gefördert.



Autoren:
Karsten Klöss (ASV)
Rohit Bohara (ASV)
Andrew Koerner (DLR)
Ingo Stierand (DLR)
Patrick Uven (DLR)
Lukas Westhofen (DLR)
Georg Hake (DLR)
Achim Rettberg (HSHL)
Fatima Idrees (HSHL)
Joao Paulo Costa de Araujo (HUB)
Lars Grunske (HU Berlin)
Emilia Cioroaica (IESE)
Joshua Frey (IESE)
Nishanth Laxman (IESE)
Jessica Kelly (IKS)
Núria Mata (IKS)
Leopold Mareis (IKS)
Felippe Schmoeller da Roza (IKS)
João-Vitor Zacchi (IKS)
Edoardo Clementi (IKS)
Balahari Balu (IKS)
Karsten Albers (INC)
Victor Pollex (INC)
Moritz Zink (KIT)
Daniel Grimm (KIT)
Ann-Therese Nägele (KIT)
Theodosios Gkamas (MUL)
Markus Schweizer (RB)
Maike Salfeld (RB)
Jürgen Niehaus (SafeTRANS)
Eduard Dojan (SGS-TÜV)
Ramya Vinod (SGS-TÜV)
Fazli Faruk Okumus (THI)
Lenz Belzner (THI)
Stefan Kugele (THI)
Wenguang Xu (THI)
Zahra Zeinaly (THI)
Diego Alejandro Parra Guzman (TTT)
Pavel Nedvedicky (TUM)
Stefan Wagner (TUM)
Eva Zimmermann (TUM)
Janis Kröger (Uni OL)



Shad Ahammed (USG)
Julien Provost (VAL)
Murat Polat (OS)
Phil Stüpfert (OS)

Dokumentenhistorie:
2025-10-31: Review-Version
2025-11-06: Veröffentlichung

Kurzfassung

Das Projekt MANNHEIM–AutoDevSafeOps (ADSO) verfolgte das Ziel, Methoden und Werkzeuge zu entwickeln, die funktionale Sicherheit und kontinuierliche Entwicklung (DevOps) in der automobilen Softwareentwicklung zusammenführen. Im Mittelpunkt stand die kontinuierliche Betrachtung und Verbesserung von Sicherheit, Zuverlässigkeit und Vertrauen über den gesamten Lebenszyklus von Fahrzeugsystemen unter Einbeziehung von KI-basierten Systemfunktionen.

Automatisierte Fahrzeuge agieren in komplexen, offenen Umgebungen. Klassische Sicherheitsprozesse, die nur in der Entwicklungsphase greifen, reichen hierfür nicht mehr aus, um die Sicherheit nachhaltig zu gewährleisten. Es braucht laufzeitbegleitende Sicherheitsstrategien, die Updates, Monitoring und Zertifizierung in einen iterativen Entwicklungszyklus integrieren – unter Einhaltung normativer Standards wie ISO 26262 und ISO 21448.

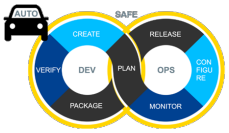
Im Rahmen des Projekts wurde mit dem ADSO-Prozess (AutoDevSafeOps-Prozess) ein generischer DevOps-Lebenszyklus entwickelt, der die sieben Phasen Plan, Create, Verify, Package, Configure, Release und Monitor umfasst. Dieser Prozess integriert sicherheitsrelevante Aktivitäten, Zertifizierungsnachweise und die Validierung von KI-Komponenten kontinuierlich in den Entwicklungs- und Betriebsablauf. Damit bildet er die methodische Grundlage für eine sichere und gleichzeitig agile Softwareentwicklung im automobilen Umfeld.

Auf dieser Basis wurden über 80 Technologiebausteine konzipiert und umgesetzt, die den DevOps-Prozess technisch und methodisch unterstützen. Dazu gehören unter anderem adaptive und resiliente Softwarearchitekturen, Mechanismen zur Automatisierung von Sicherheits- und Compliance-Prozessen, Werkzeuge zur Überwachung von Daten-Drift und KI-Modellen, Verfahren zur Simulation und Validierung von Softwareupdates im sogenannten Schattenmodus sowie formale Sicherheitsverträge, die Operational Design Domains (ODD) explizit in die Sicherheitsmodellierung einbeziehen.

Ein weiterer Schwerpunkt lag auf der Entwicklung von Konzepten und Werkzeugen für kontinuierliche Sicherheit (Continuous Safety). Durch die Automatisierung von Sicherheitsbewertungen, Tests und Zertifizierungsnachweisen wird es möglich, Sicherheit auch während des Betriebs und im Rahmen von Softwareupdates nachzuweisen – ein entscheidender Schritt in Richtung einer „Continuous Certification“. Ergänzend dazu wurden Methoden zur KI- und Laufzeitüberwachung entwickelt, die es erlauben, neuronale Netze während des Einsatzes zu beobachten und unsicheres Verhalten oder Abweichungen in den Eingangsdaten (Out-of-Distribution Detection) frühzeitig zu erkennen.

Die Praxistauglichkeit des ADSO-Ansatzes konnte in vier industriellen Use Cases demonstriert werden. Diese zeigen, dass die im Projekt entwickelten Methoden und Werkzeuge die Effizienz und Nachvollziehbarkeit sicherheitskritischer Entwicklungsprozesse verbessern und gleichzeitig die Zeit bis zur sicheren Auslieferung von Softwareupdates verkürzen.

ADSO belegt, dass sichere, kontinuierliche Softwareentwicklung im Fahrzeug möglich ist.



Der Ansatz reduziert Aufwand und Kosten für Sicherheitsnachweise, beschleunigt Releases und stärkt die Nachvollziehbarkeit sicherheitskritischer Änderungen. Damit entsteht eine skalierbare Grundlage für zukünftige softwaredefinierte Fahrzeuge – ein wesentlicher Schritt hin zu „Safe Continuous Deployment“.

Liste der Veröffentlichungen

[1] Abu Shad Ahammed, Md Shahi Amran Hossain, Sayeri Mukherjee, Roman Obermaisser, and Md. Ziaur Rahman. Integration of computer vision with adaptive control for autonomous driving using adore. 2025. Accepted at 2025 IEEE 6th International Symposium on Parallel and Distributed Systems (ISPDS).

[2] Abu Shad Ahammed, Md Shahi Amran Hossain, and Roman Obermaisser. A computer vision approach for autonomous cars to drive safe at construction zone. In *2025 IEEE 6th International Conference on Image Processing, Applications and Systems (IPAS)*, pages 1–6. IEEE, 2025.

[3] Yehia Ahmed. Probabilistic deep learning for safety monitoring in autonomous driving systems. Master's thesis, Technische Universität München, 2025.

[4] Yehia Ahmed, Felippe Roza, and Núria Mata. Temporal multimodal probabilistic transformers for safety monitoring in autonomous driving systems. volume 266, page 535 – 554, 2025.

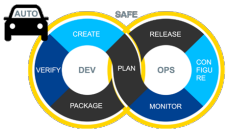
[5] Narmada Ambigapathy, Fatima Idrees, Katrin Gloewing, Charles Steinmetz, and Achim Rettberg. Autonomous driving pedestrian analysis: A digital twin approach using raspberry pi and carla simulator via mqtt. Springer, 2025. To appear in the Proceedings of the 8th International Embedded Systems Symposium (IESS 2024).

[6] Omnia Ammar. Benchmarking Hardware Accelerators for Neural Networks for Safety Critical Embdedded Systems. Bachelor's thesis, Karlsruher Institut für Technologie, 2024.

[7] Mehdi Azarafza, Ali Ehteshami Bejnordi, Katrin Glöwing, Fatima Idrees, Stefan Henkler, and Achim Rettberg. A parameter-efficient fine-tuning approach to enhance vision-language models for damaged and degraded traffic sign recognition. In *Proceedings of the 13th International Conference on Control, Mechatronics and Automation (ICCMA 2025)*. IEEE, 2025. To appear in ICCMA 2025.

[8] Mehdi Azarafza, Fatima Idrees, Ali Ehteshami Bejnordi, Charles Steinmetz, Stefan Henkler, and Achim Rettberg. Human-in-the-loop reasoning for traffic sign detection: Collaborative approach yolo with video-llava. In Kohei Arai, editor, *Advances in Information and Communication*, volume 1283 of *Lecture Notes in Networks and Systems*, pages 160–171. Springer, Cham, 2025.

[9] Jan Steffen Becker, Björn Koopmann, Ingo Stierand, and Lukas Westhofen. Providing evidence for correct and timely functioning of software safety mechanisms. In *Software Engineering 2023 Workshops*, pages 66–77. Gesellschaft für Informatik e.V., Bonn, 2023.



[10] Benjamin Bieler. Development of a graphical tool for visualizing stpa control structures. Bachelor's thesis, Technische Universität München, 2025.

[11] Pauline Blohm, Martin Fränzle, Paula Herber, Paul Kröger, and Anne Remke. Towards probabilistic contracts for intelligent cyber-physical systems. In *International Symposium on Leveraging Applications of Formal Methods*, pages 26–47. Springer, 2024.

[12] Matthias Braun. Software Requirements Engineering Speedup with Large Language Models. Master's thesis, Karlsruher Institut für Technologie, 2025.

[13] Nick Breit. Impact and Analysis of the EU AI Act's Transparency Requirements and Their Relation to Existing Standards. Bachelor's thesis, Technische Universität München, 2024.

[14] Filip Budic. Design and implementation of a scenario-based testing pipeline for autonomous driving. Bachelor's thesis, Technische Universität München, 2025.

[15] Hanbo Cai, Pengcheng Zhang, Hai Dong, Lars Grunske, Shunhui Ji, and Tianhao Yuan. Adversarial example-based test case generation for black-box speech recognition systems. *Softw. Test. Verification Reliab.*, 33(5), 2023.

[16] Adrien Castella. Active learning in ML-training for autonomous vehicles. Master's thesis, Technische Universität München, 2024.

[17] Asmita Chandrakar. Analyzing the impact of devops on safety of automotive software. Master's thesis, Universität Stuttgart, 2023.

[18] Arda Cihan. Safety Monitor Determination Tool to Continiously Improve Safety of ML-Based Automated Driving Systems. Bachelor's thesis, Karlsruher Institut für Technologie, 2025.

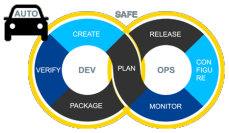
[19] João Paulo Costa de Araujo, Balahari Vignesh Balu, Eik Reichmann, Jessica Kelly, Stefan Kugele, Núria Mata, and Lars Grunske. Applying concept-based models for enhanced safety argumentation. In *ISSRE*, pages 272–283. IEEE, 2024.

[20] Moritz Friebe. Comparison of Online and Inline Anomaly Detection Strategies in Neural Networks for Validation of Inferences in Safety-Critical cyber-Physical Systems. Bachelor's thesis, Karlsruher Institut für Technologie, 2024.

[21] Tianyi Gao. AUTCOM: Automated Collection of Significant Training Data for Continuous Updates of ML-based Driving Applications. Master's thesis, Karlsruher Institut für Technologie, 2023.

[22] Giulio Gerloni. Monitoring Framework for Continuous Safety Assurance in Autonomous Driving: A Construction Zone Use Case. Master's thesis, Freie Universität Bozen, 2024.

[23] Habeeb Rilwan Giwa. Carla based radar perception by applying ros2 bridge. Bachelor's thesis, Hochschule Hamm-Lippstadt, 2025.



[24] Daniel Grimm, Moritz Zink, Marc Schindewolf, and Eric Sax. Adaptive cybersecurity monitoring for resilient vehicular architectures. In *2023 IEEE Vehicular Networking Conference (VNC)*, pages 41–48, 2023.

[25] Daniel Grimm, Moritz Zink, Marc Schindewolf, and Eric Sax. Cyber situational awareness in vehicle security operations: Holistic monitoring and a data model. In *2024 20th International Conference on Network and Service Management (CNSM)*, pages 1–7, 2024.

[26] Houssem Guissouma, Moritz Zink, and Eric Sax. Continuous safety assessment of updated supervised learning models in shadow mode. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 301–308, 2023.

[27] Moritz Hetzel. Comparison of Middleware Architectures for Runtime Evaluation of Automotive Software Updates in Shadow Mode. Bachelor's thesis, Karlsruher Institut für Technologie, 2023.

[28] Md Shahi Amran Hossain, Abu Shad Ahammed, Divya Prakash Biswas, and Roman Obermaisser. Impact analysis of data drift towards the development of safety-critical automotive system. In *2024 International Symposium ELMAR*, pages 325–330. IEEE, 2024.

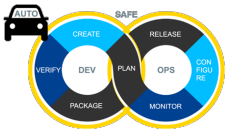
[29] Md Shahi Amran Hossain, Abu Shad Ahammed, Sayeri Mukherjee, and Roman Obermaisser. Enhanced drift-aware computer vision architecture for autonomous driving. 2025. Accepted at 2025 IEEE 51st Annual Conference of the Industrial Electronics Society (IECON).

[30] Fatima Idrees, Narmada Ambigapathy, Peer Adelt, and Achim Rettberg. Ai-driven traffic compliance in autonomous vehicles: A scalable framework with large language models on edge platforms. In *Proceedings of the 2025 IEEE International Automated Vehicle Validation Conference (IAVVC)*. IEEE, 2025. To appear in IAVVC 2025.

[31] Ivan Ivanov. Defining a domain-specific language to document test specifications for library qualification in accordance to iso 26262. Bachelor's thesis, Technische Universität München, 2025.

[32] Shunhui Ji, Changrong Huang, Bin Ren, Hai Dong, Lars Grunske, Yan Xiao, and Pengcheng Zhang. Taefuzz: Automatic fuzzing for image-based deep learning systems via transferable adversarial examples. *ACM Trans. Softw. Eng. Methodol.*, January 2025. Just Accepted.

[33] Chidvilas Karpenahalli Ramakrishna, Adithya Mohan, Zahra Zeinaly, and Lenz Belzner. The evolution of criticality in deep reinforcement learning. In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025)-Volume 3*, pages 217–224. SciTePress, 2025.



[34] Jaewoo Kim. Further development and redesign of xstampp 4: Migration to a modern web application with new backend and cloud deployment. Bachelor's thesis, Technische Universität München, 2025.

[35] Janis Kröger and Martin Fränzle. Updates at runtime for cyber physical systems. a game theoretic approach. In *Software Engineering 2023 Workshops*, pages 54–65. Gesellschaft für Informatik eV, 2023.

[36] Janis Kröger and Martin Fränzle. Mode management in contract-based design. In *SE 2024-Companion*, pages 31–42. Gesellschaft für Informatik eV, 2024.

[37] Janis Kröger, Björn Koopmann, Ingo Stierand, and Martin Fränzle. Contract-based specification of mode-dependent timing behavior. *Innov. Syst. Softw. Eng.*, 20(1):31–47, September 2023.

[38] Janis Kröger, Ingo Stierand, and Martin Fränzle. Ensuring integration conditions during the update of cyber-physical systems. In *Formal Methods for Industrial Critical Systems: 30th International Conference, FMICS 2025, Aarhus, Denmark, August 27–28, 2025, Proceedings*, page 203. Springer Nature, 2025.

[39] Maria Lelyukh. Concretizing abstract scenarios for the simulation of ads. Bachelor's thesis, Technische Universität München, 2025.

[40] Moru Liu. A Multi-Modal Framework for Reliable Uncertainty-Aware Out-of-Distribution Object Segmentation. Master's thesis, Technische Universität München, 2024.

[41] Moru Liu, Hao Dong, Jessica Ivy Kelly, Olga Fink, and Mario Trapp. Extremely simple multimodal outlier synthesis for out-of-distribution detection and segmentation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

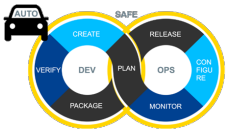
[42] Manoj Luitel. Hardware-in-the-loop integration of adaptive cruise control alert system using carla and fpga. Bachelor's thesis, Hochschule Hamm-Lippstadt, 2025.

[43] Manasa Mariam Mammen. Evaluation of AI methods for scenario variation to support the validation of highly automated driving functions using real measurement data. Master's thesis, Universität Stuttgart, 2024.

[44] Leopold Mareis. Optimizing Experimental Design for Causal Effect Estimation with Partial Measurements. *PCIC*, 2024.

[45] Lucas Mauser and Stefan Wagner. Centralization potential of automotive e/e architectures. *Journal of Systems and Software*, 219:112220, 2025.

[46] Lucas Mauser, Eva Zimmermann, Pavel Nedvědický, Tobias Eisenreich, Moritz Wäschle, and Stefan Wagner. Towards mixed-criticality software architectures for centralized hpc platforms in software-defined vehicles: A systematic literature review. In Vasilios Andrikopoulos, Cesare Pautasso, Nour Ali, Jacopo Soldani, and Xiwei Xu, editors, *Software Architecture*, pages 144–160, Cham, 2025. Springer Nature Switzerland.



[47] F. Mayer, J. Apelt, Ralf Münzenberger, F. Kraft, and C. Winkler. *Event-Chain-Focused Development of System Architectures Makes Complex Systems Manageable*, pages 645–662. VDI Wissensforum, 2023.

[48] Johannes Mock. Enhancing AI Safety: A Comparative Study of Vision Transformers and Convolutional Autoencoders in Novel Scenarios. Master's thesis, Karlsruher Institut für Technologie, 2025.

[49] Marc Müller. Data-Driven Safety Specification With Contract Mining. Bachelor's thesis, Karlsruher Institut für Technologie, 2024.

[50] Ann-Therese Tabea Nägele and Eric Sax. Smart testing for smart vehicles: Conditional regression strategies for software updates. In *2025 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*, 2025. accepted for publication.

[51] Ann-Therese Tabea Nägele, Marc Schindewolf, and Eric Sax. Collaborative continuous testing of automotive services (coco test). In *2025 IEEE International Systems Conference (SysCon)*, pages 1–8, Montreal, QC, Canada, April 2025. IEEE.

[52] Marcel Namyslo. ATHENA: Advanced Learning through Efficient Transparent Model Harnessing with Explainable Data. Bachelor's thesis, Karlsruher Institut für Technologie, 2025.

[53] Pavel Nedvedicky, Eva Zimmermann, and Stefan Wagner. Towards integrating STPA into DevOps lifecycle. In *CARS@EDCC2024 Workshop - Critical Automotive Applications: Robustness & Safety*, Leuven, Belgium, April 2024.

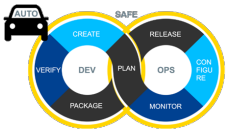
[54] Pavel Nedvědický, Eva Zimmermann, and Stefan Wagner. STPA-based continuous safety verification of autonomous driving systems during simulation. In *2025 IEEE Intelligent Vehicles Symposium (IV)*, pages 2532–2537, June 2025.

[55] Fazli Faruk Okumus, Amra Ramic, and Stefan Kugele. A systematic mapping study on contract-based software design for dependable systems. *CoRR*, abs/2505.07542, 2025.

[56] Fazli Faruk Okumus, João-Vitor Zacchi, Maike Salfeld, Markus Schweizer, Núria Mata, and Stefan Kugele. Runtime monitor synthesis for automotive software architectures. In *Software Architecture - 19th European Conference, ECSA 2025*, LNCS. Springer, 2025. to appear.

[57] Tom Olesch. Development of an In Vitro Method for Run-Time Synchronous Functional Monitoring of Neural Networks. Bachelor's thesis, Karlsruher Institut für Technologie, 2023.

[58] M. Hossain P. M. Savariya. Applying Explainable AI Technique On The Object Detection of Traffic Signs Using YOLOv7 Algorithm. Master's thesis, Universität Siegen, 2024.



[59] Gopal Chitrasen Panigrahi. Usage of external data exchanged in intelligent transport systems to ensure road safety in an automated vehicle. Master's thesis, Universität Stuttgart, 2023.

[60] Tobias Pfaller and Richard Membarth. Exploiting data redundancy in im2col convolutions. *PARS-Mitteilungen*, 37, 2025.

[61] Andrijana Radic. Exploring runtime monitoring techniques in the automotive domain for advanced Driver-Assistance systems. Bachelor's thesis, Universität Stuttgart, 2023.

[62] Amra Ramic and Stefan Kugele. A systematic mapping study on software architecture for ai-based mobility systems. *CoRR*, abs/2506.01595, 2025.

[63] Daniel Ratiu, Tihomir Rohlinger, Torben Stolte, and Stefan Wagner. Towards an argument pattern for the use of safety performance indicators. In Andrea Ceccarelli, Mario Trapp, Andrea Bondavalli, Erwin Schoitsch, Barbara Gallina, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security. SAFECOMP 2024 Workshops*, pages 160–172, Cham, 2024. Springer Nature Switzerland.

[64] M. Hossain S. Mukherjee. Context Aware Anomaly Detection for Managing Data Drift in Autonomous Driving. Master's thesis, Universität Siegen, 2025.

[65] Aniket Salvi, Gereon Weiss, and Mario Trapp. Explaining unreliable perception in automated driving: A fuzzy-based monitoring approach.

[66] Benjamin Schray. Explainable Out-of-Distribution Detection for Safe Neural Networks in the Context of Cyber-Physical Systems. Bachelor's thesis, Karlsruher Institut für Technologie, 2024.

[67] Tobias Senger. Enhancing automotive safety through an ADAS violation dashboard. Master's thesis, Universität Stuttgart, 2024.

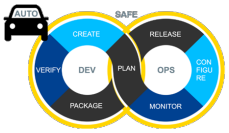
[68] A. H. Shamseddin. Increasing Edge-case Testing Coverage through Guided AI Synthetic Scenario Generation. Master's thesis, Technische Universität München, 2025.

[69] Jakob Stadelmann, Andreas Schweiger, and Richard Membarth. Evaluation of parallelization frameworks for the linear assignment problem. *PARS-Mitteilungen (PARS Workshop 2025)*, 38:1–14, 2026.

[70] Ingo Stierand, Lukas Westhofen, and Willem Hagemann. *On Using Ontologies in the Engineering of Intelligent Cyber-Physical Systems*, pages 54–75. Springer Nature Switzerland, Cham, 2025.

[71] Hendrik Stumpf. Development of explainable neural networks by learning distinct concepts. Bachelor's thesis, Karlsruher Institut für Technologie, 2025.

[72] Patrick Tache. Enhancing root cause analysis in safety-critical systems using knowledge graphs. Bachelor's thesis, Technische Universität München, 2025.



[73] Patrick Uven, Ralf Stemmer, and Gregor Nitsche. A modular architecture template for resource modeling in software-defined vehicles. In *2025 20th European Dependable Computing Conference Companion Proceedings (EDCC-C)*, pages 213–217, 2025.

[74] Ruben Verma. Charging in the cloud : implementing a virtualized runtime environment for EVSE and EV interoperability testing. Master's thesis, Universität Stuttgart, 2024.

[75] Lukas Westhofen, Jean Christoph Jung, and Daniel Neider. Temporal conjunctive query answering via rewriting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(14):15221–15229, Apr. 2025.

[76] Lukas Westhofen, Christian Neurohr, Jean Christoph Jung, and Daniel Neider. Answering temporal conjunctive queries over description logic ontologies for situation recognition in complex operational domains. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–187, Cham, 2024. Springer Nature Switzerland.

[77] Lukas Westhofen, Christian Neurohr, Jean Christoph Jung, and Daniel Neider. Topllet: An optimized engine for answering metric temporal conjunctive queries. In Nathaniel Benz, Divya Gopinath, and Nija Shi, editors, *NASA Formal Methods*, pages 314–321, Cham, 2024. Springer Nature Switzerland.

[78] Lukas Westhofen, Ingo Stierand, Jan Steffen Becker, Eike Möhlmann, and Willem Hagemann. Towards a congruent interpretation of traffic rules for automated driving – experiences and challenges. In *JURIX Workshop on Methodologies for Translating Legal Norms into Formal Representations (LN2FR)*, 2022.

[79] Wenguang Xu and Richard Membarth. Exploration of efficient computation for trajectory planning via fixed-point arithmetic. *PARS-Mitteilungen*, 37, 2025.

[80] Eva Zimmermann. Towards combining STPA and safety-critical runtime monitoring. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, ICSE-Companion '24, pages 252–254, New York, NY, USA, May 2024. Association for Computing Machinery.

[81] Moritz Zink, Daniel Grimm, and Eric Sax. Aurora networks: Auto-associative universal real-time outlier risk assessment networks. In Martin Törngren, Barbara Gallina, Erwin Schoitsch, Elena Troubitsyna, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security. SAFECOMP 2025 Workshops*, pages 499–510, Cham, 2026. Springer Nature Switzerland.

[82] Moritz Zink, Luca Seidel, Daniel Baumann, Daniel Grimm, and Eric Sax. Artemis - adaptive response and tracking of external model inputs in safety-critical systems. 2025. Accepted at 2025 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE).

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Ansatz und zentrale Beiträge . . . 3
- 1.2 Struktur des Berichts . . . 4

2 Hintergrund und Verwandte Arbeiten **6**
- 2.1 Verwandte Arbeiten . . . 6
- 2.2 Verwandte Projekte . . . 10

3 Softwareentwicklungszyklus: Der ADSO Prozess **15**
- 3.1 ADSO Process . . . 16
- 3.2 Technologiebausteine und Artefakte . . . 28

4 Projektergebnisse und Lösungen **42**
- 4.1 Integration sicherheitsrelevanter Aktivitäten in den DevOps-Prozess . . . 42
- 4.2 Integration sicherheitsrelevanter Artefakte in ein formalisiertes Nachverfolgbarkeits-Informationsmodell . . . 46
- 4.3 Erweiterung der Spezifikation von Designverträgen zur Einbeziehung von Operational Design Domains (ODD) und Modi zur effektiven Modellierung von Sicherheitsanforderungen . . . 49
- 4.4 Adaptive Softwarearchitekturen für die Resilienz sicherheitskritischer Systeme 55
- 4.5 Erweiterung des Sicherheitslebenszyklus zur Gewährleistung der Vertrauenswürdigkeit von KI-Funktionen über den gesamten Produktlebenszyklus hinweg 66
- 4.6 Optimierung des Update-Prozesses durch frühe Validierung von Software-Updates in einer simulierten Umgebung und frühzeitige Ausführung im Schattenmodus . . . 72
- 4.7 Iterative und kontinuierliche Absicherung sicherheitskritischer Funktionen durch Synthese von Felddaten . . . 78
- 4.8 Innovative Lifecycle Modelle für den sicheren Einsatz Neuronaler Netze in CPS durch neuartige Modellstrukturen zu OOD Erkennung . . . 83
- 4.9 Monitoring Strategien . . . 89
- 4.10 Kontinuierliche Integrationspipeline für System-of-Systems im Automobilbereich 98
- 4.11 Standardisierungsbedarfe . . . 101

5 Zusammenfassung **107**

1 Einleitung

Ein tiefgreifender Wandel findet derzeit in der Entwicklung sicherheitskritischer automobiler Systeme statt, ausgelöst durch den Aufstieg automatisierter und KI-basierter Systeme, die in komplexen, offenen Kontexten operieren. Automatisierte Fahrzeuge sind Systeme, die zuverlässig in Umgebungen funktionieren müssen, in denen sich die Bedingungen unvorhersehbar ändern und potenzielle Gefahren nicht immer im Voraus bekannt sind. Infolgedessen erlebt die Automobilindustrie ein rasantes Wachstum der Softwarekomplexität, angetrieben durch die steigende Funktionalität, die zur Realisierung automatisierter Fahrfunktionen erforderlich ist, sowie durch Fortschritte in der KI-Technologie.

In einer Reihe von Vorschriften, Sicherheitsstandards und Normen werden diese Herausforderungen adressiert, indem sie Leitlinien und Empfehlungen für Entwicklungsprozesse und Betrieb bereitstellen. Als wesentliche Neuerung berücksichtigen sie die Tatsache, dass eine traditionelle Entwicklung, die den die Systementwicklung begleitenden Sicherheitsprozess einmalig zur Entwicklungszeit durchführt, für die betrachteten Systeme nicht ausreichend ist. Stattdessen wird die Sicherheitsbetrachtung auf die Betriebsphase ausgeweitet, durch Maßnahmen, die Probleme — auch bisher unbekannte — zur Laufzeit erkennen und mitigieren können.

Diese Standards sind jedoch in vielen Fällen generisch gehalten und weisen deshalb erhebliche Lücken auf, wenn sie in realen Szenarien praktisch angewendet werden sollen. Insbesondere liefern sie oft keine konkreten Technologien und Methoden zur Beantwortung der Frage, wie die Systeme entwickelt und betrieben werden sollen. Daher bleiben einerseits viele Fragen offen, wie Sicherheit in kontinuierlich weiterentwickelten Systemen zuverlässig gewährleistet werden kann. Andererseits eröffnet dies Spielräume zur Etablierung neuer Technologien und Methoden, die die Prozessanforderungen der einschlägigen Sicherheitsstandards erfüllen.

Gerade die Standards für automatisierte Systeme fordern eine systematische Rückmeldung zur Qualität der Sicherheitskonzepte während des Betriebs auf Basis geeigneter Felddaten, was oft als *DevOps*-Designparadigma bezeichnet wird. Es führt einen inkrementellen Entwicklungszyklus ein, indem es kontinuierliches Monitoring nach dem Deployment und während des Betriebs vorsieht. Erkenntnisse aus diesem Monitoring können dann zur Sicherheitsbewertung herangezogen und für inkrementelle Produktverbesserungen genutzt werden.

Dieser Ansatz ermöglicht die frühzeitige Erkennung und Behebung anspruchsvoller Softwareprobleme durch zeitnahe Updates. Angesichts der zunehmenden Konnektivität von automobilen Produkten und Dienstleistungen ist eine schnelle und feingranulare Überwachung und Analyse der Sicherheitsleistung des Systems realisierbar und ermöglicht eine frühzeitige Problembehebung. Die Herausforderung besteht darin, die geeigneten Felddaten zu identifizieren, die Einblicke in die Sicherheitsleistung liefern können. Wenn relevante Einbußen in der Sicherheitsleistung festgestellt werden, muss das System mit verbessertem Sicherheitsverhalten aktualisiert werden.

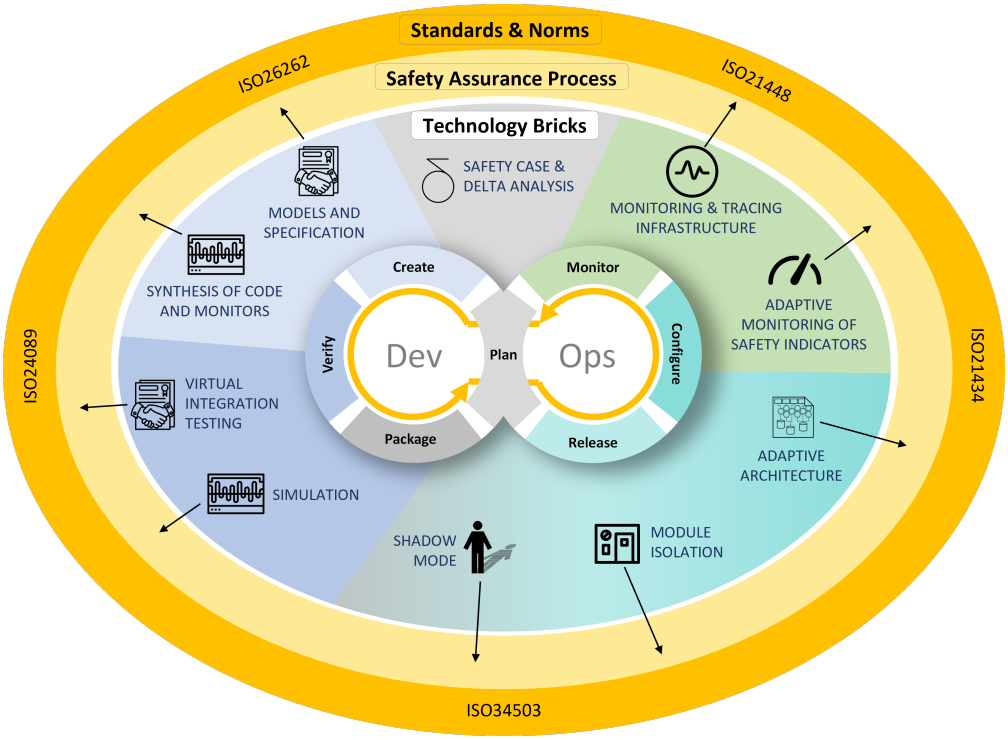
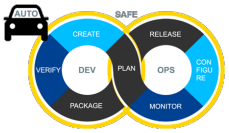


Abbildung 1.1: MANNHEIM-AutoDevSafeOps-Ansatz und zentrale Lösungsbereiche

Über den Lebenszyklus des Fahrzeugs hinweg führt dies zu einer kontinuierlichen Weiterentwicklung der Software durch regelmäßige Updates. Dies verdeutlicht den Bedarf an einem robusten und dennoch flexiblen Sicherheitsansatz sowie einem Ansatz für sichere inkrementelle Updates im Fahrzeug – insbesondere bei der Integration von Machine Learning. Die traditionelle Sicherheitsentwicklung kann diese inkrementellen Updates mit erheblichem personellen Aufwand bewältigen, jedoch erweist sich ein derart arbeitsintensiver Prozess oft als wirtschaftlich nicht tragfähig und zu langsam. In einem hochkompetitiven Marktumfeld mit starkem Time-to-Market-Druck ist es unpraktikabel, die umfangreichen Nachweise und Artefakte für einen umfassenden Safety Case rein manuell zu erzeugen. Da die Industrie schnellere und häufigere Releases anstrebt, wird deutlich, dass ein gewisser *Automatisierungsgrad* unerlässlich ist, um Effizienz und Wirtschaftlichkeit sicherzustellen.

Die Bewältigung dieser Herausforderungen und die Weiterentwicklung effektiver DevOps-Systeme ist das Ziel des Projekts MANNHEIM-AutoDevSafeOps (ADSO). Im Rahmen der Projektarbeit wurde ein iterativer DevOps-Lebenszyklus eingeführt – *der ADSO-Prozess* – für Systemdesign, der die Sicherheitsgewährleistung als zentrales Element integriert. Dieser Lebenszyklus sieht eine kontinuierliche Einbindung sicherheitsrelevanter Überlegungen im Entwicklungsprozess vor, wodurch jedes inkrementelle Update im Einklang mit Industriestandards validiert und dokumentiert werden kann.

Ein zentrales Element unseres Ansatzes ist die Entwicklung von *Technologiebausteinen*, die die generischen Vorgaben der Sicherheitsstandards und -technologien in praktisch umsetzbare



Module überführen. Durch die Einbettung dieser Bausteine in einen DevOps-Zyklus wird die durchgehende Überwachung sicherheitsrelevanter Informationen gewährleistet und neue Erkenntnisse oder Artefakte können nahtlos in den Safety Case integriert werden.

Zur Validierung des vorgeschlagenen Ansatzes wurde der generische ADSO-Prozess auf vier Anwendungsfälle angewendet, die jeweils zeigen, wie sich der Lebenszyklus in unterschiedlichen industrienahen Szenarien instanziieren lässt. Durch die Darstellung dieser Anwendungsfälle zeigen wir, dass es nicht nur möglich, sondern auch effizient ist, Sicherheit durch einen automatisierten und iterativen DevOps-Prozess zu gewährleisten. Die Vorteile liegen auf der Hand: Zeit- und Kostenaufwand für die Erstellung und Pflege von Sicherheitsdokumentation werden erheblich reduziert, und sicherheitskritische Funktionen lassen sich leichter im Rahmen kontinuierlicher Releases verwalten. Dieser Ansatz entspricht den Realitäten industrieller Praxis, in der begrenzte Budgets und schnelle Innovationszyklen auf den kompromisslosen Anspruch an Sicherheit treffen.

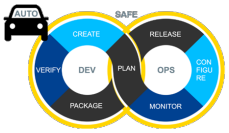
1.1 Ansatz und zentrale Beiträge

Obwohl der Begriff DevOps in verschiedenen Kontexten verwendet wird, stammt er im Bereich des Software-Lebenszyklusmanagements aus der Idee, die Softwareentwicklung auf der einen Seite und IT-Betrieb sowie Qualitätsmanagement auf der anderen Seite enger miteinander zu verzahnen. DevOps umfasst Methoden und Technologien, die eine schnellere und effizientere Softwareentwicklung fördern und gleichzeitig die Softwarequalität verbessern. Seitdem wurde die DevOps-Idee in verschiedenen Bereichen angepasst und erweitert, etwa für Geschäftsprozesse und die Einbeziehung von Sicherheitsaspekten.

Das AutoDevSafeOps-Projekt verfolgt die gleichen Ziele wie DevOps – nämlich die Verbesserung von Effizienz und Qualität –, wobei der Fokus selbstverständlich auf dem Aspekt der funktionalen Sicherheit liegt. In diesem Zusammenhang ist der im Projekt definierte generische Lebenszyklusprozess ein zentrales Element, um DevOps mit einem kontinuierlichen und konsistenten Sicherheitsgewährleistungsprozess bei der Entwicklung automatisierter Fahrzeuge in Einklang zu bringen. Er dient als Brücke zwischen den relevanten normativen Standards auf der einen Seite und den im Projekt entwickelten praktischen Lösungen auf der anderen Seite. Im Projekt wurden über 80 einzelne Technologiebausteine entwickelt. Darunter befinden sich neue Artefakttypen zur Erzeugung von Arbeitsergebnissen im Entwicklungsprozess, neuartige Methoden, etwa zur Durchführung von Gefährdungsanalysen und zur Testgenerierung, sowie insbesondere Werkzeuge zur Automatisierungsunterstützung.

Die folgende Liste beschreibt zentrale Zielsetzungen des Projekts, aus denen die Entwicklung der Technologiebausteine hervorgegangen ist:

- Stärkung der Resilienz sicherheitskritischer Systeme durch eine adaptive Softwarearchitektur, die ein reaktives Umschalten in einen sicheren Betriebsmodus bei Erkennung unsicheren Verhaltens erlaubt.
- Erweiterung des Sicherheitslebenszyklus zur Sicherstellung der Vertrauenswürdigkeit von KI-Funktionen über den gesamten Produktlebenszyklus hinweg.



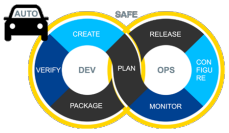
- Ausweitung der Spezifikation von Designverträgen zur Einbeziehung von Operational Design Domains (ODD) und Betriebsmodi zur effektiven Modellierung von Sicherheitsanforderungen.
- Optimierung des Update-Prozesses durch frühzeitige Validierung von Softwareupdates in einer Simulationsumgebung und frühe Erprobung im Schattenbetrieb.

1.2 Struktur des Berichts

Das vorliegende Dokument ist wie folgt aufgebaut: Kapitel 2 liefert die notwendigen Grundlagen, auf denen die Entwicklungen des Projekts aufbauen. Dazu zählen relevante Normen und Regelwerke, verwandte Arbeiten und weitere Projekte. Besonders wichtig ist dabei die Definition von DevOps, die inkrementelle Entwicklungsprozesse beschreibt, welche den gesamten Lebenszyklus von Produkten und Systemen einschließlich regelmäßiger Updates umfassen. DevOps kann als ein Referenzprozess aufgefasst werden, der als Zielstruktur für die Integration aller sicherheitsrelevanten Themen dient, die im Projekt entwickelt wurden, um den Gesamtbeitrag des Projekts zu diesem wichtigen Aspekt bewerten zu können. Kapitel 3 gibt einen Überblick darüber, wie diese Ziele durch die Technologiebausteine realisiert wurden. Die Bausteine wurden entlang einer konkreten Ausprägung des DevOps-Zyklus entwickelt, den wir als *ADSO-Prozess* bezeichnen. Abschnitt 3.2 enthält eine Liste der Technologiebausteine mit Verweisen auf die entsprechenden Dokumente mit weiterführenden Informationen. Kapitel 4 beschreibt die von den Projektpartnern im Rahmen des Projekts erarbeiteten Lösungen. Abschnitt 4.11 fasst diese Beiträge mit einer Betrachtung von Standardisierungsbedarfen für die Entwicklung sicherheitskritischer Systeme zusammen. Kapitel 5 schließt das Dokument mit einer Zusammenfassung ab.

Die in diesem Dokument vorgestellten Projektergebnisse wurden in einer Serie von Projektberichten ausführlich dargestellt:

- D1.1/1.2 [24] stellt die Verbindung der Projektergebnisse mit den wesentlichen im Automotive Sektor genutzten (Sicherheits-)Standards dar und identifiziert fehlende Standardisierungen.
- D2.1/2.2 [20, 26] stellt den in dem Projekt erarbeiteten Prozess dar, und wie dieser sich in den Stand der Regularien, Normen und Technik zur Entwicklung sicherheitskritischer Fahrzeugsysteme einbettet. Darüber hinaus werden hier Projektergebnisse zur Resilienz von KI-basierten Systemen dokumentiert.
- D3.1/3.2 [21, 27] bietet einen tiefgehenden Überblick über die methodischen Grundlagen und theoretischen Ergebnisse.
- D4.1/4.2 [22, 28] liefert technische Details zu den Technologiebausteinen im Bereich Modellierung und Validierung.
- D5.1/5.2 [23, 29] behandelt die Bausteine zu Middleware, Kommunikation und verteilten Systemen.



- D6.1 [19] definiert die Anforderungen aus den in dem Projekt behandelten Anwendungsfällen.
- D6.2 [25] beschreibt, wie die Anwendungsfälle umgesetzt werden sollen.
- D6.3 [30] beleuchtet schließlich die Anwendung der Lösungen in den einzelnen Anwendungsfällen.

2 Hintergrund und Verwandte Arbeiten

2.1 Verwandte Arbeiten

2.1.1 DevOps-Zyklus

Faktoren, die kürzlich in der Automobilindustrie beobachtet wurden, z.B. das zunehmende Softwarevolumen, der Übergang zu einem dienstbasierten Geschäftsmodell und der Betrieb in offenen Kontexten, erfordern die kontinuierliche Entwicklung und Auslieferung von Funktionalitäten auf schnelle und zuverlässige Weise. Als Mittel zur Bereitstellung von Modellen, Werkzeugen und Prozessen für die Entwicklung sicherheitskritischer Automobilsoftware in diesem komplexen Kontext nutzt MANNHEIM-AutoDevSafeOps DevOps-Prozesse und -Praktiken.

Was ist DevOps

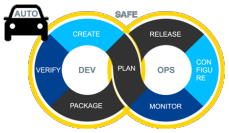
DevOps, eine Abkürzung für Development and Operations (Entwicklung und Betrieb), ist eine Sammlung kultureller Philosophien, Praktiken und Werkzeuge, die darauf abzielen, die Zusammenarbeit, Kommunikation und Integration zwischen Entwicklungs- und Betriebsteams während des gesamten Softwareentwicklungslebenszyklus zu verbessern [50, 70].

In der Literatur wird DevOps als Erweiterung moderner Softwareentwicklungsprozesse im Sinne von Prinzipien gesehen [50, 70, 63]. Um schnelles Feedback von Nutzern zu erhalten, sind moderne Softwareentwicklungsprozesse, wie z.B. agile Methoden, durch häufige und schnelle Produktveröffentlichungen gekennzeichnet [71]. Diese Methoden legen jedoch üblicherweise wenig Fokus auf deployment-spezifische Verfahren [63], was zu stressigen Übergängen in die Produktion [71] und zu widersprüchlichen Zielen hinsichtlich Agilität und Stabilität [7] innerhalb einer Organisation führen kann. DevOps begegnet dem, indem Entwicklungs- und Betriebsteams durch eine kollaborative Kultur und eine Reihe technischer Praktiken stärker integriert werden, die Betriebsaufgaben als routinemäßige Entwicklungsaktivitäten betrachten [70]. Auf diese Weise zielt DevOps darauf ab, eine effektive und reibungslose Softwareauslieferung zu ermöglichen und so das volle Potenzial moderner Entwicklungsprozesse auszuschöpfen [63].

Zentrale Prinzipien von DevOps

Im Kern verkörpert DevOps eine Reihe zentraler Prinzipien, die die Softwareentwicklung und -bereitstellung in der heutigen schnelllebigen und dynamischen Technologielandschaft optimieren. Einige davon sind [50, 70, 7]:

1. **Zusammenarbeit und Kommunikation:** DevOps betont das Auflösen von Silos innerhalb einer Organisation durch Förderung offener und effizienter Kommunikation sowie



Zusammenarbeit zwischen Entwicklung, Betrieb und anderen Beteiligten im Softwarebereitstellungsprozess. Teams arbeiten gemeinsam auf gemeinsame Ziele hin.

2. **Integration:** DevOps legt den Fokus auf die Integration und Automatisierung des gesamten Softwareentwicklungszyklus – von der Code-Entwicklung über das Testen bis hin zu Bereitstellung und Überwachung. Dies minimiert manuelle Fehler und erhöht die Effizienz.

3. **Automatisierung:** Automatisierung ist ein Grundpfeiler von DevOps und rationalisiert sich wiederholende und zeitaufwändige Aufgaben. Dazu gehören automatisierte Build-Prozesse, Tests, Bereitstellungen und Infrastruktur-Provisionierung, was zu einem effizienteren und standardisierten Arbeitsablauf führt.

4. **Continuous Delivery (CD) und Continuous Integration (CI):** DevOps setzt auf einen kontinuierlichen und iterativen Entwicklungsansatz. Continuous Integration umfasst das häufige und automatisierte Integrieren von Codeänderungen in ein gemeinsames Repository, während sich Continuous Delivery auf die Automatisierung des Bereitstellungsprozesses konzentriert, um jederzeit eine veröffentlichungsfähige Software zu gewährleisten.

5. **Überwachung und Feedback:** DevOps legt großen Wert auf die kontinuierliche Überwachung von Anwendungen und Infrastruktur, um Erkenntnisse über Leistung, Nutzung und potenzielle Probleme zu gewinnen. Diese Rückkopplungsschleife ermöglicht datengestützte Entscheidungen und verbessert Entwicklungs- und Bereitstellungsprozesse.

6. **Agilität:** DevOps fördert Agilität und Reaktionsfähigkeit auf sich ändernde Anforderungen und Kundenfeedback. Es ermöglicht schnelle Anpassungen und Updates der Software, sodass Organisationen wettbewerbsfähig bleiben.

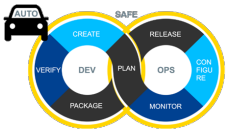
7. **Skalierbarkeit:** DevOps-Praktiken stellen sicher, dass Software leicht skalierbar ist, um steigende Anforderungen sowie Benutzer- oder Datenwachstum zu bewältigen – besonders wichtig für moderne, oft cloud-basierte Anwendungen.

8. **Sicherheitsintegration:** DevOps integriert Sicherheitspraktiken nahtlos in den Softwareentwicklungsprozess, sodass Sicherheitsaspekte von Anfang an berücksichtigt und nicht nachträglich eingeführt werden.

9. **Kultur der kontinuierlichen Verbesserung:** DevOps fördert eine Kultur des Lernens, Experimentierens und der kontinuierlichen Verbesserung. Teams reflektieren regelmäßig ihre Prozesse, identifizieren Optimierungspotenziale und streben nach effizienteren Arbeitsweisen.

DevOps-Lebenszyklus

DevOps wird oft als Unendlichkeitszeichen dargestellt, das die kontinuierliche Natur des Prozesses symbolisiert [7]. Der Lebenszyklus umfasst mehrere miteinander verbundene Phasen,



die jeweils entscheidend sind für Effizienz und kontinuierliche Verbesserung durch Automatisierung, Feedbackschleifen und Zusammenarbeit zwischen Entwicklungs- und Betriebsteams. Die zentralen Phasen des DevOps-Lebenszyklus sind [5]:

- **Planen:** In dieser Phase definieren Teams ihre Ziele und Anforderungen für das in diesem Zyklus zu entwickelnde Softwareprodukt.
- **Erstellen oder Entwickeln:** Basierend auf dem erstellten Plan konzentrieren sich die Entwicklungsteams auf den Aufbau der Lösung und die Codeüberprüfung.
- **Verifizieren:** Die Korrektheit der entwickelten Lösung wird geprüft, indem überprüft wird, ob sie den Anforderungen entspricht.
- **Testen:** Umfassende Tests werden automatisiert und kontinuierlich durchgeführt, um Qualität, Funktionalität und Zuverlässigkeit der Lösung sicherzustellen. Ziel ist es, Fehler frühzeitig zu erkennen.
- **Veröffentlichen oder Bereitstellen:** Die entwickelte Lösung wird in der Produktionsumgebung bereitgestellt. Automatisierung wird eingesetzt, um Fehler zu minimieren und einen reibungslosen Übergang sicherzustellen.
- **Konfigurieren oder Betreiben:** In dieser Phase wird die bereitgestellte Software kontinuierlich konfiguriert und von den Betriebsteams in Bezug auf Ressourcenbereitstellung und Skalierung verwaltet – idealerweise automatisiert zur Laufzeit.
- **Überwachen:** Leistung, Verfügbarkeit und allgemeiner Zustand werden kontinuierlich überwacht, um Anomalien zu erkennen und die Weiterentwicklung der Software zu steuern.

Was ist der Nutzen

Die iterative und kollaborative Natur von DevOps hat die moderne Softwareentwicklung revolutioniert. Sie optimiert Prozesse und fördert eine Kultur der ständigen Integration und Bereitstellung. Dieser Ansatz fördert nicht nur die Zusammenarbeit, beschleunigt die Entwicklung und verbessert die Produktqualität, sondern etabliert auch eine kundenorientierte Denkweise im Entwicklungsprozess.

Im Zentrum des DevOps-Lebenszyklus stehen signifikante Vorteile [70, 7], insbesondere die Kostensenkung durch Automatisierung, Prozessoptimierung und Reduktion von Verschwendung. Gleichzeitig steigert DevOps die Kundenzufriedenheit, indem qualitativ hochwertige Software schnell und zuverlässig ausgeliefert wird. Darüber hinaus wirkt sich DevOps positiv auf die Mitarbeitermotivation aus, da ein kollaboratives und unterstützendes Arbeitsumfeld gefördert wird.

Neben Kostenersparnis und Kundenzufriedenheit bietet DevOps eine beschleunigte Softwarebereitstellung, höhere Softwarequalität und eine bessere Ausrichtung auf geschäftliche Ziele [71]. Forschung und Praxis unterstreichen das transformative Potenzial von DevOps und bestätigen seine zentrale Rolle in der modernen Softwareentwicklung durch die Steigerung der IT-Performance und Produktivität von Organisationen [70].

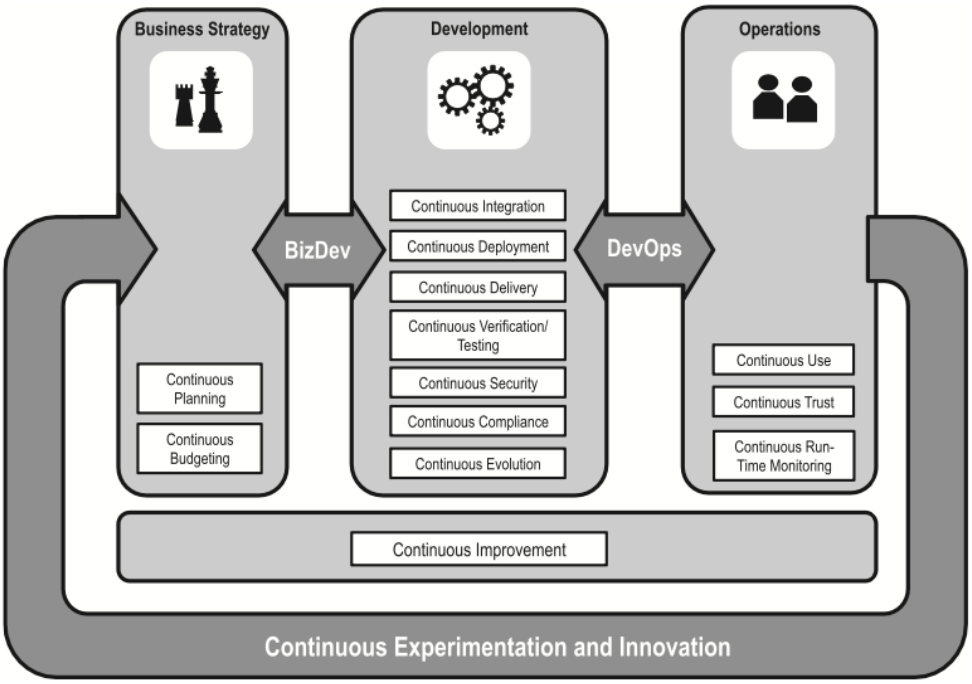


Abbildung 2.1: Continuous*-Karte [38]

2.1.2 Kontinuierliche Prozesse

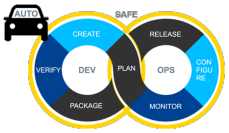
Kontinuierliche Prozesse sind eng mit DevOps verbunden und daher ein integraler Bestandteil des gesamten AutoDevSafeOps-Prozesses. Unter kontinuierlichen Prozessen in der Softwaretechnik versteht man eine Reihe von Aktivitäten, die eine häufige und automatisierte Integration von Codeänderungen, Tests und Bereitstellung ermöglichen. Allgemein bieten kontinuierliche Prozesse verschiedene Vorteile, darunter eine schnellere Markteinführung, verbesserte Softwarequalität und gesteigerte Zusammenarbeit.

Fitzgerald und Stol [38] untersuchten bestehende Initiativen im Bereich der kontinuierlichen Softwareentwicklung und führten den Begriff Continuous* als Oberbegriff für alle kontinuierlichen Aktivitäten während des gesamten Softwarelebenszyklus ein.

Im Rahmen unseres Projekts sind folgende Aktivitäten besonders relevant:

- **Continuous Integration (CI)** ist die Praxis, Codeänderungen regelmäßig in ein gemeinsames Repository zu integrieren. Entwickler übermitteln ihre Codeänderungen, die automatisch gebaut und getestet werden, wodurch sichergestellt wird, dass die Software stets in einem auslieferbaren Zustand bleibt. CI hilft bei der frühzeitigen Fehlererkennung, der Verbesserung der Codequalität und der Zusammenarbeit im Team.

- **Continuous Delivery & Deployment** zielen darauf ab, den Übergang von der Entwicklung in die Produktion zu automatisieren. Continuous Delivery stellt sicher, dass Änderungen kontinuierlich für die Bereitstellung vorbereitet werden, während Conti-



nuous Deployment noch einen Schritt weiter geht und die Freigabe vollständig automatisiert. Diese Praktiken ermöglichen schnellere, zuverlässigere Software-Releases und reduzieren menschliche Fehlerquellen.

- **Continuous Verification & Testing** umfassen die kontinuierliche Überprüfung der Software, um sicherzustellen, dass sie den geforderten Qualitäts- und Sicherheitsstandards entspricht. Dazu zählen automatisierte Tests, Sicherheitsprüfungen und die Validierung nach sicherheitskritischen Standards. Ziel ist es, Probleme frühzeitig zu identifizieren und die Zuverlässigkeit sicherzustellen.
- **Continuous Runtime Monitoring** bezeichnet die Echtzeitüberwachung und -analyse des Verhaltens der Software in ihrer Betriebsumgebung. Dadurch können unerwartete Probleme oder Abweichungen frühzeitig erkannt und behoben werden.

Die Anwendung kontinuierlicher Prozesse auf cyber-physische Systeme bringt besondere Herausforderungen hinsichtlich der Gewährleistung der Systemsicherheit mit sich. Sicherheitsbezogene Aktivitäten müssen kontinuierlich durchgeführt werden, um eine nahtlose Integration in Continuous Delivery & Deployment-Pipelines zu ermöglichen. Das Konzept der Continuous Safety Builds [86] fordert, dass Sicherheitsanalysen parallel zur Entwicklung erfolgen, Artefakte wie Quellcode behandelt werden und Sicherheitstests automatisiert in jeden Build integriert werden. Zeller [93] stellt eine Pipeline zur Continuous Safety Assessment vor und betont die Bedeutung der Automatisierung zur Beschleunigung der Sicherheitsnachweise. Meyers et al. [72] schlagen ein Sicherheits-Engineering-Framework vor, das den funktionalen Sicherheitsprozess optimiert. Die Autoren setzen auf formale Modellierung und eine vertraglich basierte Anforderungssprache, um ein höheres Maß an Automatisierung und Iteration sicherheitsrelevanter Aufgaben zu erreichen. Ähnlich präsentieren Munk und Schweizer [73] das Konzept von SafeOps, das kontinuierliche Prinzipien auf sicherheitsrelevante Aktivitäten im DevOps-Zyklus anwendet.

Bevor ein neues Update bereitgestellt wird, verlangen Regulierungsbehörden einen Nachweis, dass geeignete Sicherheitsmaßnahmen ergriffen wurden und kein unnötiges Risiko durch das Update entsteht. Daher sind Dokumentation und Nachvollziehbarkeit ein integraler Bestandteil des Entwicklungsprozesses. Dieser Schritt wird gemeinhin als Continuous Certification bezeichnet [8].

2.2 Verwandte Projekte

2.2.1 UP2Date

UP2DATE war ein EU-Horizon-2020-Projekt, das von 2020 bis 2023 mit sieben internationalen Partnern durchgeführt wurde. Es konzentrierte sich auf die Umsetzung von Mixed-Criticality in komplexen heterogenen Plattformen sowie Over-the-Air-Softwareaktualisierungen für Mixed-Criticality-Systeme unter Berücksichtigung von Sicherheits- und Schutzaspekten (SASE). Ein Ergebnis des Projekts war ein neues Software-Paradigma für SASE-Over-the-Air-Updates mit folgenden Zielen:

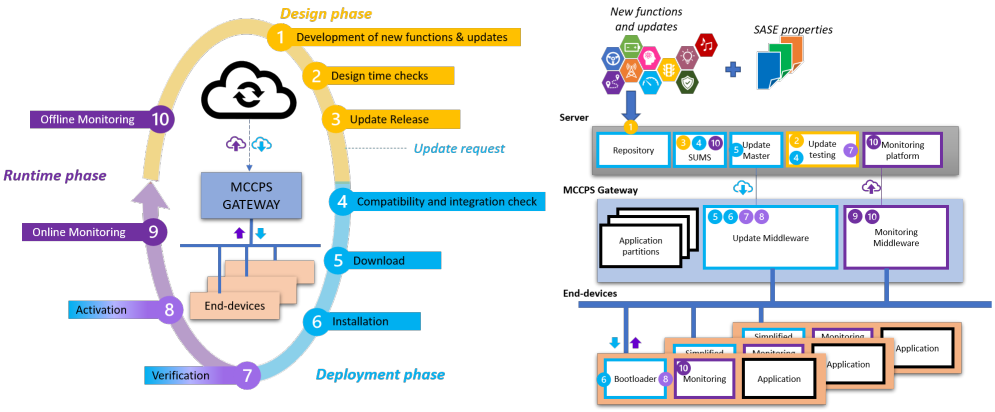


Abbildung 2.2: UP2DATE Update-Zyklus und Update-Architektur [3]

- Entwicklung von Strategien zur Bewältigung von SASE-Herausforderungen auf komplexen Hardwareplattformen
- Definition von SASE-Verträgen
- Ausarbeitung von Strategien für Beobachtbarkeit, Steuerbarkeit und Feedback
- Integration der UP2DATE-Softwarearchitektur
- Demonstration in zwei Anwendungsfällen: Automobil- und Bahnindustrie
- Bewertung der Zertifizierbarkeit der Sicherheits- und Schutzkonzepte

Das UP2DATE-Projekt veröffentlichte sein Konzept eines Update-Zyklus (Abbildung 2.2), der die Schritte eines Updates in Design-, Bereitstellungs- und Laufzeitphase darstellt [3]. Dieser Update-Zyklus wird auf eine Update-Architektur angewendet: Ein Server verarbeitet neue Funktionen und Updates, die mit SASE-Eigenschaften angereichert sind (z.B. Verwendung von Assume-Guarantee-Verträgen für Zeitverhalten von Funktionsereignissen) und verwaltet die automatisierte Integrationsprüfung betroffener Systeme. Das Update wird dann von einem Mixed-Criticality-Gateway empfangen, das es entweder auf interne Partitionen (d.h. auf demselben Steuergerät) oder auf externe Endgeräte (d.h. auf verschiedenen Steuergeräten) anwendet. Diese Interaktion wird durch Online-Monitore während der Bereitstellung und durch Offline-Monitore zur späteren Analyse überwacht.

Darüber hinaus diente die Definition einer modularen Softwarearchitektur für Mixed-Criticality-Systeme als Inspiration für die spezifischen Architekturen, die in diesem Bericht vorgeschlagen werden. Die Architektur war eines von drei UP2DATE-Themen, die im EU Innovation Radar aufgenommen wurden[1] als „UP2DATE architecture and middleware for safe and secure over the air software updates“ (OTASU).

[1] https://innovation-radar.ec.europa.eu/innovation/50049

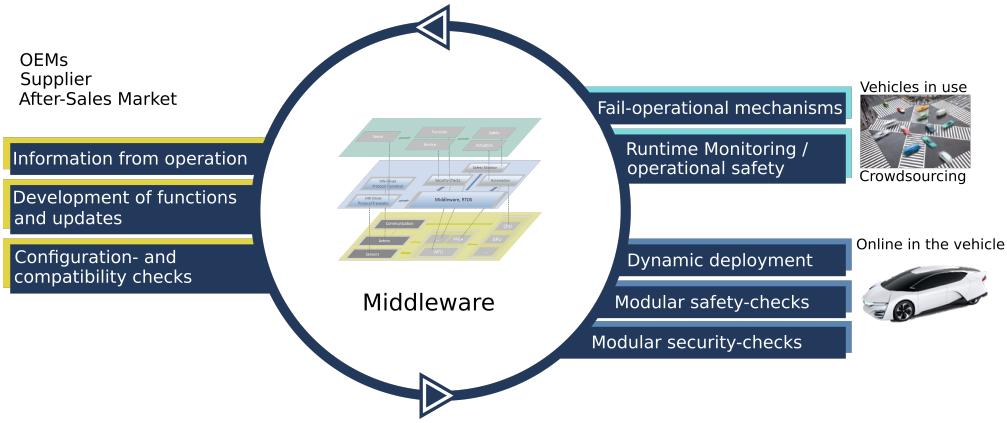


Abbildung 2.3: StepUp!CPS Update-Zyklus

2.2.2 StepUp!CPS

Das Step-Up!CPS-Projekt, gefördert durch das Bundesministerium für Bildung und Forschung (BMBF), entwickelte einen Prototyp-Prozess zur Durchführung sicherer und geschützter Softwareupdates. Der entwickelte Prozess basiert auf einem modellbasierten Entwicklungsprozess. Dieser wurde erweitert und an die Anforderungen eines iterativen Entwicklungs- und Testprozesses für modulare Updates angepasst. Unterstützt wird der Prozess durch eine Referenzarchitektur, die eine Vielzahl von Diensten für Bereitstellung, Verifikation und Überwachung von Updates bereitstellt.

Kernelemente des Prozessansatzes in Step-Up!CPS sind ein Entscheidungsprozess für Updates, ein Software- und Vertragsentwicklungsansatz, sichere Entwicklungsmethoden sowie Sicherheits- und Schutzmechanismen im Update-Gerät. Die grundlegenden Phasen des Update-Prozesses in Step-Up!CPS sind in Abbildung 2.3 dargestellt.

Der Prozess in Step-Up!CPS gliedert sich in sechs Hauptaktivitäten: Initiierung und Entwurf, Virtueller Integrationstest, Implementierung und Build, Variantenbewusster Softwaretest, Bereitstellung, Laufzeit und Überwachung. Der Prozess wurde auf Basis der Domänen Automobil, Industrie und maritime Systeme entwickelt. Der resultierende Prozess kann an die jeweilige Domäne angepasst werden, wodurch es möglich ist, den Update-Prozess spezifisch für den Anwendungsfall und die Anforderungen zu modellieren. Der Update-Prozess wird durch einen Trigger ausgelöst, der unterschiedlicher Art sein kann, z.B. regulatorische Änderungen, Probleme im Systemverhalten, Kompatibilitätsprobleme, Sicherheitsverletzungen oder Benutzeranforderungen. Auf Basis des Triggers wird das Update in eine von drei Update-Arten klassifiziert:

- **Korrektives Update:** Ein korrektives Update ist z.B. ein Bugfix. Diese verändern nicht die Benutzeroberfläche, sondern stellen die korrekte Funktionalität zur Erfüllung der Anforderungen wieder her. Der Trigger für ein korrektives Update stammt aus Berichten der eingesetzten Komponenten.
- **Perfektive Updates:** Perfektive Updates sollen die Leistung der Software verbessern.

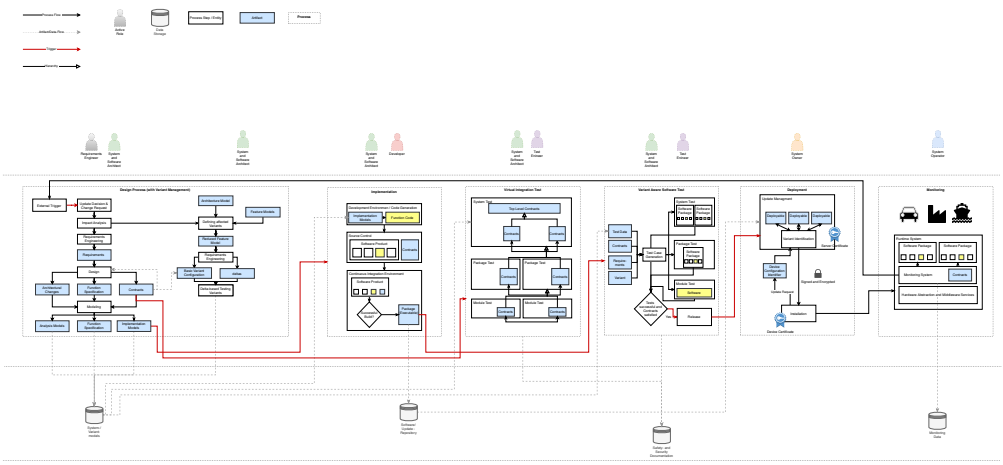


Abbildung 2.4: StepUp!CPS Gesamtprozess

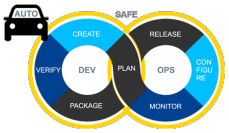
Dabei kann sich das interne Verhalten im Rahmen der Spezifikation ändern.

- **Adaptive Updates:** Adaptive Updates sind in erster Linie Erweiterungen, d.h. die Implementierung neuer Funktionalitäten. Diese erfordern eine Analyse neuer Anforderungen sowie eine Auswirkungsanalyse auf andere Komponenten oder das Gesamtsystem.

Nachfolgend eine kurze Übersicht über jede Phase des Prozesses. Eine detaillierte Beschreibung aller Phasen und Prozessschritte findet sich in [31]. Der Gesamtprozess von Step-Up!CPS ist in Abbildung 2.4 dargestellt.

Design-Prozess

Nachdem eine Entscheidung zur Durchführung eines Updates aufgrund eines Triggers getroffen wurde, erfolgt eine Auswirkungsanalyse der anstehenden Änderungen. Dabei wird ebenfalls untersucht, ob durch das Update einer Komponente auch andere Komponenten betroffen sind. Abhängig vom Trigger folgt eine Anforderungsanalyse der zu aktualisierenden Softwarekomponenten. Basierend auf den Auswirkungen des Updates wird eine Sicherheitsanalyse in Form einer HARA (Hazard and Risk Analysis) und einer TARA (Threat and Risk Analysis) durchgeführt. Der abschließende Schritt dieser Phase ist das eigentliche Systemdesign. Je nach Systemkonfiguration können unterschiedliche Systemvarianten entwickelt und ausgerollt werden. Ein entsprechendes Variantenmanagement auf Basis delta-basierter Verifikation ist Teil dieser Prozessphase und läuft parallel zum „normalen“ Prozess. Am Ende dieser Phase wird ein Softwarepaket zur Bereitstellung erzeugt.



Virtuelle Integration

Im Projekt Step-Up!CPS wurde auch die Contract-based Design-Methodik verwendet. Unter Anwendung dieser Methodik wird ein virtueller Integrationstest als zentraler Schritt in dieser Prozessphase durchgeführt, um sicherzustellen, dass die erstellten Verträge konsistent sind. Jede Komponente erhält bereits in der Entwurfsphase einen Vertrag, sodass in diesem Schritt über alle Hierarchien hinweg überprüft wird, ob das System als Ganzes konsistent ist und gemäß seiner Verträge verifiziert werden kann.

Implementierung

In der Implementierungsphase werden alle Schritte durchgeführt, um aus einer Spezifikation fertigen Code zu generieren, der in einem Repository abgelegt wird. Aus diesem Code werden ausführbare Pakete erstellt. Wird ein Update ausgelöst, findet in dieser Phase auch eine erneute Implementierung statt, damit das Update passend bereitgestellt werden kann.

Verifikation und Validierung

Die Verifikations- und Validierungsphase wurde eingeführt, um sicherzustellen, dass Updates ihren Anforderungen genügen. Dazu gehören Systemtests, Softwaretests und das Release-Management. Für die Softwaretests wurde die genutzte Contract-based Design-Methodik im Prozess berücksichtigt, und aufgrund der verschiedenen Varianten wurde ein variantenbewusster Softwaretest auf Basis eines digitalen Zwillings in den Prozess integriert.

Bereitstellung

In der Bereitstellungsphase werden zunächst Format und Inhalt der Update-Pakete definiert. Anschließend erfolgt die Übertragung der Pakete vom Server zu den jeweiligen Clients. Es wird zwischen verschiedenen Update-Szenarien unterschieden, etwa Pull- und Push-Szenarien. Hinsichtlich des Prozesses müssen in der Bereitstellungsphase auch Sicherheitsaspekte berücksichtigt werden. Verletzungen können insbesondere bei der Übertragung auftreten. Sicherheitsziele werden definiert. Installationen und Installationsprüfungen gehören ebenfalls zu dieser Phase. Falls Tests fehlschlagen, erfolgt ein Rollback zur vorherigen Version. All dies wird durch das Update-Management gesteuert.

Überwachung

Aufgrund der zunehmenden Komplexität cyber-physischer Systeme ist es nicht möglich, diese gegen alle denkbaren Situationen zu testen oder für jede mögliche Situation einen ausfallsicheren Modus zu identifizieren. Deshalb wurde die Online-Überwachungsphase im Prozess berücksichtigt. Dabei wurde zwischen Überwachung auf Funktionsebene und auf Systemebene unterschieden.

3 Softwareentwicklungszyklus: Der ADSO Prozess

Das MANNHEIM-AutoDevSafeOps-Projekt befasst sich mit der inkrementellen Entwicklung sicherheitskritischer Systeme. Dies umfasst die Erfassung von Daten zur Laufzeit, die Ableitung von Maßnahmen aus diesen Daten – wie etwa Mittel zur Verbesserung der Safety-Argumentation des Systems – sowie Aktualisierungen zur Laufzeit. Sowohl die Entwicklung als auch die Ausbringung von Updates werden dabei berücksichtigt. Der DevOps-Zyklus und seine einzelnen Phasen werden als generisches Mittel beschrieben, um solche Entwicklungsprozesse abzudecken. Eine detaillierte Diskussion zur Idee von DevOps findet sich in Abschnitt 2.1.1.

Da das Ziel des MANNHEIM-AutoDevSafeOps-Projekts die Entwicklung konkreter Lösungen (Technologiebausteine) entlang der einzelnen Phasen des DevOps-Zyklus ist, betrachten wir einen DevOps-Zyklus, der eine spezifische Umsetzung darstellt – den sogenannten ADSO-Prozess. Das Ziel der Definition dieses Prozesses ist es sicherzustellen, dass die einzelnen Beiträge der Projektpartner möglichst nahtlos zusammenpassen und einen konsistenten DevOps-Zyklus bilden. Der ADSO-Prozess gewährleistet zudem, dass alle Partnerbeiträge sinnvoll kombiniert und organisiert werden: Ein einheitlicher Prozess stellt sicher, dass diese Beiträge konsistent sind und dass ihre unterschiedlichen Methoden und Schnittstellen kompatibel sind und aufeinander aufbauen.

Das MANNHEIM-AutoDevSafeOps-Projekt ist entlang von vier verschiedenen Use Cases organisiert. Jeder Use Case umfasst verschiedene User Stories, die jeweils ein spezifisches Anwendungsszenario abdecken. Als zentrales Projektergebnis stellt jede User Story eine separate Instanz des ADSO-Prozesses dar, d.h. jede User Story eines Use Cases besteht aus Aktivitäten und Beiträgen der Partner, die einem oder mehreren zusammenhängenden Prozessschritten und Artefakten zugeordnet sind. Der ADSO-Prozess beschreibt somit in seiner Gesamtheit alle Partnerbeiträge über alle User Stories der einzelnen Use Cases hinweg und bringt sie in einen konsistenten und geordneten Zusammenhang. Durch diese Aktivitäten ermöglicht der Prozess die Identifikation von Lücken und Schwerpunkten im bearbeiteten DevOps-Zyklus. Dadurch wird ein Vergleich mit anderen Projekten möglich. Auf Basis dieses Vergleichs lassen sich Unterschiede, Ergänzungen und Gemeinsamkeiten zwischen verschiedenen Projekten und Forschungsfeldern identifizieren. Durch diesen Prozess können außerdem Beiträge zu Normen und Standards identifiziert werden – ein wichtiger Bereich bei der Aktualisierung cyber-physischer Systeme.

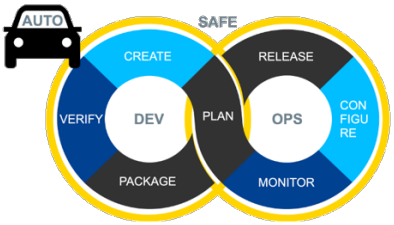


Abbildung 3.1: Das ADSO-Logo zeigt die DevOps-Phasen des ADSO-Prozesses

3.1 ADSO Process

Ein zentrales Ziel des Projekts war die Umsetzung von Entwicklungslösungen. Jede Lösung kann in einem oder mehreren Schritten des Entwicklungs- (DevOps-) Zyklus angewendet werden. Jeder Schritt nimmt bestehende Designartefakte als Eingabe und führt eine Reihe von Aktivitäten aus, um die erforderlichen Ausgabe-Artefakte zu erstellen.

Die von den Projektpartnern entwickelten Lösungen – Technologiebausteine – wurden den einzelnen Phasen des DevOps-Zyklus zugeordnet. Abbildung 3.1 zeigt den allgemeinen AutoDevSafeOps-Zyklus mit sieben Hauptphasen: Plan, Create, Verify, Package, Configure, Release und Monitor.

Nach der ersten Zuordnung von Technologiebausteine zu den Hauptphasen, wurden Technologiebausteine mit ähnlichen inhaltlichen Schwerpunkten gruppiert und in *Prozessschritten (PS)* zusammengefasst. Ein Prozessschritt stellt somit eine spezifische Menge von Aktivitäten dar, die ähnliche Ziele verfolgen. Mehrere Prozessschritte können zu einem Teilprozess oder einer Phase kombiniert werden. Alle Prozessschritte zusammen bilden den Gesamtprozess. Abbildung 3.2 zeigt eine Übersicht des Gesamtprozesses. Die Phasen sowie die zugehörigen Prozessschritte (PS) und Artefakte (A) werden in den folgenden Unterabschnitten kurz beschrieben.

Infolgedessen deckt der ADSO-Prozess alle im AutodevSafeOps-Projekt betrachteten Aktivitäten ab – von der Planung eines Updates über Modellierung, Verifikation, Implementierung und Laufzeitüberwachung bis hin zur Rückführung in die Entwicklungsphase mit den während der Laufzeit gesammelten Informationen. Der ADSO-Prozess ermöglicht damit die kontinuierliche Entwicklung, Verwaltung und Ausbringung von Updates im Betrieb. Je nach Anwednungsszenario, ist es möglich, einen konkreten Prozess zu instanziieren. Aus diesem Grund sind die einzelnen Prozessschritte nicht als verpflichtend anzusehen. Vielmehr bieten sie einen Rahmen und Leitpfade, die an die Bedürfnisse des jeweiligen Anwendungsszenarios angepasst werden können. In [30] sind die im Rahmen des Projekts betrachteten Use Cases, ihre User Stories sowie die dadurch jweilige Instanziierung des ADSO-Prozesses beschrieben.

3.1.1 Plan

Die *Plan*-Phase, siehe Abbildung 3.3, ist die Anfangsphase für die Entwicklung eines sicheren und zuverlässigen automobilen Systems. In dieser Phase finden Sicherheitsbewertungen und Anforderungsanalysen statt, um sicherzustellen, dass die nachfolgenden Phasen mit einem klaren Verständnis der Sicherheitsziele und erforderlichen Maßnahmen ablaufen. In einem

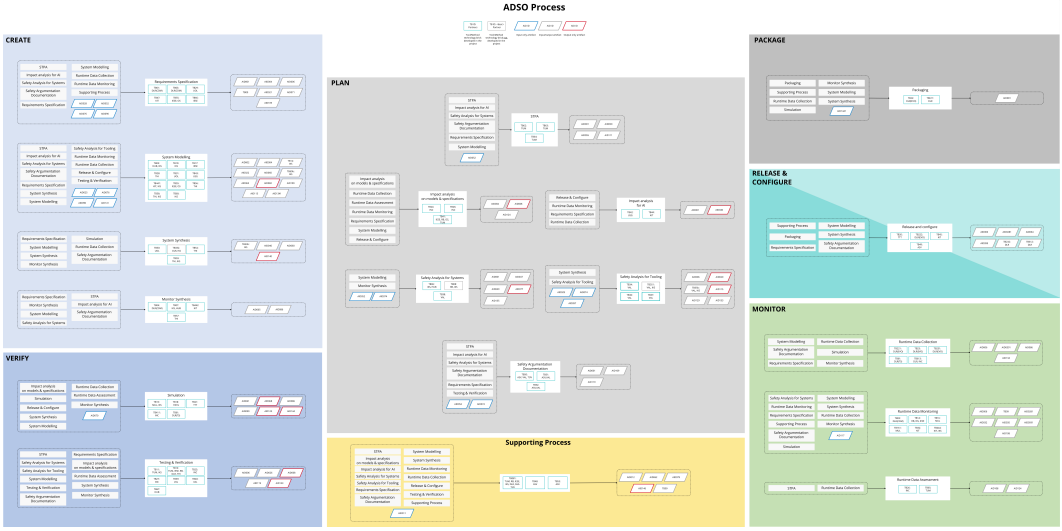


Abbildung 3.2: ADSO Gesamtprozess

DevOps-Zyklus bedeutet die Rückkehr in diese Phase, die Sicherheitsbewertung und Anforderungen mit dem aus dem Systembetrieb gewonnenen Wissen zu aktualisieren und ggf. notwendige Anpassungen einzuleiten. Folgende abstrakte Prozessschritte wurden aufgrund der entwickelten Technologiebausteine erstellt:

- **STPA (System-Theoretic Process Analysis):** STPA ist ein strukturiertes Verfahren zur Identifizierung und Analyse von Gefahren in sicherheitskritischen Systemen. Der Fokus liegt darauf zu verstehen, wie das System versagen kann, und so eine solide Grundlage für Sicherheitsanforderungen und Minderungsstrategien zu schaffen. Die folgenden Artefakte werden durch Aktivitäten innerhalb dieses Prozessschrittes erstellt:
  - Requirements (AID001)
  - Hazardous scenarios (AID050)
  - Leading Indicator Assumptions (AID055)
  - STPA Report (AID111)
- **Impact analysis on models & specifications** Der Prozessschritt *Impact Analysis on models & specifications* bewertet die Auswirkungen von Änderungen oder Ereignissen auf die Sicherheit, das funktionale und zeitliche Verhalten des Systems. Innerhalb dieses Schrittes kann analysiert werden, wie externe Auslöseereignisse oder Modifikationen die Systemsicherheit beeinflussen. Daher erhält dieser Schritt Eingaben aus anderen Phasen der Systementwicklung. Die Informationen können beispielsweise aus Simulationen, Analyseschritten des Systementwurfs oder aus Beobachtungen und Monitoren des laufenden Systems im Feld stammen. Die Impact Analysis analysiert zudem Änderungen am geplanten Systemdesign, um Probleme zu identifizieren wodurch das Gesamtverhalten des Systems auf Basis der gesammelten Informationen verbessert werden kann. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:

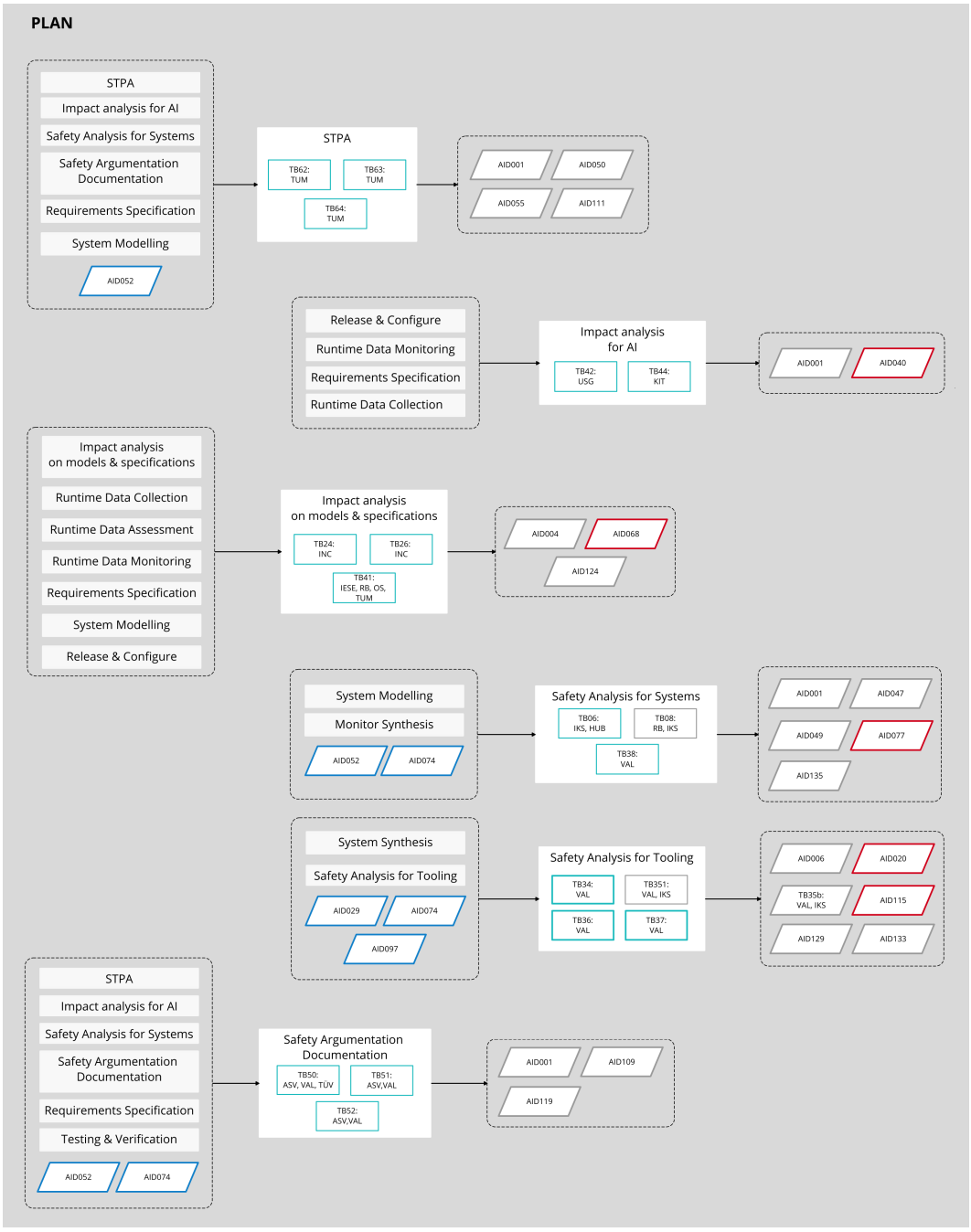
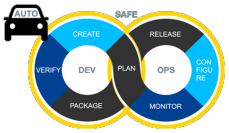
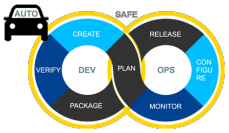


Abbildung 3.3: Planungsphase des ADSO-Prozesses

- DDI (AID004)
- Report timing behavior (AID068)
- Timing model (AID124)



- **Impact analysis for AI:** Der Prozessschritt Impact Analysis for AI beinhaltet explizit die Analyse der Auswirkungen von Data Drifts auf neuronale Netzmodelle sowie die Identifikation von Datensatzanforderungen. Er beinhaltet die zwei Technologiebausteinen: *Change Impact Analysis of Data Drift in Neural Network (NN) Model* und *Dataset Requirements Analysis*. Diese Bausteine zielen darauf ab, die Effekte von Data Drift auf die Modellleistung zu bewerten und die notwendigen Datensatzanforderungen für ein erneutes Training oder die Aktualisierung des Modells zu bestimmen. Der Baustein *Change Impact Analysis of Data Drift in Neural Network (NN) Model* analysiert die Auswirkungen von Data Drifts auf Modelle von neuronale Netzen. Dazu werden Bilddaten aus der Simulation Carla als Eingabe verwendet. Basierend auf diesen Eingaben liefert er einen Drift-Prozentsatz als Ausgabe. Der Baustein *Dataset Requirements Analysis* nutzt kritische Samples und ODD-Beschreibungen als Eingabe, um Datensatzanforderungen zu analysieren und Re-Trainings-Bedarf zu identifizieren. Er erzeugt dazu Datensatzanforderungen und -definitionen als Ausgabe. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Requirements (AID001)
  - Drift Percentage (AID040)
- **Safety Analysis for Systems:** Die Sicherheitsanalyse ist eine umfassende Untersuchung der Sicherheitsaspekte des Systems. Sie bezieht verschiedene Sicherheitsstandards ein, darunter ISO 26262 (Funktionale Sicherheit) [47] und ISO 21448 (Safety of the Intended Functionality) [49], um sicherzustellen, dass alle sicherheitsrelevanten Aspekte berücksichtigt werden. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Requirements (AID001)
  - GSN-based safety argumentation for NN (AID047)
  - HARA report (AID049)
  - Requirement-based Tests (AID077)
  - Triggering Conditions (AID135)
- **Safety Analysis for Tooling:** Um langfristig eine sichere Wartung und Aktualisierung eines sicherheitskritischen Systems zu gewährleisten, muss nicht nur das Produkt selbst, sondern auch die im Entwicklungsprozess verwendeten Tools abgesichert sein. Im Prozessschritt Safety Analysis for Tooling erweitern wir daher unsere Compliance-Argumentation um eine Integration von Sicherheitsaspekte für Software-Entwicklungstools und Toolchains. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Report (AID006)
  - Compliance Argumentation (AID020)
  - Safety Analysis (CZA) (TB35b)
  - TCA Models (AID115)
  - Manuals (AID129)



  - Toolchain TARA (AID133)

- **Safety Argumentation Documentation:** Die Integration regulatorischer Anforderungen ist ein entscheidender Aspekt, um Sicherheit, Schutz und Homologation in der Softwareentwicklung sicherzustellen. Dazu wird eine geeignete Dokumentation benötigt, die Anforderungen aus relevanten Vorschriften und Standards einbezieht. Dies umfasst die Dokumentation der Liste von Phasen und Artefakten, die zur Implementierung des Systems verwendet werden, einschließlich einer unstrukturierten Liste aller Artefakte und Anforderungen in den Bereichen Sicherheit, Schutz und Homologation. Zudem wird eine Spezifikation für die (Meta-)Daten definiert, die für diese Artefakte gesammelt werden. Durch die Aktivitäten in diesem Prozessschritt können Verantwortliche sicherstellen, dass alle notwendigen Anforderungen erfüllt werden, was letztlich in einer umfassenden Safety-Argumentation-Dokumentation resultiert. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Requirements (AID001)
  - Specification (AID109)
  - Test results (AID119)

3.1.2 Create

Die *Create*-Phase, siehe Abbildung 3.4, folgt unmittelbar auf die Plan-Phase und konkretisiert das Design eines Systems sowie des Updates auf Grundlage der vorangegangenen Plan-Phase. Sie ist daher eine wesentliche Phase bei der Entwicklung eines Updates. Sie besteht aus vier Prozessschritten: *Requirement Specification*, *System Modelling System Synthesis* und *Monitor Synthesis*.

- **Requirement Specification:** Basierend auf den aus den Prozessschritten der *Plan*-Phase resultierenden Sicherheitsanforderungen wird in diesem Prozessschritt sowohl das Update und das System spezifiziert. Diese Spezifikation umfasst Eigenschaften wie Timing, Funktionalität, sowie Safety. Im MANNHEIM-AutoDevSafeOps-Projekt werden sogennante Contracte verwendet, um das System und das Update in Bezug auf diese Eigenschaften zu spezifizieren. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Requirements (AID001)
  - DDI (AID004)
  - Report (AID006)
  - Safety Case (TB09)
  - Component Capabilities (AID021)
  - ODD (AID071)
  - Specification (AID109)

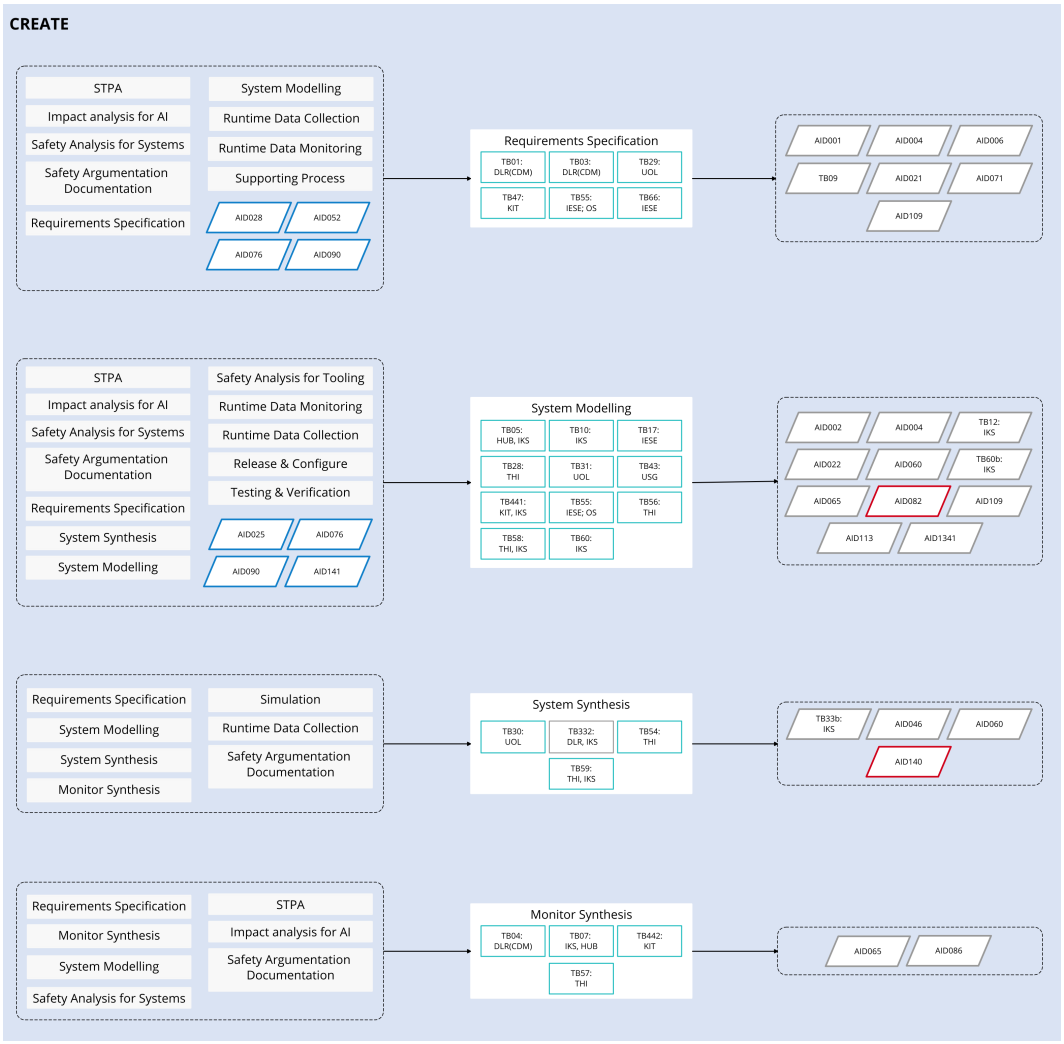
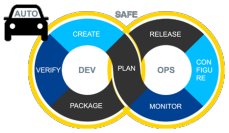


Abbildung 3.4: Create-Phase des ADSO-Prozesses

- **System Modelling:** In diesem Schritt wird das Systems modelliert, sodass es darauf basierend analysiert werden kann. Die Modelle können mathematisch, visuell oder simuliert sein und werden in Bereichen wie Ingenieurwesen, Softwareentwicklung und Wirtschaft eingesetzt. Teilweise wird später aus den entworfenen Modellen direkt ausführbarer Code synthetisiert. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Virtual vehicle (AID002)
  - DDI (AID004)
  - CZA - Contruction Zone Assist (TB12)
  - Component Modell (AID022)
  - Code (AID060)



  - AS Mode Manager (TB60b)
  - Neural Network Monitor (AID065)
  - ROS messages (AID082)
  - Specification (AID109)
  - System Architecture (AID113)
  - Training data set (AID1341)

- **System-Synthese:** In diesem Schritt werden diverse Artefakte systematisch erzeugt, unter Berücksichtigung der Spezifikation des Systems und des Updates. Dies umfasst z. B. eine Strategie, die den Rollout eines Updates garantiert, aber auch funktionaler Code. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Update (CZA) (TB33b)
  - Functional software (AID046)
  - Code (AID060)
  - Strategy (AID140)

- **Monitor-Synthese:** In der Monitor-Synthese werden Methoden und Werkzeuge zur Implementierung von Laufzeit-Monitoren entwickelt, einschließlich der Synthese von Montioren für interpretierbare neuronale Netze. Eingaben für diese Methoden sind u. a. Fähigkeitsspezifikationen, Trainingsdatensätze, Sicherheitsanforderungen und Vertragsspezifikationen. Die Ausgaben umfassen C-Code für Laufzeitmonitore, trainierte Modelle mit OOD-Monitoren und Werkzeuge zur Synthese von Laufzeitmonitoren. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Neural Network (AID065)
  - Runtime Monitors (AID086)

3.1.3 Verify

Traditionell umfasst die Entwicklung von Automobilsystemen ein strenges Verfahren mit Spezifikation, Zertifizierung und Validierung funktionaler und nicht-funktionaler Anforderungen. Mit zunehmender Komplexität der Fahrzeuge steigt jedoch auch die Anzahl der Anforderungen, was die Entwicklung sicherer Systeme zu einer Herausforderung macht. Dies hat zu einer Verlagerung von hardwareorientierten zu softwareorientierten Fahrzeugen geführt, bei denen IT-Innovationen moderne E/E-Architekturen unterstützen. Das ADSO-Projekt möchte zu dieser Digitalisierung beitragen, indem es Strategien für die Definition, Entwicklung, Prüfung, Validierung und Bereitstellung von Automobilsystemen entwickelt. So kann die Einführung von Digital Twins (DT) die Entwicklung, Prüfung und Bereitstellung neuer Softwarefunktionen beschleunigen. Diese Technologie ermöglicht eine probabilistische Modellbewertung ohne physische Prototypen und ist hoch skalierbar. In der Verifikationsphase unterteilen sich die Abreiten in die Prozesschritte *Simulation* und *Testing & Verification*

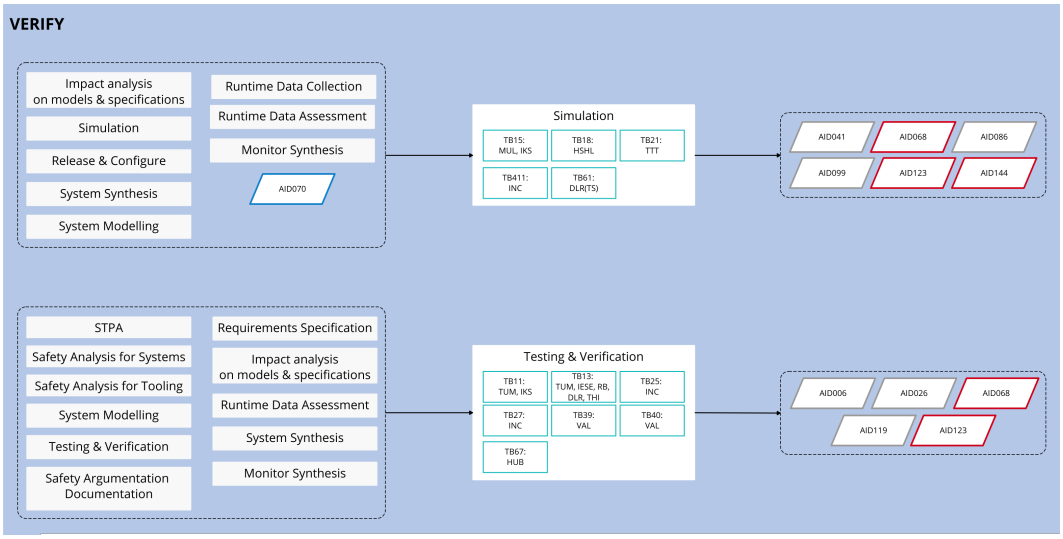


Abbildung 3.5: Verify-Phase des ADSO-Prozesses

- **Simulation:** In diesem Prozesschritt steht vor allem die Simulation des Systems im Fokus. Dabei wird zwischen einer Simulation des Systems und dem realen System in einer simulierten Umgebung unterschieden. Die Simulation liefert Ergebnisse, die in die verschiedenen Analysen der Planungsphase zurückfließen, und ermöglicht so eine Verkürzung der Tests ohne Ausbringung auf ein reales System. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Drifted time-series dataset (AID041)
  - Report timing behavior (AID068)
  - Runtime Monitors (AID086)
  - Shadow Mode (AID099)
  - Simulation Software (AID123)
  - VECU result (AID144)
- **Testing & Verification:** In diesem Schritt werden sowohl Softwaretests als auch formale Verifizierungstests durchgeführt. Mit diesen Tests kann überprüft werden, ob der zuvor generierte Code gemäß der Spezifikation korrekt ist, ob die Spezifikation selbst konsistent und korrekt ist und ob die Monitore korrekt entworfen wurden. Die Verifizierung wird durch verschiedene Techniken wie virtuelle Integrationstests oder Testfallgenerierung durchgeführt. In diesem Prozessschritt werden die folgenden Artefakt-Typen erstellt:
  - Report (AID006)
  - Test Scenarios (AID026)
  - Report timing behavior (AID068)

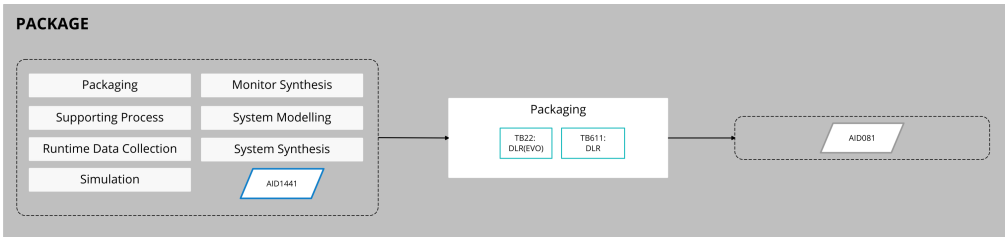


Abbildung 3.6: Package-Phase des ADSO-Prozesses

  - Test Results (AID119)
  - Simulation (AID123)

3.1.4 Package

Die Arbeiten in Package-Phase sind verantwortlich für die Vorbereitung des Updates zur Ausbringung auf das Zielsystem, indem dieser in eine geeignete Einheit gebündelt wird. Das Ergebnis ist ein konsistentes Artefakt, wie beispielsweise ein Docker-Image oder eine Archivdatei, das zuverlässig in unterschiedlichen Umgebungen ausgebracht werden kann.

Diese Phase unterstützt außerdem die Automatisierung innerhalb von CI/CD-Pipelines, indem versionierte Pakete erstellt werden, die in Artefakt-Repositories gespeichert werden können. Darüber hinaus spielt sie eine Rolle in den Bereichen Sicherheit und Compliance, indem Prüfungen auf Sicherheitslücken oder Lizenzprobleme durchgeführt werden. Insgesamt stellt die Package-Phase sicher, dass ein Update für eine konsistente, sichere und wiederholbare Ausbringung bereit ist.

- **Packaging:** Unter Berücksichtigung der erfolgreichen Verifikation, der synthetisierten Artefakte und vorherigen Simulationen erzeugt der Packaging-Schritt geeignete Softwarepakete für die Ausbringung des Updates. Dieses Paket muss alle Daten enthalten, die auf das Zielsystem übertragen werden müssen – einschließlich funktionaler Änderungen in Form eines Update-Binary, geänderter Spezifikationen sowie Metadaten für die Update-Ausführung.
  - Update Package (AID081)

3.1.5 Release and Configure

Die Release- und Configure-Phasen folgen auf die Package-Phase und sind entscheidend, um die Anwendung kontrolliert und zuverlässig in der realen Umgebungen bereitzustellen.

Die Release-Phase umfasst die Bereitstellung der verpackten Anwendung für die Ausbringung des Updates. Dazu gehören die Koordination des Deployment-Zeitpunkts, das Management von Genehmigungen oder Change-Management (falls erforderlich) sowie die Sicherstellung, dass die Zielumgebung sich in einem updatefähigen Zustand befindet. Ziel ist es, das getesteten Paket in einen Zustand zu überdühren, in dem es sicher und kontrolliert ausgebracht werden kann.

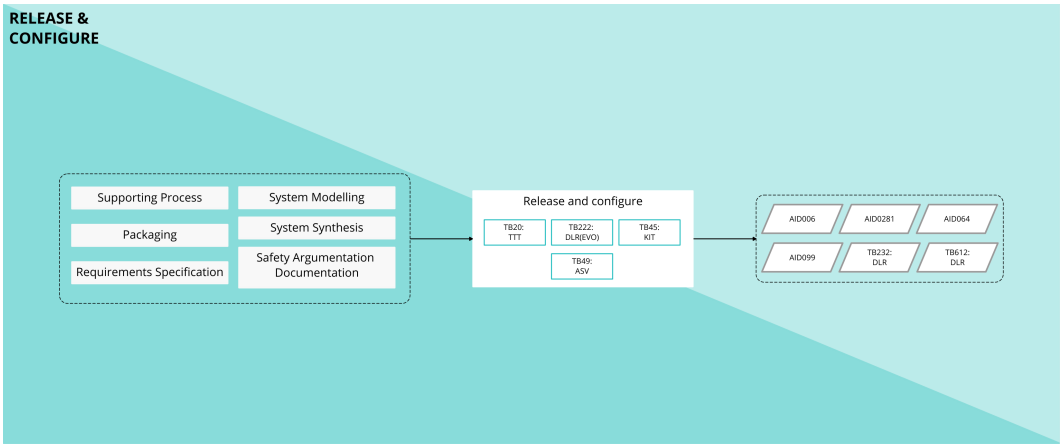


Abbildung 3.7: Configure- und Release-Phase des ADSO-Prozesses

Die Configure-Phase konzentriert sich auf die Einrichtung der Anwendung und ihrer umgebungsspezifischen Einstellungen nach der Bereitstellung des Updatepakets auf dem Zielsystem. Dazu zählen das Setzen von Umgebungsvariablen, das Konfigurieren von Datenbankverbindungen, das Aktualisieren von Konfigurationsdateien sowie die Integration mit anderen Systemdiensten. Die Konfiguration stellt sicher, dass sich die Anwendung in der Zielumgebung – ob Entwicklung, Staging oder Produktion – korrekt verhält.

Gemeinsam sorgen Release und Configure dafür, dass die Software nicht nur ausgeliefert, sondern auch korrekt eingerichtet und funktionsfähig in ihrer Zielumgebung betrieben wird. Damit stellen sie wesentliche Aktivitäten bereit, um ein reibungsloses und zuverlässiges Ausbringen innerhalb eines DevOps-Prozesses zu gewährleisten.

- **Release und Konfiguration:** Der Schritt Release und Konfiguration bringt das Update auf das Zielsystem und konfiguriert es gemäß der Update-Definition. Er übernimmt das Paket aus dem vorangegangenen Packaging-Schritt, entpackt es, führt (wie in den Metadaten definiert) notwendige lokale Verifikationsprozesse aus und installiert das Update. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Report (AID006)
  - Critical samples (AID0281)
  - Changes (AID064)
  - Shadow Mode (AID099)
  - Modular Embedded OSS Architecture (TB232)
  - ROS Translator (ROS1 Bridge deployment platform) (TB612)

3.1.6 Monitor

Im Kontext des Projekts spielt die Monitor-Phase, siehe Abbildung 3.9, eine zentrale Rolle bei der Sicherstellung der Zuverlässigkeit und Sicherheit des Systems über seinen gesamten

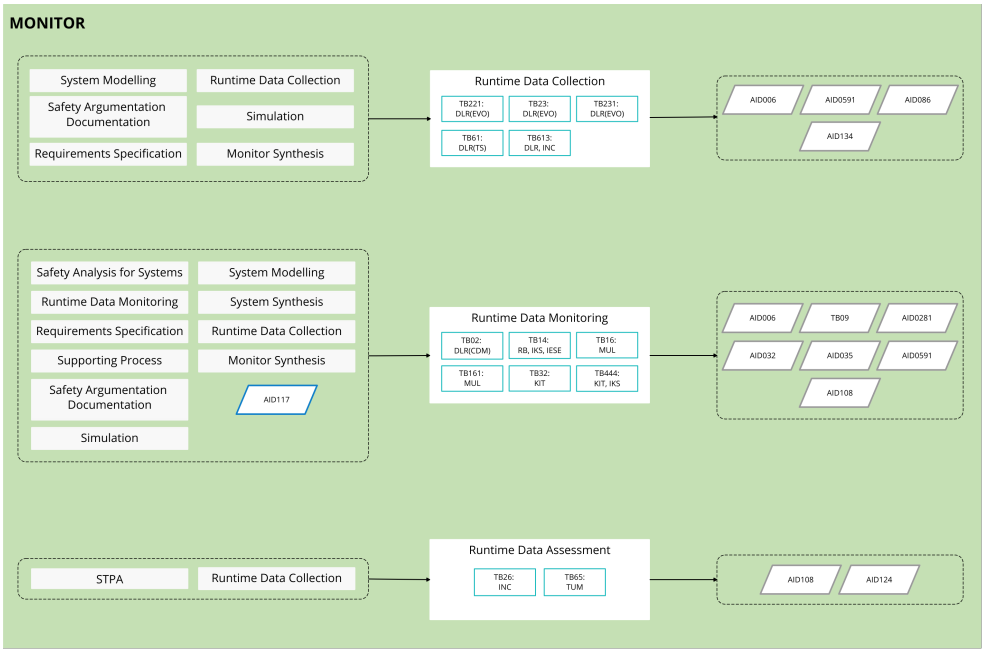


Abbildung 3.8: Monitor-Phase des ADSO-Prozesses

Betriebszeitraum hinweg. Monitoring im ADSO-Prozess ist eine kontinuierliche Phase, die in drei Hauptprozessschritte unterteilt ist: *Runtime Data Collection*, *Runtime Data Monitoring* und *Runtime Data Assessment*. Jeder dieser Prozessschritte wird durch spezifische Artefakte unterstützt, die zur Gesamtwirksamkeit der Monitor-Phase beitragen.

Die mit jedem Prozessschritt verbundenen Artefakte dienen den Stakeholdern als zentrale Werkzeuge, um die Systemintegrität zu wahren und Beteiligte zu schützen.

- **Runtime Data Collection:** Die Erfassung von Laufzeitdaten konzentriert sich auf die Aufnahme von Echtzeitdaten aus der Betriebsumgebung. Diese Daten werden zur Erstellung eines digitalen Zwillings des Systems und für weitere Simulationsprozesse genutzt, um die Entwicklung und das Testen von automobilen Systemen zu unterstützen. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - Report (AID006)
  - Logged Data (AID0591)
  - Runtime monitors (AID086)
  - Trace data (AID134)
- **Runtime Data Monitoring:** Beim *Runtime Data Monitoring* im ADSO-Prozess werden kontinuierlich Daten gesammelt und analysiert, die während des Betriebs des au-

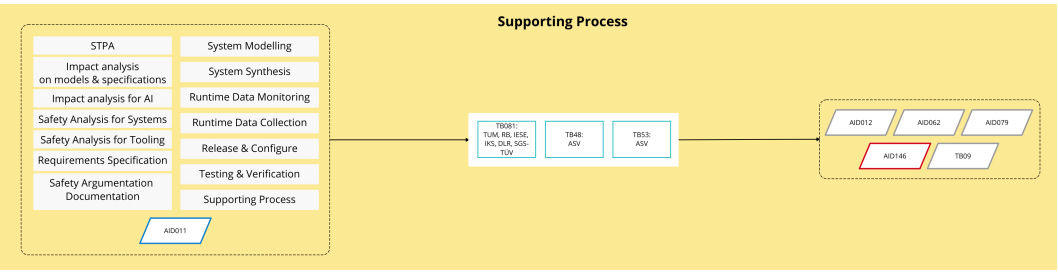


Abbildung 3.9: Supporting Process des ADSO-Prozesses

tomatisierten automobilen Systems in realen Szenarien entstehen. Ziel ist es, Abweichungen vom erwarteten Verhalten und potenzielle Sicherheitsrisiken durch geeignete Analysemechanismen zur Laufzeit zu erkennen und bereitzustellen. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt oder aktualisiert:

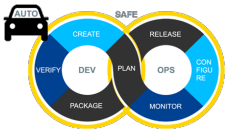
  - Report (AID006)
  - Critical samples (AID0281)
  - Dashboard output (AID032)
  - Data drift detection component (AID035)
  - Logged Data (AID0591)
  - SPI Violations (AID108)

- **Runtime Data Assessment:** Dieser Schritt ermöglicht es, Daten zu speichern, vorab zu filtern und anschließend an die aufbauende Analysen zu übermitteln. Ein solcher Schritt ist aufgrund der begrenzten Verbindungsverfügbarkeit und des begrenzten Durchsatzes erforderlich, da das Fahrzeug zur Operationszeit möglicherweise nur begrenzt Daten übertragen kann, während es in Ruhephasen eine erhöhte Übertragungsrade aufweisen kann. Die Daten könnten (im logischen Sinne, aber auch auf technischer Ebene mithilfe von Komprimierung) gebündelt und je nach Datentyp an unterschiedliche Nutzer übermittelt werden. Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt:
  - SPI Violations (AID108)
  - Timing Model (AID124)

3.1.7 Supporting Process

Der Supporting Process spielt eine unterstützende Rolle bei der gesamten Entwicklung und Ausbringung von Systemen, mit Fokus auf Sicherheit, Schutz und Datenintegrität. Er besteht aus drei Technologiebausteinen: *Safety Argumentation methodology for Operational safety*, *Distributed Ledger Technology* und einem *Integrity Dashboard*.

Die *Safety Argumentation methodology for Operational safety* erstellt durch Analyse und Adressierung von Unsicherheiten ein Safety Case. Es werden Eingabeartefakte wie z. B. HARA-Berichte und STPA-Berichte genutzt, um einen Safety Case in Form eines GSN-Graph zu erzeugen.



Die *Distributed Ledger Technology* und das *Integrity Dashboard* sind Tools, die Blockchain-Netzwerke und Backend-Servermodule nutzen, um Software- und Artefakt-Metadaten zu verwalten und zu visualisieren, mit dem Ziel, Entwicklungsprozesse durch sichere Datenspeicherung und -abfrage zu unterstützen.

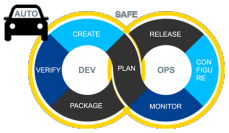
Die folgenden Artefakttypen werden in diesem Prozessschritt erstellt oder aktualisiert:

- Blockchain network with smart contract (AID012)
- Metadata (AID062)
- REST APIs (AID079)
- Visualisations of the data in the blockchain (AID146)

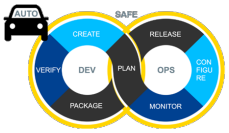
3.2 Technologiebausteine und Artefakte

Technologiebausteine Die folgende Tabelle führt alle Technologiebausteine auf, die von den Partnern im Rahmen des Projekts erstellt wurden. Die Tabelle enthält eine kurze Beschreibung, die Input- und Output-Artefakte, sowie die Prozessschritte, in denen der Technologiebaustein genutzt wird. Sie enthält zudem eine Liste von Referenzen, in denen die Technologiebausteine detailliert beschrieben werden.

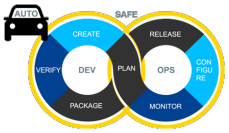
ID	Technology Brick	Input Artefakt(e)	Output Artefakt(e)	Partner	ADSO Process Step(s)	Referenzen
TB01	Definition of component capabilites	AID135, AID001	AID001, AID071, AID021	DLR	Requirements Specification	[D3.2:Sect. 4.1.1] [D4.2:Sect. 4.2]
TB02	MTCQ Language & Answering Tool	AID117, AID135	AID006	DLR	Runtime Data Monitoring	[D3.2:Sect. 8.1.2] [D4.2:Sect. 7.1.1]
TB03	Validation of component capabilities	AID049, AID021	AID006	DLR	Requirements Specification	[D4.2:Sect. 6.1]
TB04	Code generation for component capabilities for implementing monitors	AID021		DLR	Monitor Synthesis	[D4.2:Sect. 7.2.1]



TB05	Modular Neural Networks	AID006, AID071, AID025	AID065	HUB, IKS, THI	System Modelling	[D3.2:Sect. 6.1]
TB06	Development of improved safety argumentation	AID065	AID047	IKS, HUB, THI	Safety Analysis for System	[D3.2:Sect. 4.5]
TB07	Definition of runtime metrics for interpretable NN	AID065, AID047	AID086	IKS, HUB, THI	Monitor Synthesis	[D3.2:Sect. 8.2]
TB08	Performing HARA	AID052, AID113, TB012	AID049, AID001, AID135	RB, IKS	Safety Analysis for System	[D4.2:Sect. 4.6.2] [D4.2:Sect. 4.6.2]
TB081	Safety Argumentation methodology for Operational safety	AID006, AID049, AID111, TB12, AID035, AID047, AID050	TB09	TUM, RB, IESE, IKS, DLR, SGS-TÜV	Supporting Process	[D3.2:Sect. 4.4]
TB09	Safety Case			TUM, RB, IESE, IKS, DLR, SGS-TÜV	Supporting Process, Requirements Specification, Runtime Data Monitoring	[D3.2:Sect. 4]
TB10	Integration Platform	AID060, AID113	AID002	IKS	System Modelling	[D4.2:Sect. 2.3.3]
TB11	Test scenario generation	AID049, AID111, TB12 , AID050, AID113	AID026	TUM, IKS	Testing & Verification	[D4.2:Sect. 6.4]
TB12	CZA - Contruction Zone Assist		TB12	IKS	System Modelling	[D4.2:Sect. 4.6]

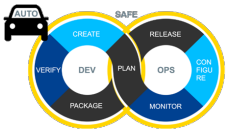


TB13	Simulation + Data Aquisition	AID026, TB12	AID119, AID123	IKS,TUM, RB, DLR, THI	Testing & Verification	[D4.2:Sect. 6.5]
TB14	Dashboard + Data analysis	TB09	AID032,TB09, AID108	RB, IKS, IESE	Runtime Data Monitoring	[D4.2:Sect. 7.4.1]
TB15	Data Drift Injection	AID070	AID041	MUL, IKS	Simulation	[D4.2:Sect. 7.3.2]
TB16	Data Drift Detection	AID041	AID035	MUL	Runtime Data Monitoring	[D4.2:Sect. 7.3.2]
TB161	Data Drift Analytics	AID035	AID032	MUL	Runtime Data Monitoring	[D4.2:Sect. 7.3.2]
TB17	DSL for Models	AID141, AID046	AID060	IESE	System Modelling	[D3.2:Sect. 8.4.1] [D3.2:Sect. 8.4.1] [D4.2:Sect. 4.3]
TB18	QEMU Interfaces	AID060	AID123	HSHL	Simulation	[D4.2:Sect. 6.6] [D5.2:Sect. 4.3]
TB20	Update Manager	AID062	AID099	TTT	Release and configure	[D3.2:Sect. 7.4]
TB21	vECU	AID099	AID099, AID144, AID026	TTT	Simulation	[D5.2:Sect. 4.2]
TB22	Update Packager	AID081, AID062, AID1441	AID081	DLR	Packaging	[D5.2:Sect. 3.2]
TB221	Monitoring Middleware	AID1341	AID0591, AID006	DLR	Runtime Data Collection	[D5.2:Sect. 5.1]
TB222	Update Middleware	AID081	AID064	DLR	Release and configure	[D5.2:Sect. 3.2]
TB23	Timing Observer	AID109	AID134	DLR	Runtime Data Collection	[D4.2:Sect. 7.2.3]
TB231	Resource Observer	AID109	AID134	DLR	Runtime Data Collection	[D5.2:Sect. 5.1]
TB232	Modular Embedded OSS Architecture			DLR	Release and configure	[D5.2:Sect. 3.1]

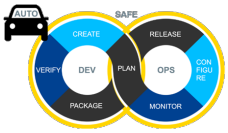


TB24	Impact Analysis / Timing Metrics	AID134, AID124	AID068,	INC	Impact Analysis on models & specifications	[D4.2:Sect. 6.7]
TB25	Contract-based Design	AID109, AID124	AID068	INC	Testing & Verification	[D4.2:Sect. 7.1.3] [D4.2:Sect. 7.1.3] [D4.2:Sect. 7.1.3]
TB26	Derive models from Traces	AID134	AID124	INC	Impact Analysis on models & specifications, Runtime Data Assessment	[D4.2:Sect. 7.3.5]
TB27	VIT	AID124	AID068	INC	Testing & Verification	[D4.2:Sect. 2.3.5]
TB28	Code generation	AID060	AID060	THI	System Modelling	[D5.2:Sect. 3.4.1]
TB29	Specification of mode management	AID022, AID052	AID109	UOL	Requirements Specification	[D3.2:Sect. 5.3]
TB30	Synthesis of (winning strategy)/Plan	AID109	AID140	UOL	System Synthesis	[D3.2:Sect. 7.3]
TB31	Development Update View	AID022	AID022, AID109	UOL	System Modelling	[D3.2:Sect. 7.1]
TB32	Adaptive Data Collector	AID046, AID086, AID109	AID0591	KIT	Runtime Data Monitoring	[D2.2:Sect. 6.6.2]
TB33b	Update Software for Vehicle Function (CZA) on-demand			IKS, DLR	System Synthesis	[D4.2:Sect. 6.3]
TB332	ADORE CI/CD	AID002	AID046, TB33b	DLR, IKS	System Synthesis	[D4.2:Sect. 2.3.1]

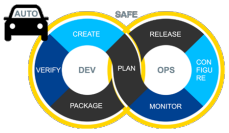
TB34	Modelling of a Safety and Security argumentation for the safe (and secure) usage of tools and libraries	AID074	AID020	VAL	Safety Analyses for Tooling	
TB35b	Safety Analysis of vehicle function (CZA) safety related activities			VAL, IKS	Safety Analyses for Tooling	
TB351	Tool Chain Analyzer	TB33b	TB35b	VAL, IKS	Safety Analyses for Tooling	
TB36	Security analysis for Toolchains (and Libraries)	AID129, AID097, AID029	AID133, AID006	VAL	Safety Analyses for Tooling	[D4.2:Sect. 8.6]
TB37	Database for Tool Classification and Qualification	AID129	AID115, AID129	VAL	Safety Analyses for Tooling	[D4.2:Sect. 8.2]
TB38	Database for Library Qualification	AID074	AID077	VAL	Safety Analysis for System	[D4.2:Sect. 8.1]
TB39	Safe Usage Checker for ToolChains	AID129	AID006	VAL	Testing & Verification	[D4.2:Sect. 8.5]
TB40	Safe Usage Checker for Libraries	AID060	AID006	VAL	Testing & Verification	[D4.2:Sect. 8.4]



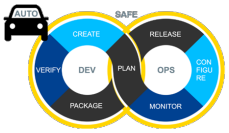
TB41	Change impact analysis	AID004, AID108, AID064	AID004	IESE, RB, OS, TUM	Impact Analysis on models & specifications	[D4.2:Sect. 5.1]
TB411	Co-Simulation	AID124, AID046	AID068, AID123	INC	Simulation	[D5.2:Sect. 4.1]
TB42	Change impact analysis of Data Drift in NN model	AID134	AID040	USG	Impact Analysis for AI	[D4.2:Sect. 4.9]
TB43	Integration of development artifacts into a formal model using different modeling tools	AID134	AID065	USG	System Modelling	[D4.2:Sect. 4.9]
TB44	Dataset requirements analysis	AID071, AID0281	AID001	KIT	Impact Analysis for AI	[D3.2:Sect. 6.3.1]
TB441	Assured Dataset Creation	AID001, AID134, AID0281	AID1341	KIT, IKS	System Modelling	[D3.2:Sect. 6.3.2]
TB442	Neural Network Architecture with XAI-based OOD Detection	AID1341	AID086, AID065	KIT	Monitor Synthesis	[D2.2:Sect. 6.6.3], [D3.2:Sect. 6.2]
TB444	Safeguard for Neural Networks	AID086, AID134	AID0281	KIT	Runtime Data Monitoring	[D3.2:Sect. 8.3]
TB45	Shadow mode release	AID046, AID081, AID109	AID0281	KIT	Release and configure	[D2.2:Sect. 6.6.1]
TB47	Contract Miner	AID028, AID109, AID134	AID109	KIT	Requirements Specification	[D3.2:Sect. 4.3.1] [D4.2:Sect. 4.4]



TB48	Distributed Ledger Technology	AID109, AID011, AID012, AID062	AID012, AID079, AID062	ASV	Supporting Process	[D4.2:Sect. 8.3.3]
TB49	Integrity Monitoring Toolset	AID081, AID062	AID006	ASV	Release and configure	[D5.2:Sect. 5.4]
TB50	Regulatory Requirements Integration	AID074	AID001	ASV,VAL, TÜV	Safety Argumentation Documentation	[D4.2:Sect. 8.3.1]
TB51	Documentation of the List of Stages/ Artifacts used to implement the UseCase	AID001, AID052	AID119	ASV,VAL	Safety Argumentation Documentation	[D4.2:Sect. 8.3]
TB52	Artifacts (meta)data specification	AID119	AID109	ASV, VAL	Safety Argumentation Documentation	[D4.2:Sect. 8.3.2]
TB53	Integrity Dashboard	AID133, AID006, AID012, AID079, AID109, AID026, AID119, AID032, AID060, AID0591, AID064, AID004, AID111, AID049, AID001	AID146	ASV	Supporting Process	[D4.2:Sect. 7.4.3]
TB54	Optimization of parallelism	AID060	AID060	THI	System Synthesis	[D5.2:Sect. 3.4.2]



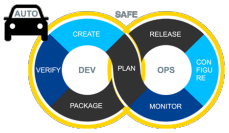
TB55	DDI Tool-adapter - Model integrator	AID090, AID076	AID004	IESE, OS	Requirements Specification, System Modelling	[D4.2:Sect. 4.5]
TB56	Development of concepts of automotive SoA specification for innovative functions and learned/learning behaviors		AID109	THI	System Modelling	[D3.2:Sect. 5.1]
TB57	Contract Based Runtime Monitoring Specification and Synthesis	AID001, AID109	AID086	THI	Monitor Synthesis	[D3.2:Sect. 8.1.1] [D4.2:Sect. 7.1.2]
TB58	Specification of resilient architectures		AID113	THI, IKS	System Modelling	[D2.2:Sect. 5.3]
TB59	Research on methods for the synthesis of adaptation strategies.	AID113, AID086	AID140	THI, IKS	System Synthesis	[D2.2:Sect. 5.5.3], [D4.2:Sect. 7.3.3]
TB60	AS Mode Manager	AID002, AID071	AID082, AID060, TB60b	IKS	System Modelling	[D3.2:Sect. 5.2]
TB60b	AS Mode Manager			IKS	System Modelling	[D4.2:Sect. 4.7]
TB61	ROS Based telemetry System	AID086	AID086	DLR	Simulation, Runtime Data Collection	[D5.2:Sect. 3.3]
TB611	ROS2 Packager	AID086, AID060	AID081	DLR	Packaging	[D5.2:Sect. 3.3]



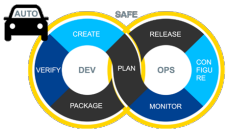
TB612	ROS Translator(ROS1_Bridge deployment platform)			DLR	Release and configure	[D5.2:Sect. 3.3]
TB613	ROS Observer	AID086	AID134	DLR, INCHRON	Runtime Data Collection	[D5.2:Sect. 3.3]
TB62	Generation of Leading Indicators	AID111	AID055	TUM	STPA	[D3.2:Sect. 4.2] [D4.2:Sect. 5.2.2]
TB63	STPA	AID052, AID113, AID071, AID001	AID111, AID001	TUM	STPA	[D4.2:Sect. 5.2]
TB64	Specification of Hazardous Scenarios	AID111	AID050	TUM	STPA	[D4.2:Sect. 6.4]
TB65	Monitoring of Leading Indicators	AID055	AID108	TUM	Runtime Data Assessment	[D4.2:Sect. 7.4.2] [D4.2:Sect. 7.4.2]
TB66	SPI definition in context of a safety case	TB09	TB09	IESE	Requirements Specification	[D3.2:Sect. 4.4.4] [D4.2:Sect. 4.8]
TB67	Robustness Testing of Neural Networks	AID065, AID109	AID119	HUB	Testing & Verification	[D4.2:Sect. 6.2]

Artefakte Die nachfolgende Tabelle führt alle Artefakte auf, die als Input, Output, oder beides von einem oder mehreren Technologiebausteinen genutzt wird. Die Artefakte sind dabei abstrakt gehalten, und geben daher in der Regel die Rolle anstatt eines konkreten Datenformats an.

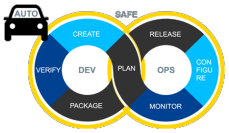
ID	Artefakt	Input von Prozessschritt(en)	Output von Prozessschritt(en)
AID001	Requirements	STPA, Safety Argumentation Documentation, Requirements Specification, System Modelling, Monitor Synthesis, Supporting Process	Impact Analysis for AI, STPA, Safety Analysis for System, Safety Argumentation Documentation, Requirements Specification



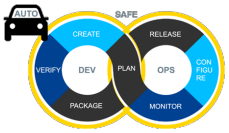
AID002	Virtual vehicle	System Modelling, System Synthesis	System Modelling
AID004	DDI	Impact Analysis on models & specifications, Supporting Process	Impact Analysis on models & specifications, System Modelling, Requirements Specification
AID006	Report	System Modelling, Supporting Process	Safety Analyses for Tooling, Runtime Data Monitoring, Runtime Data Collection, Release and configure, Testing & Verification, Requirements Specification
AID011	Blockchain network configurations	Supporting Process	
AID012	Blockchain network with smart contract	Supporting Process	Supporting Process
AID020	Compliance argumentation		Safety Analyses for Tooling
AID021	Component Capabilities	Requirements Specification, Monitor Synthesis	Requirements Specification
AID022	Component Modell	Requirements Specification, System Modelling	System Modelling
AID025	Concepts from domain experts	System Modelling	
AID026	Test Scenarios	Testing & Verification, Supporting Process	Testing & Verification
AID028	Contract template	Requirements Specification	
AID0281	Critical samples	Impact Analysis for AI, System Modelling	Runtime Data Monitoring, Release and configure
AID029	CVEs	Safety Analyses for Tooling	
AID032	Dashboard output	Supporting Process	Runtime Data Monitoring



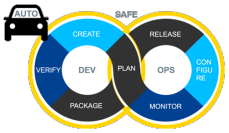
AID035	Data drift detection component	Runtime Data Monitoring, Supporting Process	Runtime Data Monitoring
AID040	Drift Percentage		Impact Analysis for AI
AID041	Drifted time-series dataset	Runtime Data Monitoring	Simulation
AID046	Functional software	System Modelling, Simulation, Release and configure, Runtime Data Monitoring	System Synthesis
AID047	GSN-based safety argumentationfor NN	Monitor Synthesis, Supporting Process	Safety Analysis for System
AID049	HARA report	Requirements Specification, Testing & Verification, Supporting Process	Safety Analysis for System
AID050	Hazardous scenarios	Testing & Verification, Supporting Process	STPA
AID052	Informal description	STPA, Safety Analysis for System, Safety Argumentation Documentation, Requirements Specification	
AID055	Leading Indicator Assumptions	Runtime Data Assessment	STPA
AID0591	Logged Data	Supporting Process	Runtime Data Monitoring, Runtime Data Collection
AID060	Code	System Modelling, System Synthesis, Testing & Verification, Simulation, Packaging, Supporting Process	System Synthesis, System Modelling
AID062	Metadata	Packaging, Release and configure, Supporting Process	Supporting Process
AID064	Changes	Impact Analysis on models & specifications, Supporting Process	Release and configure



AID065	Neural Network	Safety Analysis for System, Monitor Synthesis, Testing & Verification	Monitor Synthesis, System Modelling
AID068	Report Timing behavior		Impact Analysis on models & specifications, Testing & Verification, Simulation
AID070	Normal/Unchanged Time-series dataset	Simulation	
AID071	ODD	STPA, Impact Analysis for AI, System Modelling	Requirements Specification
AID074	Regulations and standards	Safety Analysis for System, Safety Analyses for Tooling, Safety Argumentation Documentation	
AID076	ReqSuite model elements	System Modelling, Requirements Specification	
AID077	Requirement-based Tests		Safety Analysis for System
AID079	REST APIs	Supporting Process	Supporting Process
AID081	Update Package	Packaging, Release and configure	Packaging
AID082	ROS messages		System Modelling
AID086	Runtime monitors	Simulation, Packaging, Runtime Data Collection, Runtime Data Monitoring, System Synthesis	Runtime Data Collection, Simulation, Monitor Synthesis
AID090	safeTbox model elements	System Modelling, Requirements Specification	
AID097	SBOM	Safety Analyses for Tooling	
AID099	Shadow Mode	Simulation	Release and configure, Simulation
AID108	SPI Violations	Impact Analysis on models & specifications	Runtime Data Assessment, Runtime Data Monitoring



AID109	Specification	Requirements Specification, Monitor Synthesis, System Synthesis, Testing & Verification, Release and configure, Runtime Data Collection, Runtime Data Monitoring, Supporting Process	Safety Argumentation Documentation, System Modelling, Requirements Specification
AID111	STPA Report	STPA, Testing & Verification, Supporting Process	STPA
AID113	System Architecture	STPA, Safety Analysis for System, System Synthesis, Testing & Verification	System Modelling
AID115	TCA Models		Safety Analyses for Tooling
AID117	Temporal query and traffic data	Runtime Data Monitoring	
AID119	Test results	Safety Argumentation Documentation, Supporting Process	Safety Argumentation Documentation, Testing & Verification
AID123	Simulation		Testing & Verification, Simulation
AID124	Timing Model	Impact Analysis on models & specifications, Testing & Verification, Simulation	Impact Analysis on models & specifications, Runtime Data Assessment
AID129	Manuals	Safety Analyses for Tooling, Testing & Verification	Safety Analyses for Tooling
AID133	Toolchain TARA	Supporting Process	Safety Analyses for Tooling
AID134	Trace data	Impact Analysis on models & specifications, Impact Analysis for AI, Requirements Specification, System Modelling, Runtime Data Monitoring, Runtime Data Assessment	Runtime Data Collection



AID1341	Training data set	Monitor Synthesis, Runtime Data Collection	System Modelling
AID135	Triggering Conditions	Requirements Specification, Runtime Data Monitoring	Safety Analysis for System
AID140	Strategy		System Synthesis
AID141	vECU templates	System Modelling	
AID144	VECU result		Simulation
AID1441	Verfication Results	Packaging	
AID146	Visualisations of the data in the blockchain		Supporting Process
TB09	Safety Case	Requirements Specification, Runtime Data Monitoring	Runtime Data Monitoring, Requirements Specification, Supporting Process
TB12	CZA - Contruction Zone Assist	Safety Analysis for System, Testing & Verification, Supporting Process	System Modelling
TB33b	Update (CZA)	Safety Analyses for Tooling	System Synthesis
TB35b	Safety Analysis (CZA)		Safety Analyses for Tooling
TB60b	AS Mode Manager		System Modelling
TB612	ROS Translator(ROS1_Bridge deployment platform)		Release and configure
TB232	Modular Embedded OSS Architecture		Release and configure

4 Projektergebnisse und Lösungen

4.1 Integration sicherheitsrelevanter Aktivitäten in den DevOps-Prozess

Die Integration sicherheitsrelevanter Aktivitäten in den DevOps-Lebenszyklus stellt einen grundlegenden Wandel von traditionellen Sicherheitsansätzen hin zu einer kontinuierlichen Sicherheitsgewährleistung dar. Da sich Fahrzeugsysteme zunehmend zu softwaredefinierten Fahrzeugen mit immer komplexeren automatisierten Fahrfunktionen entwickeln, wird die Notwendigkeit iterativer Sicherheitsprozesse, die häufige Updates ermöglichen, immer wichtiger. Dieser Abschnitt zeigt, wie sicherheitsrelevante Aktivitäten, insbesondere die System-Theoretische Prozessanalyse (STPA) [65], systematisch in den DevOps-Lebenszyklus integriert werden können, um eine kontinuierliche Sicherheitsgewährleistung zu ermöglichen.

Der Wandel der Automobilindustrie hin zu vernetzten und autonomen Fahrzeugen erfordert eine Abkehr von der traditionellen, wasserfallartigen Sicherheitsentwicklung. DevOps bietet mit seinem Fokus auf kontinuierliche Integration, Lieferung, Bereitstellung und Feldbeobachtung ein Rahmenwerk, um die Sicherheit in sich schnell entwickelnden Systemen zu gewährleisten. Die Herausforderung besteht jedoch darin, wie etablierte und neue sicherheitstechnische Methoden effektiv integriert werden können, ohne entweder die Entwicklungsgeschwindigkeit oder die Sicherheitsintegrität zu beeinträchtigen, und geeignete Metriken bereitzustellen, um die Sicherheitsleistung des Systems über seinen gesamten Lebenszyklus hinweg zu beobachten. Abbildung 4.1 zeigt die Beziehung zwischen sicherheitsrelevanten Artefakten und der Sicherheitsleistung der Fahrzeuge als ein Beispiel für die Anwendung eines solchen integrierten DevOps-Ansatzes.

Kernelement ist ein Maschinen-analysierbarer Safety Case, der durch strukturierte Argumentation nachweist, dass ein System für eine bestimmte Anwendung und Umgebung sicher ist. Der Safety Case wird anhand von Metriken (sog. Safety Performance Indicators (SPIs)) kontinuierlich validiert, welche die Leistungsfähigkeit des Safety Konzepts widerspiegeln. Die Metriken werden sowohl während der Entwicklungsphase als auch während des Betriebs erhoben. Die zu erreichenden Sicherheitsziele und deren Nachweise/Evidenzen sind mit den entsprechenden Sicherheitsanforderungen verknüpft, die systematisch durch Methoden wie STPA abgeleitet und den Komponenten der Systemarchitektur zugeordnet werden.

Erweiterte, contract-basierte Entwurfsmethoden zur Spezifikation der Sicherheitsanforderungen ermöglichen eine frühzeitige Validierung des Sicherheitskonzepts sowie die Synthese von Sicherheitsmonitoren.

Das Erfassen und Analysieren systematisch definierter SPIs und die Darstellung der Ergebnisse auf geeigneten Dashboards bieten den Safety Ingenieuren kontinuierliche Einblicke in die Sicherheitsleistung der Fahrzeugflotte.

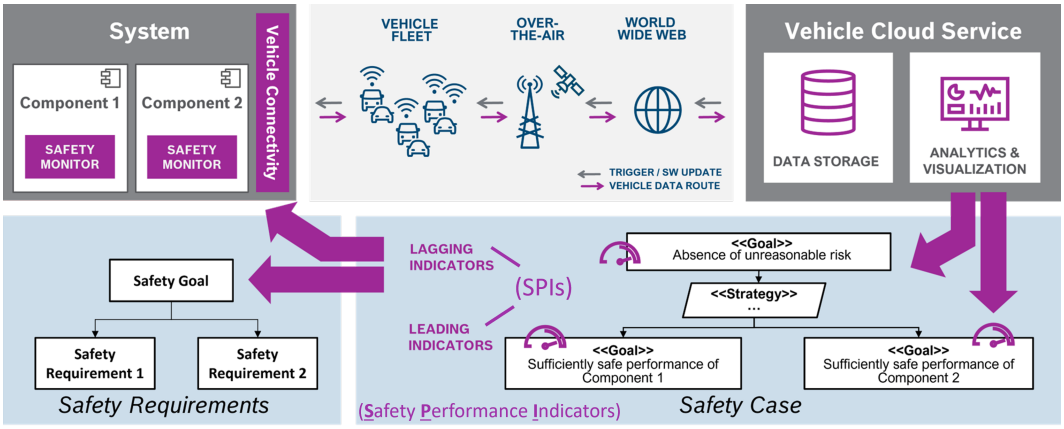


Abbildung 4.1: Übersicht über kontinuierliche Sicherheitsgewährleistung

Um die Daten-Abhängigkeiten entlang des Sicherheitslebenszyklus effizient zu handhaben und bei notwendigen Änderungen am System die von Safety Normen geforderte Auswirkungsanalyse bestmöglich zu unterstützen, ist ein werkzeuggestütztes Nachverfolgbarkeitsmodell (Traceability Infomation Model) entscheidend.

4.1.1 Sicherheitsanforderungen

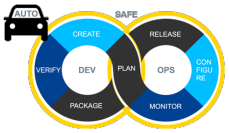
Sicherheitsanforderungen bilden die Grundlage jeder sicherheitskritischen Systementwicklung. Im Kontext von DevOps müssen diese Anforderungen sowohl streng als auch anpassungsfähig sein, um iterative Entwicklungszyklen zu unterstützen.

Safety by Design: Gemäß den Prinzipien eines sicherheitsorientierten Designs müssen Sicherheitsanforderungen von Beginn an in jede Entwicklungsphase integriert werden. STPA bietet einen systematischen Ansatz zur Ableitung von Sicherheitsanforderungen durch Analyse der Kontrollstruktur des Systems und Identifikation unsicherer Steuerungsaktionen (UCAs). STPA kann durchgeführt werden, sobald die funktionale Spezifikation des Systems vorliegt, beginnend auf einer hohen Abstraktionsebene und zunehmend konkret werdend.

Die Ableitung von Sicherheitsanforderungen durch STPA folgt einem strukturierten Ablauf:

- **Schritt 1:** Definition von Systemgefahren und entsprechenden Sicherheitsanforderungen
- **Schritt 2:** Modellierung der Kontrollstruktur inklusive Steuerungen, kontrollierten Prozessen und deren Interaktionen
- **Schritt 3:** Identifikation unsicherer Steuerungsaktionen unter bestimmten Bedingungen
- **Schritt 4:** Ableitung von Verlustszenarien und Entwicklung von Sicherheitsanforderungen zur Vermeidung oder Minderung dieser UCAs

Diese Sicherheitsanforderungen können dann bestimmten Systemkomponenten und -fähigkeiten zugeordnet werden, wie z.B. in Abschnitt 4.3 mit der PLotoDL-Sprache zur formalen Spezifikation gezeigt wird. Die Nachverfolgbarkeit zwischen Sicherheitsanforderungen und Syste-



mimplementierung ermöglicht automatisierte Verifikation und kontinuierliche Validierung im gesamten DevOps-Zyklus.

4.1.2 Verifikation der Systemsicherheit während der Entwicklung

Die Entwicklungsphase im DevOps-Kontext erfordert eine kontinuierliche Verifikation sicherheitsrelevanter Eigenschaften, um sicherzustellen, dass inkrementelle Änderungen die Systemsicherheit nicht gefährden.

Kontinuierliches Sicherheitstesten via CI/CD: Die Integration der Sicherheitsverifikation in die Continuous Integration und Continuous Deployment (CI/CD)-Pipeline stellt sicher, dass Sicherheitsanforderungen mit jedem Code-Commit validiert werden. Dieser Ansatz implementiert kontinuierliche Tests zur Verifikation von aus STPA abgeleiteten Sicherheitsanforderungen, reduziert manuellen Aufwand und gewährleistet gleichzeitig Konsistenz der Tests. Die Tests sind in die Build-Pipeline integriert und werden automatisch als Teil des Build-Prozesses ausgeführt.

Szenarienbasiertes Testen: Sicherheitssysteme arbeiten in komplexen Umgebungen mit vielfältigen, auch seltenen Situationen. STPA bietet einen systematischen Ansatz zur Identifikation kritischer Szenarien durch Analyse von:

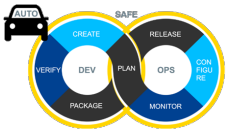
- Identifizierten Gefahren und deren Auslösebedingungen
- Unsicheren Steuerungsaktionen und deren Kontext
- Prozessmodellen, die das Systemverhalten beschreiben
- Verlustszenarien, die Kausalketten zu Gefahren aufzeigen

Diese STPA-Artefakte können zur Generierung von Testszenarien in gängigen Beschreibungsformaten verwendet werden, z.B. dem 6-Schichten-Modell für automatisiertes Fahren. Die Automatisierung der Übersetzung von STPA-Analysen in ausführbare Testszenarien stellt einen bedeutenden Fortschritt für die umfassende Sicherheitsvalidierung dar.

Formale Verifikation: Für sicherheitskritische Eigenschaften wird das Testen durch formale Verifikationsmethoden ergänzt. Durch STPA identifizierte UCAs können formalisiert und mittels Verfahren wie Model Checking gegen Komponentenverhaltensmodelle geprüft werden. Dieser Ansatz ermöglicht mathematische Nachweise sicherheitsrelevanter Eigenschaften und bietet stärkere Sicherheit als Tests allein.

4.1.3 Verifikation der Systemsicherheit während des Betriebs

Die Sicherheitsverifikation nach der Inbetriebnahme ist entscheidend, um neu auftretende Gefährdungen zu identifizieren und zu validieren, sodass die Sicherheitsannahmen im realen Betrieb weiterhin gültig sind. **Laufzeitüberwachung:** Die Betriebsphase erfordert eine kontinuierliche Überwachung zur Erkennung von Abweichungen vom erwarteten sicheren Verhalten. Zwei Konzepte sind dabei zentral:



- **Leading Indicators (LIs):** Diese erkennen potenzielle Gefahren vorausschauend, indem sie gefährliche Trends identifizieren, bevor es zu Ausfällen kommt. LIs werden aus STPA-Annahmen abgeleitet und überwachen, ob der Betriebskontext des Systems innerhalb der während des Designs angenommenen Grenzen bleibt [64].
- **Safety Performance Indicators (SPIs):** Wie in Normen wie UL 4600 [84] definiert, liefern SPIs messbare sicherheitsrelevante Metriken, die kontinuierlich überwacht werden können. Sie helfen, Sicherheitsgrenzverletzungen zu erkennen und angemessene Reaktionen auszulösen.

Die Formalisierung der STPA-Ergebnisse ermöglicht die automatische Ableitung von Überwachungsspezifikationen. Indem Verstöße gegen die im Rahmen von STPA identifizierten Sicherheitsannahmen überwacht werden, kann das System erkennen, wenn es außerhalb seiner definierten Sicherheitsgrenzen betrieben wird.

Auswertung und Analyse: Die im Betrieb gesammelten Daten müssen verarbeitet werden, um Folgendes zu identifizieren:

- Beinahe-Unfälle, die auf potenzielle Sicherheitsprobleme hinweisen
- Trends in LIs/SPIs nach Systemupdates
- Unerwartetes Verhalten, das im Design nicht berücksichtigt wurde
- Umweltbedingungen, die gefährliche Systemzustände auslösen

Diese Analyse fließt zurück in den Entwicklungszyklus und informiert über notwendige Anpassungen der Systemimplementierung und der Sicherheitsanforderungen.

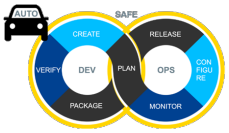
4.1.4 Änderungswirkungsanalyse von Sicherheitsartefakten

In einem DevOps-Umfeld mit häufigen Updates ist das Verständnis der Auswirkungen von Änderungen auf sicherheitsrelevante Artefakte entscheidend zur Wahrung der Systemsicherheit.

Nachverfolgbarkeit und Ausbreitung: Änderungen an der Systemfunktionalität oder neu entdeckte Gefahren erfordern eine systematische Analyse ihrer Auswirkungen auf bestehende Sicherheitsartefakte. Dazu gehören:

- Aktualisierung von STPA-Modellen zur Abbildung der Systemänderungen
- Neubewertung der Sicherheitsanforderungen auf Basis neuer Gefahren
- Anpassung der Testszenarien zur Abdeckung neuer Risikobereiche
- Anpassung der Laufzeitüberwachung zur Erkennung neuer Fehlermodi

Inkrementelle Aktualisierung des Safety Case: Jedes Systemupdate erfordert entsprechende Updates des Safety Case. Durch Aufrechterhaltung der Nachverfolgbarkeit zwischen



STPA-Artefakten und Safety-Case-Elementen (z.B. in Goal Structuring Notation) können Änderungen systematisch propagiert werden. Dies ermöglicht eine koordinierte Synchronisierung zwischen Sicherheitsanalyse und Sicherheitsargumentation und stellt sicher, dass der Safety Case trotz häufiger Systemänderungen gültig bleibt.

Weiterentwicklung von Mitigationsstrategien: Wenn die Betriebsüberwachung neue Gefahren oder ineffektive Maßnahmen aufdeckt, muss das Sicherheitskonzept weiterentwickelt werden. Dies umfasst:

- Ursachenanalyse entdeckter Sicherheitsprobleme
- Aktualisierung von STPA-Modellen mit neu identifizierten UCAs
- Entwicklung verbesserter Mitigationsstrategien
- Validierung dieser Strategien im Entwicklungspipeline

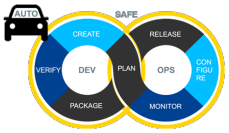
Die Integration dieser sicherheitsrelevanten Aktivitäten schafft ein geschlossenes System, in dem Sicherheit kontinuierlich analysiert, umgesetzt, verifiziert und überwacht wird – entlang des gesamten DevOps-Lebenszyklus. Dieser Ansatz stellt sicher, dass Sicherheit ein zentrales Anliegen bleibt und gleichzeitig die schnelle Iteration und Bereitstellung moderner Fahrzeugsysteme ermöglicht.

4.2 Integration sicherheitsrelevanter Artefakte in ein formalisiertes Nachverfolgbarkeits-Informationsmodell

In der traditionellen Sicherheits- und Systementwicklung werden die relevanten Entwicklungsartefakte mit unterschiedlichen Werkzeugen erstellt und verwaltet. Beispielsweise werden Gefährdungsanalysen und Risikobewertungen mit tabellenbasierten Editoren wie MS Excel durchgeführt, während die Systemarchitektur mit UML-Modellierungswerkzeugen modelliert und Anforderungen in einem Requirements-Management-Tool wie IBM Doors spezifiziert werden. Die Verwaltung sicherheitstechnischer Artefakte ausschließlich in solch verschiedenen Anwendungen ermöglicht jedoch keine effektive Etablierung von Nachverfolgbarkeit zwischen diesen heterogenen Artefakten, da sie meist nicht formalisiert vorliegen oder generell inkompatible Formate nutzen. Ein Nachverfolgbarkeits-Informationsmodell, das von den einzelnen Entwicklungswerkzeugen gepflegt wird, könnte jedoch die erforderliche Rückverfolgbarkeit ermöglichen, um Analysen (z.B. Änderungswirkungsanalysen) durchzuführen und Konsistenz sicherzustellen — beispielsweise zwischen Argumenten, die im Assurance Case darlegen, warum ein System sicher ist, und den Artefakten, die als Nachweise für diese Argumente dienen.

4.2.1 Das Traceability Information Model „Digital Dependability Identity“

Die Digital Dependability Identity (DDI) ist ein Austauschformat, das auf dem Open Dependability Exchange (ODE) Metamodell basiert und eine umfassende, werkzeugunabhängige Repräsentation von Sicherheits- und Sicherheitsartefakten cyber-physischer Systeme bietet [94]. Das ODE-Metamodell ist in drei Sammlungen unterteilt: eine unterstützt strukturierte



Safety Cases (entweder nach dem SACM-Standard [87] oder durch direkte Darstellung von Goal Structuring Notation-Elementen), eine zweite deckt Elemente der Produktentwicklung ab (wie Systemarchitektur, Gefährdungs- und Risikoanalysen, Fehlermodelle und Sicherheitsanforderungen), und eine dritte erfasst die zugrundeliegenden Engineering-Prozesse. Durch die Integration aller Artefakte und ihrer formalen Verknüpfungen in einem einzigen Modell wird die DDI zu einem einzigartigen „Zuverlässigkeitspass“ für ein System oder eine Komponente — mit dem jede Behauptung, Anforderung, Analyse und jeder Prozessschritt verbunden und konsistent gehalten wird.

Da DDIs im Eclipse Modeling Framework (EMF) [16] definiert und über einen Java-basierten Tool-Adapter mit Remote Procedure Call (RPC)-Schnittstelle zugänglich gemacht werden, können sie von verschiedenen Modellierungswerkzeugen unabhängig von der Implementierungssprache importiert, exportiert und bearbeitet werden. Alle Metamodell-Elemente werden automatisch zwischen EMF- und RPC-Datenstrukturen abgebildet, was einen nahtlosen Austausch ermöglicht. Darüber hinaus unterstützt die formale Nachverfolgbarkeit des DDI-Formats die Nutzung von Epsilon-Skripten [54] für Modelltransformation, Validierung, Vergleich, Zusammenführung, Refaktorisierung und andere (teil-)automatisierte Aufgaben. Diese Kombination aus standardisiertem Austausch, Tool-Interoperabilität und skriptbasierter Modellverwaltung macht DDI zu einem leistungsstarken Mittel, um manuellen Aufwand zu reduzieren, Fehler zu minimieren und Konsistenz über verschiedene Teams und Phasen des System- und Safety-Engineerings hinweg aufrechtzuerhalten.

4.2.2 Tool-Adapter-Erweiterung: Integrierte DDI mit nahtloser Tool-Interoperabilität

Die ursprüngliche Version des DDI-Tool-Adapters war für die lokale Nutzung auf der Maschine gedacht, auf der auch das jeweilige Modellierungswerkzeug läuft. Sie ermöglichte den Export von Modellelementen aus einem Modellierungswerkzeug in eine DDI-Modelldatei. Auf diese Weise konnten verschiedene Werkzeuge genutzt werden, um jeweils die sicherheits- und systemtechnischen Artefakte zu exportieren, die das jeweilige Werkzeug unterstützte. Beispielsweise konnten mit einem Requirements-Tool Anforderungen in ein DDI-Modell exportiert werden, während mit einem UML-Werkzeug das Architekturdesign in ein separates DDI-Modell überführt wurde. In einem zusätzlichen Schritt mussten beide DDI-Modelle eingesammelt, zusammengeführt und (teilweise manuell) integriert werden, um z.B. formale Rückverfolgbarkeit zwischen Architekturmodellen und den sie spezifizierenden Anforderungen herzustellen.

Im Rahmen von Mannheim-AutoDevSafeOps wurde der Tool-Adapter dahingehend erweitert, dass er nun auch als zentrale Serveranwendung fungieren kann. Dadurch können angebundene Modellierungswerkzeuge das zentral gespeicherte DDI-Modell eines spezifischen Systems laden, es mit den modellierungsspezifischen Elementen anreichern, die sie verwalten, und die Änderungen zurück an das zentrale DDI-Modell synchronisieren, um stets eine nahtlose Integration sicherheits- und systemtechnischer Artefakte zu ermöglichen. Letzteres wird dadurch realisiert, dass das hochgeladene DDI-Modell mit der zentral verwalteten DDI-Modellversion verglichen und anschließend in eine neue Version zusammengeführt wird. Zusätzlich zum Lade- und Update-Service wurden auch DDI-Modell-Upload/Download sowie

ein einfacher Abfrageservice technisch realisiert. Darüber hinaus wurden Konzepte für generische Trace-Erstellung, Versionierung und Merge-Konfliktauflösung entwickelt. Figure 4.2 zeigt die im Rahmen von Mannheim-AutoDevSafeOps konzeptionell und/oder technisch umgesetzten Erweiterungen des Tool-Adapters. Die grünen Bereiche wurden neu eingeführt, die gelben zeigen Anpassungen und die grauen Bereiche symbolisieren den unveränderten alten Stand.

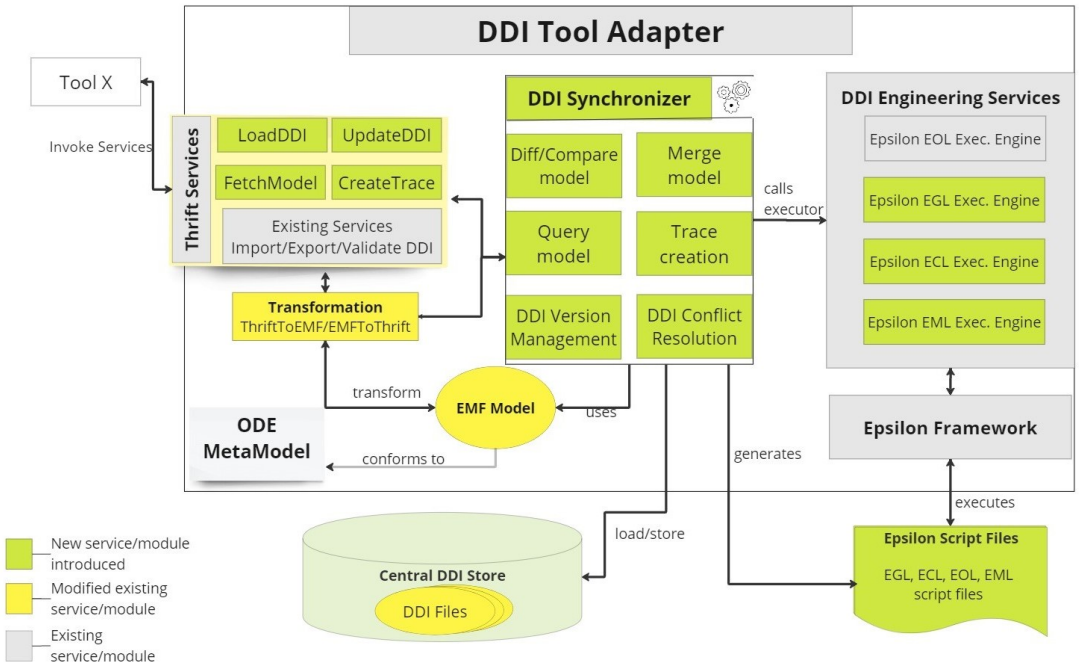
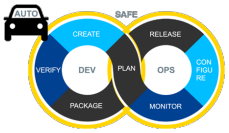


Abbildung 4.2: Tool-Adapter-Erweiterungen: DDI-Integrationsframework

Insgesamt unterstützt die aktuelle Version des DDI-Tool-Adapters im Rahmen eines Auto Dev Safe Ops-Zyklus die Erstellung und Pflege eines Traceability Information Models in der „Create“-Phase, indem sie die Integration von Modellelementen aus heterogenen Modellierungswerkzeugen erlaubt. Außerdem ermöglicht sie in der „Plan“-Phase Analysen wie etwa eine Root-Cause-Analyse basierend auf Sicherheitsleistungsindikator-Verletzungen, die durch die Laufzeitüberwachung vorheriger Fahrzeugflotten identifiziert wurden (siehe auch Abschnitt 4.7.2), und die ein formalisiertes, integriertes Modell erfordern. Die DDI-Tool-Adapter Erweiterung wurde anhand eines Beispiels mit Hilfe von den zwei Tools safeTbox und reqSuite® rm evaluiert. Dabei wurde ein Fehleranalyseergebnis aus safeTbox in ein DDI Modell integriert. Diese DDI Version wurde im Anschluss von reqSuite® rm geladen und darin anschließend für die enthaltenen Fehler Sicherheitsanforderungen definiert, welche schließlich erneut in das DDI Modell zurückgeführt und dort persistent gespeichert wurden. Eine genauere Beschreibung der Evaluation sowie Informationen über die Konzeption und der technischen Realisierung der DDI-Tool-Aadapter Erweiterung sind in D4.2[28] beschrieben.



4.3 Erweiterung der Spezifikation von Designverträgen zur Einbeziehung von Operational Design Domains (ODD) und Modi zur effektiven Modellierung von Sicherheitsanforderungen

Ähnlich wie bei der *funktionalen Sicherheit*, wie sie in dem Standard ISO 26262 spezifiziert ist, besteht eines der Hauptziele der *Sicherheit der intendierten Funktion* (SOTIF) darin, eine Nachweiskette zu erstellen, die belegt, dass alle relevanten Schadensursachen identifiziert und mitigiert wurden. Während in der funktionalen Sicherheit Fehlfunktionen im System als Ursache für Gefährdungen betrachtet werden, untersucht die SOTIF *funktionale Insuffizienzen* als Ursachen. Insuffizienzen betreffen entweder die Spezifikation oder die technischen Fähigkeiten der Implementierung von Systemkomponenten. Beide werden jedoch durch sogenannte *auslösende Bedingungen* (triggering conditions) aktiviert, die in der Umgebung des Systems auftreten können.

Dementsprechend definiert die ISO 21448 mehrere Aktivitäten zur Identifikation von funktionalen Insuffizienzen und deren auslösenden Bedingungen. Diese Bedingungen und Insuffizienzen werden, ähnlich wie Fehlfunktionen in der funktionalen Sicherheit, für die Entwicklung eines Sicherheitskonzepts genutzt. Abb. 4.3 zeigt einen Teil eines SOTIF-konformen Entwicklungsprozesses mit Methoden und Artefakten, die in dem Projekt entwickelt wurden und sich auf die Integration sogenannter *Capabilities* zur Spezifikation und Umsetzung von Sicherheitskonzepten konzentrieren.

Der Prozess beginnt oben links mit der Spezifikation der zulässigen Betriebsdomäne (Operational Design Domain, ODD) als zentrales Element der SOTIF. Eine ODD wird abstrakt beschrieben als "Bedingungen, unter denen ein bestimmtes System betrieben werden soll" [49], z.B. Wetter- und Straßenverhältnisse, aber auch Systemeigenschaften wie die Fähigkeiten zur Umfelderkennung. ISO 21448 definiert auslösende Bedingungen als solche Bedingungen in der ODD, die vorhandene Insuffizienzen im System aktivieren, und so zu Gefährdungen führen können.

In dem Projekt wurde die Sprache PLotoDL zur formalen Spezifikation von Capabilities entwickelt, mit denen Elemente der ODD mit Anforderungen an die Implementierung der Systemfunktionen in Beziehung gesetzt werden können. Die Formalisierung ermöglicht die Integration von formalen Verifikationen und Codegenerierung. Die Sprache unterstützt die Definition von ODDs in Form von Ontologien durch die Spezifikation relevanter Konzepte, Eigenschaften und Relationen. Für die Analyse von Ontologien können diese PLotoDL Spezifikationen in die weit verbreitete Ontologiesprache OWL2 exportiert werden.

Ein wichtiger Schritt bei der Identifikation auslösender Bedingungen ist die Ableitung von Szenarien, in denen gefährdendes Verhalten (hazardous behavior) auftreten kann. Abb. 4.3 zeigt oben rechts die Aktivität *Gefährdungs- und Risikoanalyse* (HARA), gefolgt von einer Analyse der Ursachen für potenziell gefährdendes Verhalten. In der funktionalen Sicherheit sind Fehlerbaumanalyse (FTA) und Auswirkungsanalyse (FMEA) als etablierte Methoden in ISO 26262 spezifiziert. Sie konzentrieren sich allerdings auf interne Ursachen im System.

Im Kontext von SOTIF wurde die System-Theoretische Prozessanalyse (STPA) als Methode vorgeschlagen, um HARA und Ursachenermittlung zu kombinieren. STPA berücksichtigt

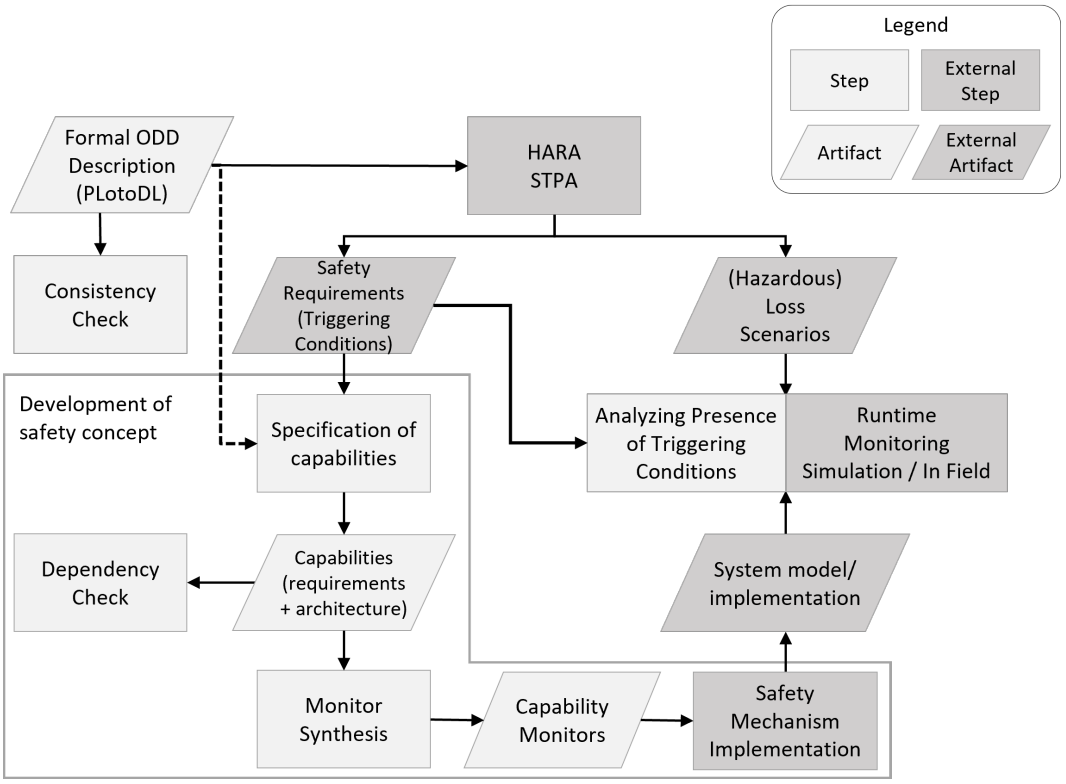


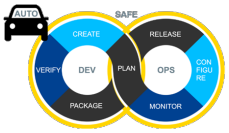
Abbildung 4.3: Schritte und Artefakte eines SOTIF-konformen Entwicklungsprozesses im Zusammenhang mit der Integration von Capabilities

explizit die Interaktion zwischen System und Umgebung und fokussiert sich auf Probleme in dieser Interaktion. Dadurch kann STPA auslösende Bedingungen in der Umgebung als Ursachen für gefährliches Verhalten identifizieren (vgl. Abschnitt 4.1). Die Ergebnisse der STPA bilden eine wichtige Grundlage für die Durchführung der in diesem Abschnitt beschriebenen Schritte.

4.3.1 Überwachung auslösender Bedingungen

Auslösende Bedingungen müssen in Daten beobachtbar sein, um eine nachträgliche Flottenüberwachung zu erlauben. Nur so kann beispielsweise geprüft werden, ob ein Sicherheitskonzept tatsächlich identifizierte Risiken mitigiert. Metric Temporal Conjunctive Queries (MTCQs) bieten eine Möglichkeit für die Spezifikation und Beobachtung von auslösenden Bedingungen [89, 88].

MTCQs sind eine logikbasierte Datenabfragesprache, ähnlich zu SQL. Der Nutzer kann jedoch mit einer Ontologie Hintergrundwissen spezifizieren. MTCQs erlauben es, in den Daten Objekte und Ereignisse zu finden, die bestimmte räumliche und temporale (auslösende) Bedingungen erfüllen. Ein Vorteil von MTCQs ist die Referenzierung der ODD als Ontologie, welche in einer ausdrucksstarken Beschreibungslogik (DL) formuliert werden kann. Die DL-



Repräsentation der PLotoDL-ODD-Formalismen von oben kann also direkt verwendet werden, um auslösende Bedingungen zu formalisieren. ODD-Axiome können damit die spätere Datensuche gezielt unterstützen. MTCQs sind im Software-Tool Topllet implementiert [90]. Im Rahmen des hier beschriebenen Entwicklungsprozesses wird dies genutzt, um in Simulationen von Szenarien bzw. im späteren Betrieb auslösende Bedingungen zu identifizieren. Gemeinsam mit einem Monitoring von Capabilities im Fahrzeug kann hiermit analysiert werden, ob ein implementiertes Sicherheitskonzept bei dem Auftreten von auslösenden Bedingungen korrekt reagiert.

4.3.2 Capabilities zur Umsetzung von Sicherheitskonzepten

Ein Sicherheitskonzept hat das Ziel, die in vorangegangenen Schritten abgeleiteten Sicherheitsanforderungen umzusetzen (vgl. Abb. 4.3). Im Rahmen dieser Arbeit liegt der Fokus auf Anforderungen der SOTIF, also auf solche Anforderungen, die sich auf die Reduktion der Auswirkungen von auslösenden Bedingungen beziehen. Die zentralen Aktivitäten hierzu in der ISO 21448 sind in Kapitel 8 „Funktionale Modifikationen zur Adressierung SOTIF-bezogener Risiken“ beschrieben. Der Standard unterscheidet dabei drei Arten funktionaler Modifikationen zur Realisierung von Sicherheitskonzepten:

1. Verbesserung der Sensorik, Aktorik oder Umfelderkennung zur Erkennung bestimmter Bedingungen.
2. Erhöhung der Wahrnehmbarkeit des Systems durch andere Verkehrsteilnehmer.
3. Begrenzung der ODD zur Vermeidung gefährlicher Bedingungen.

Das Beispiel in Abb. 4.4 zeigt einen *Construction Zone Assist* (CZA), der das Fahrzeug in einer Baustelle automatisch steuert. Die Umgebungserkennung detektiert dafür Leitkegel (Pylone). Wird ein Kegel erkannt, aktiviert der Mode Manager die entsprechende Funktion. Eine Sicherheitsanforderung spezifiziert, dass der CZA nicht in Situationen mit Sichtverhältnissen aktiviert wird, in denen Kegel nicht zuverlässig erkannt werden können – z.B. bei Regen oder in Tunneln.

Die ODD-Spezifikation in PLotoDL beschreibt entsprechende auslösende Bedingungen (zusammengefasst in `CriticalCondition`). Daraus wird das Sicherheitskonzept abgeleitet. Es besagt, dass das CZA System nicht aktiviert wird, wenn eine solche Bedingung erkannt wird. Die betroffene Komponente (`ODD Handling`) benötigt hierfür die Capability (`requires observe CriticalCondition`), um zu entscheiden, ob der CZA aktiviert werden darf. Im Falle des Detektion wird dies dem `ModeManager` signalisiert.

Zum Überprüfung der konsistenten Umsetzung des Sicherheitskonzepts wurde ein automatisierte *Dependency Check*-Analyse implementiert, die überprüft, ob die in den Capabilities genutzten Eigenschaften von dem System überhaupt realisiert werden können. Alle formulierbaren Eigenschaften hängen direkt oder indirekt mit der spezifizierten ODD zusammen, und basieren auf der Annahme, dass diese beobachtet werden können. In dem Anwendungsbeispiel sind dies Regen und Tunnel. In der CZA-Anwendung wird beispielsweise überprüft, ob die in der Capability spezifizierten Eigenschaften des `ODD Handling` entweder durch eine

Safety Requirement: The CZA must not be activated in tunnels or if it is raining

Formalized ODD:

Consistency Check

```
Tunnel isa Object;
is-in(Object, Object);
…
Environment.Rain : bool;
…
CriticalCondition() := (exists Tunnel(t): is-in(ego,t))
  || (exists Environment(e): is-in(ego,e) && e.Rain = true);
```

Capabilities:

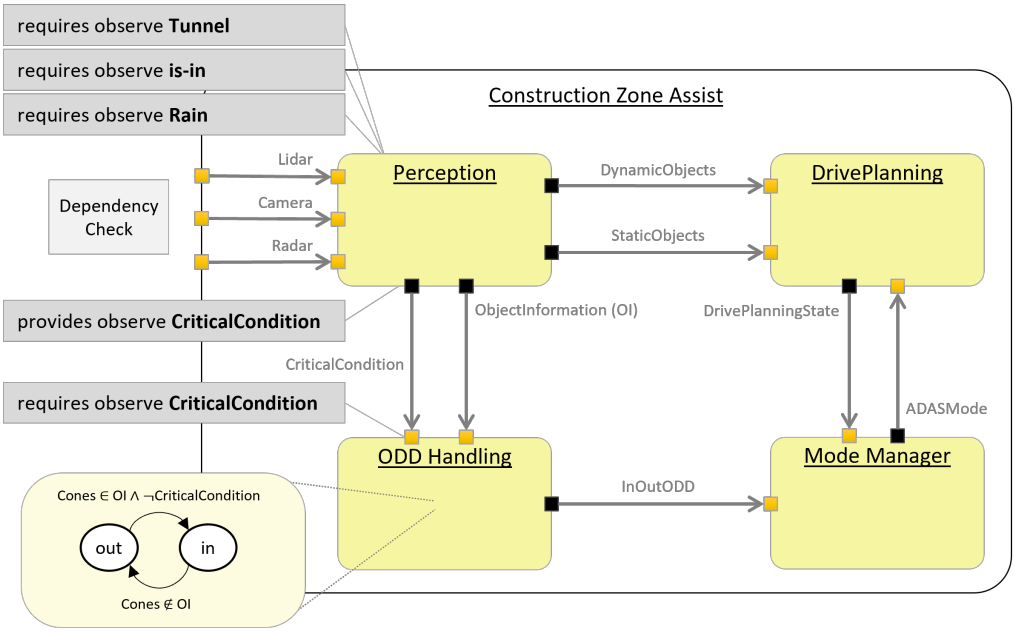
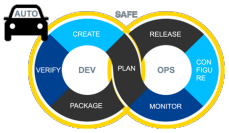


Abbildung 4.4: Sicherheitsanforderung aus der HARA eines Construction Zone Assist, ein entsprechender ODD-Ausschnitt in PLotoDL, sowie abgeleitete Capabilities im Systemmodell

verbundene Komponente (z.B. `Perception`) zur Verfügung gestellt wird, oder mit Hilfe der vorhandenen Eingangsdaten berechnet werden kann.

Für die Umsetzung des Sicherheitskonzeptes werden darüber hinaus Anforderungen in Form von Contracts formuliert, mit denen mode-basiertes Verhalten spezifiziert werden kann. Die Spezifikation der Modi und Transitionen in `ODD Handling` basiert auf einer Erweiterung der MTSL-Timing-Contracts [57]. Der folgende Contract beispielsweise zeigt auf, wie das Ausgangssignal `InOutODD` der Komponenten `ODD Handling` auf der Basis der Präsenz oder Abwesenheit einer kritischen Umgebungsbedingung gestellt wird:

A	*CriticalCondition* occurs every 100 *ms*.	1
	OI occurs every 100 *ms*.	2
G	Reaction([*OI.Cones, CriticalCondition.false*],*set(mode, in)*) within $[55, 66]$ *ms*.	3
	Reaction(*OI.NoCones*,*set(mode,out)*) within $[55, 66]$ *ms*.	4
	InOutODD.in occurs every 100 *ms*. in mode *in*.	5
	InOutODD.out occurs every 100 *ms*. in mode *out*.	6



Diese formalen Contract-Spezifikationen erlauben eine automatische Verifikation, z.B. ob die Sicherheitsmechanismen die Anforderungen erfüllen [10]. Außerdem können auf diesen Contracts basierende Laufzeit-Monitore generiert werden. Die Monitor-Generierung hierzu erweitert mit automatisch generierbarem Code aus PLotoDL-Spezifikationen.

4.3.3 Updates mit modusbasierten Komponenten-Verträgen

Die Spezifikation von Betriebsmodi auf Komponentenebene [56, 57] erlaubt es, Updates zur Laufzeit durchzuführen. Dazu wird sich die Eigenschaft der gegenseitigen Ausschließung von Modi zu nutze gemacht. Diese Eigenschaft besagt, dass nur ein Modus einer Compoennte zur selben Zeit aktiv sein kann. Die Grundidee ist die aktiven phase eines Modus zu nutzen, um ein Update über die inaktiven Modi durchzuführen und so eine Sicherstellung von Integrationsbedingungen zu garantieren [58].

Um dies zu erreichen wird jeder Modus m_i durch eine eigene Komponente C_{m_i} mit zugehörigem Contract $\mathscr{C}_{m_i}$ modelliert. Unter der Annahme der Eigenschaft der gegenseitigen Ausschließung gilt:

$$\mathscr{C}_C \succcurlyeq \bigotimes_{i=1}^{n} \mathscr{C}_{m_i}.$$

Ein Update in Form des Austauschens einer Komponente – in diesem Fall einer Moduskomponente – kann erfolgen, sofern der betroffene Modus inaktiv ist und garantiert werden kann, dass er über eine festgelegte Dauer inaktiv bleibt. Die Verfeinerung der Verträge zwischen System- und Moduskomponenten orientiert sich an den Regeln des vertragsbasierten Designs [12]. So entsteht ein systematischer, verifizierbarer Ansatz für Software-Updates in sicherheitskritischen Systemen zur Laufzeit.

4.3.4 Synthese von Update-Strategien

Um zur Laufzeit Updates garantiert sicher durchführen zu können, müssen sichere Zeitpunkte gefunden werden, in denen die im vorherigen Abschnitt beschriebenen Aspekte i) der betroffene Modus ist inaktiv und ii) der betroffene Modus bleibt für die notwendige Zeit des Updates inaktiv, garantiert werden. Zur Synthese dieser Zeitpunkte wird ein spieltheoretischen Ansatz verwendet, wie er in [59] beschrieben ist, in welchem die Spezifiaktion von Modi eine zentrale Methodik darstellt.

Der Updateprozess wird als reaktives, zeitliches 2-Spieler Spiel modelliert. Spieler 1 repräsentiert das Update, während Spieler 2 das System inklusive der Umgebung repräsentiert und als vollständig unkontrollierbar betrachtet wird. Die Verhaltensweisen in Form von Moduswechseln der Spieler lassen sich anhand ihrer Spezifiaktion ableiten.

Das Spiel ist definiert als zeitlicher Automat:

$$A = (L, L_0, C, \Sigma_c, \Sigma_u, E, I),$$

in dem reale Uhren C die Verweildauern der Zustände regulieren und Übergänge in Σ_c (kontrollierbar für Spieler 1) und Σ_u (nicht kontrollierbar für Spieler 2) unterschieden werden [11]. Jede Moduskombination zwischen den Spielern wird durch eine Location im zeitlichen Automaten des Spiels repräsentiert. Die Semantik erlaubt abwechselnd Verzögerungen und diskrete

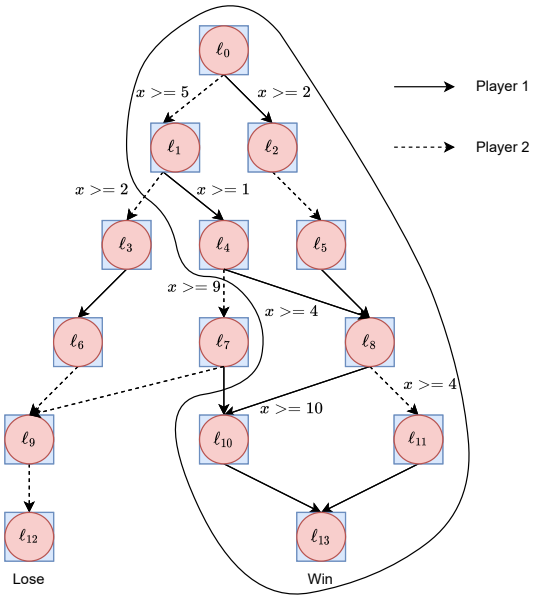


Abbildung 4.5: Beispiel einer Gewinnstrategie in einer Timed Game Automata

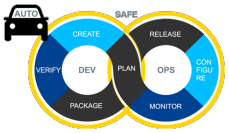
Schritte, in denen Wechsel zwischen den Locations vollzogen werden, wobei beide Spieler eine Verzögerungszeit vorschlagen und der kleinere Vorschlag die gemeinsamen Zeitverzögerung und den ausführenden Spieler für einen diskreten Übergang bestimmt.

Es wird nun versuchen, eine Abfolge von Zügen (Strategie) für Spieler 1 auf der Grundlage von Gewinnbedingungen zu synthetisieren. Dazu werden zwei Gewinnbedingungen in Bezug auf Erreichbarkeit und Sicherheit kombiniert:

- **Erreichbarkeit:** Spieler 1 erreicht irgendwann einen Endzustand.
- **Sicherheit:** Spieler 1 vermeidet stets kritische Zustände (z.B. gleichzeitige Aktivität von Update und betroffener Komponente).

Abb. 4.5 zeigt ein Beispiel eines zeitlichen Spiels in form eines Spielgraphen, in dem eine Gewinnstrategie identifiziert wurde. Spieler 1 gewinnt das Spiel, sobald er bestimmte Zustände (Modi) durchlaufen und den Endzustand erreicht hat. Aufgrund der Ableitung von minimalen und maximalen Zeiten in jedem Modus kann die Strategiesynthese vollständig zur Entwurfszeit des Updates durchgeführt werden.

Die Synthese einer Gewinnstrategie liefert somit direkt einen vollständigen Aktualisierungsplan, der garantiert, dass das Update niemals in Konflikt mit der Systemfunktionalität gerät und stets dessen Zeitvorgaben einhält. Mit Algorithmen wie in UPPAAL Tiga [11] kann eine solche Strategie, basierend auf allen möglichen Spielverläufen und Uhrwerten, synthetisiert werden.



4.4 Adaptive Softwarearchitekturen für die Resilienz sicherheitskritischer Systeme

Im sich wandelnden Umfeld automobiltechnischer Systeme ist Resilienz zu einer zentralen Anforderung geworden, um sowohl Sicherheit als auch betriebliche Zuverlässigkeit zu gewährleisten. Resilientes Design in automobilen Systemen bezieht sich auf die Fähigkeit eines Systems, sich an Veränderungen anzupassen, sich von Fehlern zu erholen und die Funktionalität unter unterschiedlichen Bedingungen aufrechtzuerhalten. Dieser Abschnitt definiert die Prinzipien des resilienten Designs und untersucht Strategien zur Erreichung von Resilienz in automobilen Architekturen. Durch die Integration von Resilienz sowohl in die Entwicklungs- als auch in die Betriebsphasen unterstützt der ADSO-Prozess moderne automobile Systeme dabei, ihre Fähigkeit zur Bewältigung von Unsicherheiten zu verbessern und so Sicherheit und Effizienz zu gewährleisten.

Automobile Systeme müssen sich an zwei Arten von Veränderungen anpassen:

- (Post-)Entwicklungs-Systemerweiterungen: Dies umfasst Änderungen am System, an seinen Anforderungen oder an Vorschriften nach der ursprünglichen Entwicklungsphase, insbesondere während des Betriebs. Diese Änderungen können auf unterschiedlichen Zeitskalen auftreten – potenziell so schnell wie innerhalb weniger Tage oder seltener – und werden hauptsächlich von den Systementwicklern überwacht und durch die Lebenszyklusprotokolle gesteuert. Resilienz gegenüber solchen Änderungen wird in erster Linie durch die Architektur des Systems gewährleistet.

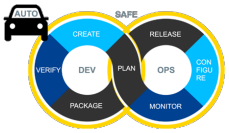
- Betriebliche Umweltveränderungen: Dies betrifft Änderungen in der Umgebung während des Systembetriebs, wie Bewegungen anderer Verkehrsteilnehmer usw. Diese Änderungen sind kontinuierlich und können innerhalb von Sekundenbruchteilen auftreten. Die Reaktion auf diese Änderungen erfordert daher ein autonomes Handeln des Systems. Die Bewältigung dieses Veränderungstyps nutzt Verfahren des maschinellen Lernens, die es dem System ermöglichen, adaptiv auf seinen dynamisch wechselnden Betriebskontext zu reagieren.

Resilienz kann in verschiedene Typen eingeteilt werden, je nach Art der Herausforderungen und den Strategien zu ihrer Bewältigung. Wir haben das REMIND-Framework übernommen, das Resilienz in vier Schlüsselstrategien klassifiziert: *Detection*, *Mitigation*, *Recovery* und *Endurance* [79].

4.4.1 Resiliente adaptive Architekturen

Zur Stärkung der Resilienz sicherheitskritischer Systeme werden adaptive Softwarearchitekturen entwickelt, die eine reaktionsfähige Umstellung auf einen sicheren Betrieb ermöglichen, sobald unsicheres Verhalten erkannt wird.

Resilienz umfasst Mechanismen zur Fehlererkennung und -behandlung, zur Systemwiederherstellung, zur Redundanz sowie zur Aufrechterhaltung sicherer und autonomer Betriebszustände innerhalb der Arbeitsumgebung. Diese Mechanismen erfordern das architektonische Design als Rückgrat eines resilienten Systementwurfs.



Traditionelle monolithische oder eng gekoppelte Architekturen haben Schwierigkeiten, die Flexibilität, Anpassungsfähigkeit und Fehlerisolation bereitzustellen, die für resiliente automobile Systeme erforderlich sind. Um eine Architektur zu entwerfen, die diese Anforderungen erfüllt, haben wir Service-oriented Architectures (SOA) übernommen, die das ideale Rückgrat für die inkrementelle Entwicklung von Fahrzeugfunktionen und die Erweiterung ihres Einsatzbereichs darstellen. Die Kerneigenschaften von SOA – lose Kopplung, Service-Autonomie und dynamische Service-Komposition – entsprechen direkt den Bedürfnissen moderner automobiltechnischer Softwareplattformen und stellen sicher, dass Funktionalität und Sicherheit auch unter Fehlerbedingungen erhalten bleiben.

Während SOA modulare und skalierbare Lösungen bietet, erfüllen sie strenge Sicherheitsanforderungen nur unzureichend. Diese Arbeit verbessert die Zuverlässigkeit und Sicherheit automatisierter Fahrsysteme durch die Einführung einer Taxonomie von Monitoring-Aspekten und eines Ansatzes zur Synthese von Laufzeitmonitoren, beide zugeschnitten auf Service-oriented Architectures [76].

Basierend auf der von Broy und Stølen [15] eingeführten FOCUS-Notation, mit Anpassungen für automobile Anwendungen wie von Kugele et al. [60] vorgeschlagen, führt unser Ansatz [76] Änderungen und Erweiterungen ein, um besser mit den Anforderungen resilienter adaptiver Architekturen übereinzustimmen: (i) Erweiterung von SOA um contract-based design, indem Schnittstellenzusicherung mit Verträgen verknüpft werden; (ii) Vorschlag einer Taxonomie von Monitoring-Aspekten für SOA-Designs, wobei jeder Aspekt systematisch aus Verträgen abgeleitet wird. Diese beiden Punkte werden im Folgenden detailliert.

Im Kontext dieser Notation wird ein System $\mathcal{S} \in \texttt{SYSTEM}$ vollständig durch seine *syntaktische* und *semantische* Schnittstelle charakterisiert. Die syntaktische Schnittstelle umfasst typisierte Kanäle $c \in \texttt{CHANNEL}$, die eine Interaktion mit der Systemumgebung ermöglichen. Beachte, dass `SYSTEM`, `CHANNEL` und `MESSAGE` das Universum aller Systeme, Kanäle und Nachrichten darstellen.

Definition 1 (Syntaktische Schnittstelle)**.** Sei $I \subseteq \texttt{CHANNEL}$ eine Menge typisierter Eingangskanäle und $O \subseteq \texttt{CHANNEL}$ eine Menge typisierter Ausgangskanäle mit $I \cap O = \varnothing$. Das Paar (I, O) charakterisiert die *syntaktische Schnittstelle* eines (Sub-)Systems. Die syntaktische Schnittstelle wird beschrieben durch $(I \triangleright O)$.

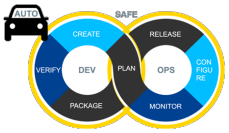
Die *semantische Schnittstelle* hingegen definiert das Verhalten des Systems über eine Eingabe/Ausgabe-Relation.

Definition 2 (Semantische Schnittstelle)**.** Das logische und zeitliche Verhalten, d. h., die *semantische Schnittstelle* eines Dienstes $\mathfrak{s}$ mit syntaktischer Schnittstelle $(I \triangleright O)$ ist durch die Funktion $f : \vec{I} \to \wp(\vec{O})$ definiert.

Definition 3 (Gezeiteter Strom)**.** Sei $M \subseteq \texttt{MESSAGE}$ die Menge von Nachrichten gleichen Typs $T \in \texttt{TYPE}$. Ein *gezeiteter Strom* s des Typs T ist eine Funktion

$$\mathsf{s} : \mathbb{N}^+ \to M \cup \{\bot\}.$$

Die syntaktische Schnittstelle definiert die Interaktionspunkte eines Systems, aber nicht alle sind für jedes Subsystem relevant. Durch *Abstraktion* wird sie zur *Dienstschnittstelle*



vereinfacht, die sich auf die angebotenen und konsumierten Dienste konzentriert, wie z. B. das Erzeugen, Konsumieren oder Verarbeiten von Daten, anstatt auf detaillierte Interaktionen.

Definition 4 (Dienst, Dienstschnittstelle)**.** Ein *Dienst* $\mathfrak{s}$ wird charakterisiert durch seine (i) syntaktische Schnittstelle $(I^{\mathfrak{s}} \triangleright O^{\mathfrak{s}})$ und seine (ii) semantische Schnittstelle, gegeben durch die Funktion f, mit $I^{\mathfrak{s}} \subseteq I$ und $O^{\mathfrak{s}} \subseteq O$, wobei $(I \triangleright O)$ die syntaktische Schnittstelle des Systems $\mathcal{S}$ ist, welches den Dienst $\mathfrak{s}$ anbietet. Die Menge der Dienste eines Systems $\mathcal{S}$ wird durch $\mathfrak{S}$ beschrieben. Die *Dienstschnittstelle* eines (Sub-)Systems $s \in \texttt{SYSTEM}$ wird beschrieben durch $(P, C)^{\#}_{s}$, wobei P die Menge der *angebotenen* und C die Menge der *konsumierten* Dienste beschreiben.

Subsysteme, ihre Interaktionen und zugehörige Schnittstellenzusicherungen bilden die *Architekturspezifikation* eines Systems.

Definition 5 (Architekturspezifikation)**.** Eine *Architekturspezifikation* $\mathcal{A}$ für ein System $\mathcal{S} \in \texttt{SYSTEM}$ ist ein Trippel $(S, \multimap, Q)$, wobei $S \subseteq \texttt{SYSTEM}$ die Menge der beinhalteten Subsysteme, $\multimap \subseteq (\mathfrak{S} \times \mathfrak{S})$ die Menge aller *Angebot/Nutzungs-Beziehungen* zwischen Subsystemen und Q die Menge aller *Schnittstellenzusicherungen* ist.

Wie alle anderen Systemkomponenten ist auch der Monitor ein (Sub-)System. Um einen geeigneten Laufzeitmonitor $\mathcal{M}$ zu erzeugen, müssen sowohl die *syntaktische* als auch die *semantische Schnittstelle* des Monitors für das beobachtete System $\mathcal{S}$ beschrieben werden.

Definition 6 (Laufzeitmonitor)**.** Sei $\mathcal{M} = (\mathcal{S}, O_{\mathcal{M}})$ ein *Laufzeitmonitor* für das System $\mathcal{S}$ mit syntaktischer Schnittstellen $(I_{\mathcal{S}} \triangleright O_{\mathcal{S}})$ und $(I_{\mathcal{M}} \triangleright O_{\mathcal{M}})$, entsprechend, mit $I_{\mathcal{M}} = I_{\mathcal{S}} \cup O_{\mathcal{S}}$ und $O_{\mathcal{M}}$ sei die Menge der Ausgangskanäle. Der Monitor überwacht die Einhaltung der Schnittstellenzusicherung Q des Systems (d. h. den Kontrakt) welcher in der Architekturspezifikation $\mathcal{A}$ definiert ist.

Um Laufzeitmonitore zu synthetisieren, sind zwei Dinge wesentlich: (i) die Identifikation der spezifischen Komponente oder des Dienstes; (ii) die Bestimmung der zu überwachenden Eigenschaften. Das Schnittstellenverhalten eines Systems $\mathcal{S}$ kann als Formel in Prädikatenlogik angegeben werden. Die Schnittstellenzusicherung Q definiert das *Schnittstellenverhalten* $f_{\mathcal{S}}$ des Systems $\mathcal{S}$. Für Eingabe- und Ausgabe-Historien $x \in \vec{I}$ und $y \in \vec{O}$ schreiben wir auch $q(x, y)$. Das Schnittstellenverhalten $f_{\mathcal{S}}$ erfüllt die Spezifikation des Systems $\mathcal{S}$ mit der Schnittstellenzusicherung $q(x, y)$, falls $\forall\, x \in \vec{I}, y \in \vec{O}\colon\ y \in f_{\mathcal{S}}(x) \Rightarrow q(x, y)$. Eine Schnittstellenzusicherung ist eine *Invariante*, die stets erfüllt sein muss. Ein häufig verwendetes Paradigma ist das der *Annahme/Garantie* (A/G)-Spezifikationen, d. h. die Schnittstellenzusicherung Q kann als $Q \equiv A \Rightarrow G$ geschrieben werden, wobei A und G die Prädikate beschreiben, die die *Annahme* und die *Garantie* darstellen (vgl. Abb. 4.6). Eine Menge von Schnittstellenzusicherungen wird kanonisch als logischer Ausdruck dargestellt, indem die einzelnen Zusicherungen logisch verknüpft werden.

Diese Arbeit führt eine Taxonomie ein, die komplexe Anforderungen in klar abgegrenzte Monitoring-Aspekte unterteilt und so die Komplexität beherrschbar macht. Die Taxonomie bietet einen strukturierten Ansatz, um Anforderungen auf hoher Ebene in spezifische, messbare Einheiten zu zerlegen. Darüber hinaus können diese Aspekte als Grundlage für Verträge

$[\mathcal{S}]$
in I
out O
Q

Abbildung 4.6: Schnittstellenzusicherung eines Systems

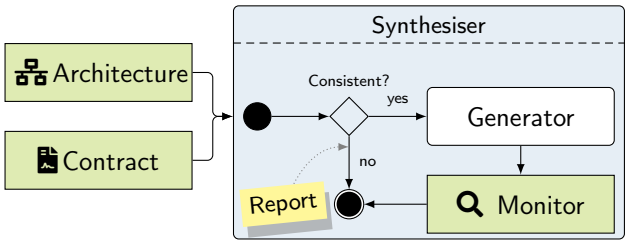


Abbildung 4.7: Monitor-Syntheseprozess

dienen, die das erwartete Verhalten von Systemkomponenten definieren und damit die automatisierte Laufzeitverifikation unterstützen. Nur durch die Überwachung dieser Aspekte lässt sich feststellen, ob Zusicherungen eingehalten wurden und, falls nicht, welche Maßnahmen ergriffen werden müssen.

Bei der Erstellung von Monitoren unterscheiden wir zwischen Black-Box- und Glass-Box-Ansatz. Der Black-Box-Ansatz berücksichtigt ausschließlich die Eingabe/Ausgabe-Beziehungen, während der Glass-Box-Ansatz auch interne Daten betrachtet und Systemänderungen erfordert, um relevante Informationen zu veröffentlichen. In diesem Abschnitt konzentrieren wir uns ausschließlich auf den Black-Box-Ansatz, der den höchsten Automatisierungsgrad bietet, da er keine internen, manuellen Modifikationen am System erfordert. Darüber hinaus sind viele Standard- und KI-gestützte Komponenten, die häufig als Drittprodukte in größere Systeme integriert werden, schwer instrumentierbar. Der Black-Box-Ansatz ermöglicht die Überwachung des intendierten Verhaltens solcher Komponenten, ohne invasive Änderungen vorzunehmen.

Kenntnisse über die Architektur und den zu erfüllenden Vertrag sind erforderlich, um Laufzeitmonitore zu synthetisieren. Vor der Codegenerierung ist die Konfliktprüfungsphase entscheidend, um die Konsistenz und Umsetzbarkeit der Schnittstellenzusicherungen zu validieren. Jede Schnittstellenzusicherung wird in dieser Phase analysiert, um die Kompatibilität mit den anderen sicherzustellen. Beispiel: Wenn eine Softwarekomponente frische Daten jede Sekunde publizieren muss, darf ihre Antwortzeit eine Sekunde nicht überschreiten. Solche potenziellen Konflikte frühzeitig zu erkennen, ermöglicht es, dass die generierten Monitore Anforderungen effektiv durchsetzen können, ohne die Systemleistung zu beeinträchtigen. Der generierte Monitor kann an die vorgesehenen Ein- und Ausgänge jedes Systems angeschlossen werden und so die Echtzeitleistung während des Betriebs beobachten. Sobald die Konsistenzprüfung erfolgreich abgeschlossen ist, beginnt der Generierungsprozess, und ein zusätzliches Systemelement – der Laufzeitmonitor – wird in die ursprüngliche Architektur eingebettet.

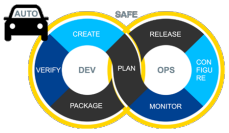


Tabelle 4.1: Parametrierte Monitoring-Aspekte, Unteraspekte, und Kontrakte

Temporal Monitoring

Response Time (τ)

Verifies that the time between receiving an input and producing the corresponding output does not exceed τ.

in $I = \{i\}$ **out** $O = \{o\}$

$\forall\, i \in \vec{I}, o \in \vec{O}, \forall\, t_i \in \mathbb{N}^+ \,.\, \exists\, t_o \in \mathbb{N}^+, t_o \leq t_i + \tau \colon i(t_i) \neq \bot \Rightarrow o(t_o) \neq \bot$

I/O Rates (f)

The rate (frequency in Hz) of messages with $\ell = 1/f$.

in $I = \{i\}$ **out** $O = \{o\}$

$\forall\, i \in \vec{I}, o \in \vec{O}, \forall\, t \in \mathbb{N}^+ \colon i(t) \neq \bot \Rightarrow o(t) \neq \bot \wedge m\#((i \uparrow t) \downarrow \ell) = m\#((o \uparrow t) \downarrow \ell)$ with $m \neq \bot$

Plausibility Monitoring

Range Check ($[l, u]$)

Checks whether an in-/output value is within a given range $[l, u]$.

out $O = \{o\}$

$\forall\, o \in \vec{O}, \forall\, t \in \mathbb{N}^+ \colon o(t) \in [l, u]$

Threshold Check ($\circ, \theta$)

Checks whether an in-/output value is below/above/equal (with relational operator $\circ \in \{<, \leq, >, \geq, =\}$) a given threshold θ.

out $O = \{o\}$

$\forall\, o \in \vec{O}, \forall\, t \in \mathbb{N}^+ \colon o(t) \circ \theta$

Temporal Consistency Check ($\Delta t, \epsilon$)

Verifies whether data values remain consistent over time, constrained by physical laws (e. g., velocity, acceleration). Two approaches: (i) limited deviation ϵ over time Δt or (ii) limited deviation ϵ relative to the mean within a sliding window of length Δt.

out $O = \{o\}$

$\forall\, o \in \vec{O}, \forall\, t \in \mathbb{N}^+ \colon$ (i) $|o(t + \Delta t) - o(t)| \leq \epsilon$ or (ii) $\left| o(t) - 1/\Delta t \sum_{i=1}^{\Delta t} o(t - i) \right| \leq \epsilon$ with $\Delta t \leq t$

Cross Data Consistency Check (τ, θ)

Verifies whether different data sources consistently describe the same phenomenon (e. g., distance) based on a similarity measure S, even when time-shifted by τ.

in $I = \{i_1, i_2\}$

For $i_1, i_2 \in \vec{I}, \forall\, t \in \mathbb{N}^+ \colon S(\tau_{max}) \geq \theta$ with $\tau_{max} = \arg max_{\eta \in [-\tau, \tau]} S(\eta)$ and $S(\tau) = \frac{i_1(t) \cdot i_2(t-\tau)}{\sqrt{i_1(t)^2 + i_2(t-\tau)^2}}$

Resource Monitoring

Resource Usage Check (ρ, θ)

Verifies whether resource usage ρ exceeds defined thresholds θ. Resources include $\rho \in \texttt{RESOURCE} = \{Flash, SRAM, EEPROM, DRAM, CPU, time, Energy\}$

$\forall\, t \in \mathbb{N}^+, \rho \in \texttt{RESOURCE} \colon \hat{\rho}(t) \leq \mathsf{c}(t, \rho)$ if $\mathsf{c}(t, \rho) \neq \bot$, $\hat{\rho}(t)$ is the resource usage of ρ at time t.

Liveliness Monitoring

Liveliness Check ($\Delta t, \epsilon$)

Checks whether $\mathcal{S}$ is responsive, i. e., messages are sent within an interval Δt and tolerance ϵ.

out $O = \{o\}$

$\forall\, o \in \vec{O}, \forall\, k \in \mathbb{N}^+ \exists\, t_k \in \mathbb{N}^+ \colon o(t_k) \neq \bot \wedge \Delta t - \epsilon \leq (t_{k+1} - t_k) \leq \Delta t + \epsilon$

If certain aspects apply to both input and output values, we illustrate them solely for the output values without any loss of generality.

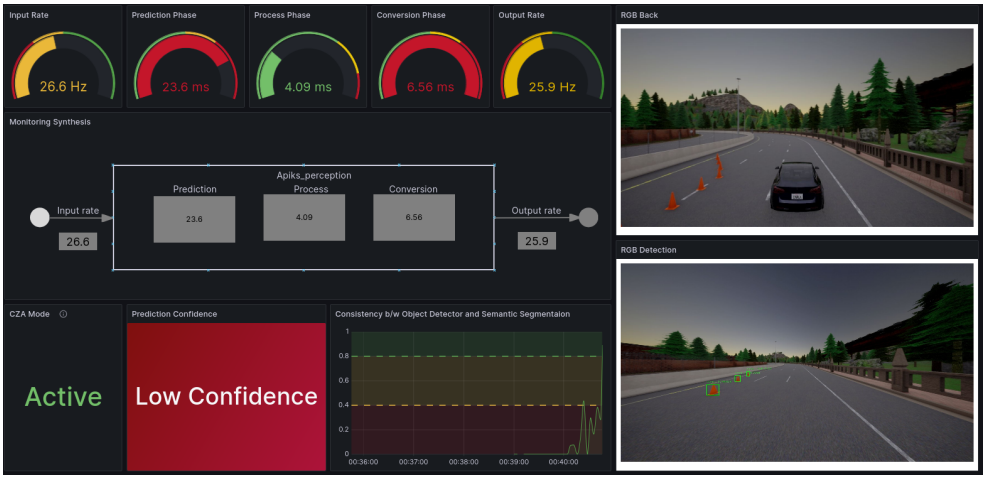
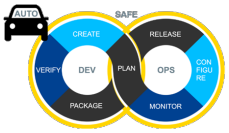


Abbildung 4.8: Safety-Dashboard und CARLA-Simulation

Schlägt die Prüfung fehl, wird ein Bericht erstellt, der Ingenieuren hilft, die Verträge zu verbessern. Der Prozess ist in Abb. 4.7 dargestellt.

Das Monitoring-Framework wurde auch in einem realistischen Szenario unter Verwendung des CARLA-Simulators evaluiert, der mit einem ROS-2-basierten Autonomes-Fahren-Stack integriert wurde. Der ausgewählte Anwendungsfall konzentrierte sich auf CZA, bei dem das Ego-Fahrzeug auf temporäre Hindernisse wie Pylonen und veränderte Straßenlayouts reagieren muss (siehe Abb. 4.8). Das Monitoring konzentrierte sich auf das Perceptions-Subsystem. Anforderungen – wie Antwortzeitgrenzen und Schwellwertprüfungen – wurden in strukturierten JSON-Dateien definiert. Nach der Validierung wurden Monitore automatisch generiert und eingesetzt, ohne die ursprünglichen Komponenten zu verändern oder zu unterbrechen. Zur Laufzeit abonnierten die Monitore die relevanten Topics und bewerteten die Vertragseinhaltung in Echtzeit. Jede Abweichung von den spezifizierten Beschränkungen – wie verzögerte Ausgaben oder Objekterkennungen mit niedriger Konfidenz – wurde erfolgreich erkannt, protokolliert und auf einem Dashboard visualisiert. Dies bestätigte die Fähigkeit des Frameworks, Fehler effektiv zu erkennen, ohne den normalen Betrieb des Systems zu beeinträchtigen.

Durch die Übernahme von serviceorientierten Architekturen (SOA) ermöglichen wir die notwendige Flexibilität, Anpassungsfähigkeit und Fehlerisolation, die traditionellen monolithischen Designs fehlen. Unsere Arbeit verbessert SOA speziell für sicherheitskritische Anwendungen durch eine neuartige Taxonomie von Monitoring-Aspekten und einen Ansatz zur Synthese von Laufzeitmonitoren. Zusammen mit der Erweiterung des mode-aware contract-based design, wie in [57] entwickelt, das die Spezifikation von Modi, Moduswechseln und modusspezifischem Verhalten ermöglicht, unterstützt die beschriebene Architektur die Spezifikation von Post-Development-Systemerweiterungen. Eine weitere entscheidende Erweiterung dieser Arbeit wären Mitigationsstrategien. Wenn eine Abweichung oder ein Fehler durch das Monitoring erkannt wird, werden Mitigationsstrategien als dynamisch synthetisierte Aktionspläne entworfen, die sicherstellen sollen, dass das System weiterhin sicher betrieben werden kann oder zumindest kontrolliert in einen sicheren Zustand überführt wird, um Schaden zu



minimieren. Sie rüsten das System im Wesentlichen mit dem Wissen aus, wie im Fehlerfall zu handeln ist, und verhindern so einen vollständigen Systemausfall oder einen Übergang in einen gefährlichen Zustand.

Darüber hinaus unterstützen wir die Entwicklung und Integration von Komponenten in einem System mithilfe von SOA. Während oft der Fokus auf den Diensten und deren Funktionalität liegt, erfordern auch die Softwarekomponenten zwischen Hardware und funktionalen Diensten Aufmerksamkeit. Hypervisoren zur Bereitstellung von Isolation oder Containerisierung für Software-Pakete und Updates tragen zwar zur Modularität bei, bringen aber auch zusätzliche Komplexität und benötigen eigene Ressourcen. In [73] präsentieren wir eine modulare Architekturvorlage mit dem Ziel, alle Softwarekomponenten zu spezifizieren – einschließlich Hypervisoren, Betriebssystem bis hin zu den Diensten selbst. Mit diesem Ansatz können wir die Hierarchie der Module und deren Ressourcenzuweisung definieren, was einfachere Änderungen (z. B. Updates) durch Hinzufügen, Verschieben oder Entfernen von Modulen ermöglicht. Wir erlauben die Verwendung einer Ressourcenspezifikationssprache, einschließlich solcher mit Contracts (entsprechend dem *Contract-Based Design*), auf den Modulschnittstellen, um Kompatibilitätsprüfungen und die Ableitung von Laufzeitmonitoren zur Unterstützung des Ressourcenmonitorings zu ermöglichen.

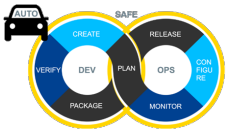
4.4.2 Resilienz gegenüber Post-Development-Systemerweiterungen

Der Entwicklungslebenszyklus moderner Fahrzeuge erstreckt sich über die Produktion hinaus, insbesondere bei softwaredefinierten Fahrzeugen, bei denen Fahrzeugfunktionen zunehmend nachträglich entwickelt werden. Resiliente adaptive Architekturen müssen Systemmodifikationen und Updates unterstützen, die die Sicherheit auch bei Funktions-Upgrades gewährleisten.

Post-Produktion-Softwareänderungen fallen typischerweise in drei Kategorien [9]: corrective updates für Bugfixes, perfective updates für Performanceverbesserungen und adaptive updates. Adaptive Updates sind der primäre Anwendungsfall des ADSO-Prozesses, da sie das Hinzufügen neuer Funktionalitäten (z. B. Funktion-on-Demand oder neues KI-Modell) oder Anpassungen an veränderte Arbeitsbedingungen ermöglichen.

Beim Einspielen eines Softwareupdates mit neuer Funktionalität – unabhängig davon, ob es vom Fahrer oder vom OEM initiiert wird – gehen wir davon aus, dass: (i) das Fahrzeug mit den notwendigen Sensoren und Aktuatoren ausgestattet ist, einschließlich ausreichender Ressourcen für das Update; (ii) es als softwaredefiniertes Fahrzeug mit spezifizierten APIs für neue Funktionen betrieben wird; und (iii) ein Teil der Softwaremodule sicher über Funkverbindung eingespielt werden kann. Die Integration von Mode-Management-Mechanismen in die Softwarearchitektur, wie in [56] beschrieben und in [92] implementiert, erfüllt die Anforderungen der ISO/TR 4804 [48] und stellt sicher, dass Softwareupdates weder die Performance noch die Sicherheit bestehender Funktionen gefährden; andernfalls wäre eine Fahrzeug-Delta-Zertifizierung erforderlich.

Fahrzeugfunktionen sind für optimale Performance unter vordefinierten Arbeitsbedingungen ausgelegt. Automatisierte Fahrfunktionen müssen sich jedoch an offene und dynamische Umgebungen anpassen, was die Spezifikation dieser Bedingungen erschwert. Das ODD hilft, die Systemfähigkeiten basierend auf dem Kontext (z. B. Straßentyp, Wetter) zu modellieren, der sich im Produktlebenszyklus ändern kann. Resilienz gegenüber Post-Development-



Änderungen wird durch ODD-bewusstes Mode-Management erreicht. Dies ermöglicht sichere Übergänge zwischen Betriebszuständen als Reaktion auf Kontextänderungen, wobei eine ODD-Handling-Schicht unterschiedliche Bedingungen bewertet und der ADS Mode-Manager [48] die Aktivierung oder Deaktivierung von Fahrzeugfunktionen steuert.

Die beschriebene adaptive Softwarearchitektur teilt das interne Fahrzeugverhalten in zwei Systeme: das *managed* System und das *managing* System [91]. Im *managing* System wurden zwei Hauptschichten spezifiziert:

- Die ODD-Handling Layer, die die Betriebsparameter des Fahrzeugs beschreibt und diese zur Laufzeit durch Interpretation von Sensordaten bewertet.
- Die ADS Mode-Manager Komponente, die festlegt, welche Fahrzeugfähigkeiten beim Betreten oder Verlassen eines ODDs gemäß definierten Modi aktiviert oder deaktiviert werden.

Dieser Spezifikationsansatz ermöglicht die Implementierung eines Umschaltmechanismus für ODD-adaptive Fahrzeugfunktionen. Das architektonische Design ist sowohl resilient gegenüber Erweiterungen als auch gegenüber dem Betriebskontext. Bei der Einführung einer neuen Funktion-on-Demand kann die ODD-Handling-Komponente um zusätzliche ODDs erweitert werden. Diese neuen Fähigkeiten werden in bestehende oder neue Aktionslisten integriert, die vom ADS-Mode-Manager verarbeitet werden. Entscheidend ist, dass das Hinzufügen einer neuen Funktion sicherstellt, dass sie im deaktivierten Zustand die Performance bestehender Fahrzeugfunktionen nicht beeinträchtigt, wodurch die Gesamtintegrität des Systems erhalten bleibt.

Im nächsten Absatz wird der ADS Mode-Manager beschrieben. In 4.4.3 wird der ODD-Handler vorgestellt, da diese Komponente der Resilienz gegenüber dem Betriebskontext dient.

ADS Mode-Management

Der in [92] vorgeschlagene ASD Mode-Manager kann als exemplarischer Dienst innerhalb der zuvor eingeführten resilienten, FOCUS-basierten dienstorientierten Architektur gelesen werden. Indem dem ASD Mode-Manager eine eigene syntaktische Schnittstelle $(I^{\mathrm{MM}} \triangleright O^{\mathrm{MM}})$ und Verhaltensfunktion f_{MM} zugewiesen wird, wird die in Abschnitt 5.3 in [26] geforderte Modularität und Fehlerisolation erreicht. Interne Fehler des Managers, wie veraltete Wahrnehmungseingaben oder beschädigte Zustandsvariablen, bleiben auf seine wohldefinierten Kommunikationskanäle beschränkt und können sich nicht auf nicht verwandte Dienste ausbreiten. Diese strikte Trennung der Verantwortlichkeiten ermöglicht auch eine zusammensetzbare Verifikation, die es erlaubt, ein globales Sicherheitsargument in lokale Verträge für den Mode-Manager und seine abhängigen Dienste zu zerlegen.

Der ADS Mode-Manager beobachtet kontinuierlich sowohl den Umwelt- als auch den Fahrzeugzustand, bewertet diese anhand des deklarierten ODD und bestimmt, ob im *manuellen*, *nominalen Automatisierungs-* oder *degradierten Fallback*-Modus gearbeitet werden soll. Dieser Mechanismus unterstützt direkt die in der Grundlage diskutierten Aspekte der Dienstebene-Überwachbarkeit und Gesundheitsüberwachung. Abbildung 4.9 illustriert beispielhaft die Modusübergangslogik in einem Fahrzeug mit zwei automatisierten Fahrzeugfunktionen:

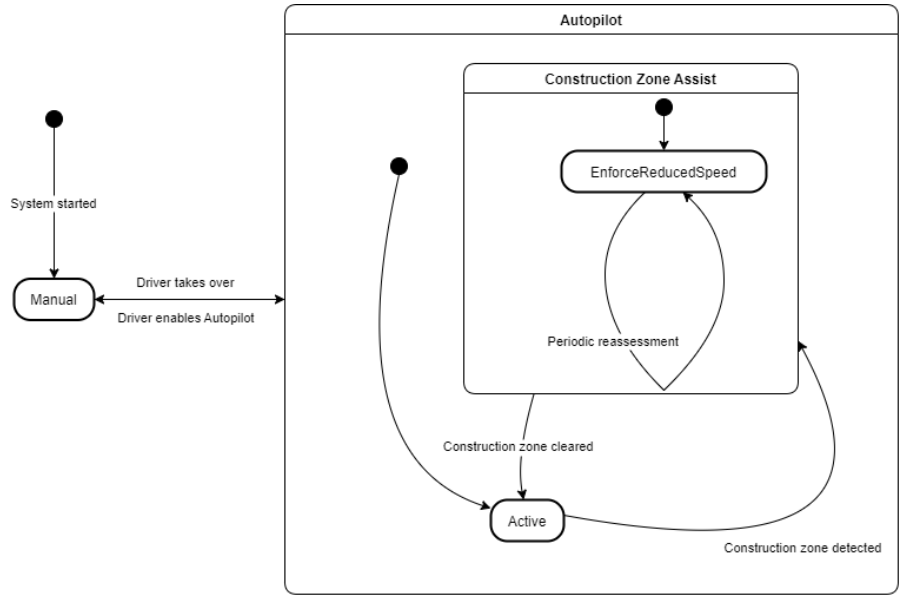


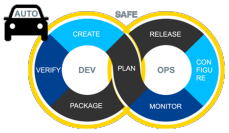
Abbildung 4.9: ADS Mode-Manager-Spezifikation für Baustellenassistent.

- Autopilot
- Baustellenassistent (CZA)

Einerseits aktiviert/deaktiviert der Fahrer die Autopilot-Funktion und wechselt zwischen zwei möglichen Modi: *MANUAL_MODE*, *AUTONOMOUS_MODE*. Wenn sich das Fahrzeug im *Autopilot*-Modus befindet, wird die CZA-Funktion vom ADS Mode-Manager aktiviert, wenn der spezifizierte ODD betreten wird. Innerhalb dieses Modus erzwingt das System eine reduzierte Geschwindigkeit und bewertet die Umweltbedingungen regelmäßig neu, um zu erkennen, wann der ODD verlassen wurde. Wenn die Baustelle geräumt ist, wechselt das System zurück zum regulären Autopilot. In diesem speziellen Fall ist das Fallback-Manöver eine Übergabe an die manuelle Steuerung, was implizit im Diagramm modelliert ist.

In FOCUS-Begriffen ist die semantische Schnittstelle f_{MM} deterministisch und nebenwirkungsfrei in Bezug auf ihre Eingaben, was sie für die Laufzeitvalidierung und Rückverfolgbarkeit geeignet macht. Monitore können den Eingabestrom wiedergeben, um Übergänge zwischen Betriebszuständen deterministisch zu reproduzieren, und bieten eine klare Prüfspur von Umwelteingaben zu Moduswechselentscheidungen, wie von ISO TR 4804 [48] gefordert. Darüber hinaus können externe Entitäten bei Abweichungen, z. B. fehlenden Fahrbahnmarkierungen, die den ODD beeinträchtigen, über dieselbe formale Schnittstelle Abschalt- oder Übernahmebefehle erteilen, wodurch ein kontrollierter Abbau und ein sicheres Fallback ermöglicht werden.

Durch die Integration des Managers in ein dynamisches Dienstregister unterstützt die Architektur das Lebenszyklusmanagement und die fehlertolerante Anpassung. Mehrere Implementierungen, wie ein Python-Prototyp oder eine eingeschränkte AUTOSAR Adaptive-Komponente, können unter einer gemeinsamen abstrakten Schnittstelle veröffentlicht wer-



den. Wenn eine Instanz ausfällt oder ihren Vertrag verletzt, widerruft das Register sie und fördert nahtlos einen alternativen Anbieter. Diese Fähigkeit erfüllt die in Abschnitt 3.1 in [26] beschriebenen Anforderungen an die Anpassungsfähigkeit und gewährleistet eine Echtzeit-Neukonfiguration mit minimaler Unterbrechung. Darüber hinaus ermöglicht dieselbe Dienstinfrastruktur die Weiterentwicklung nach der Bereitstellung: Korrektive, perfekte oder adaptive Updates können verbesserte Entscheidungslogik oder erweitertes ODD-Handling einführen, ohne bestehende Schnittstellen ungültig zu machen. Vorausgesetzt, die aktualisierte Implementierung entspricht dem ursprünglichen syntaktischen Vertrag und besteht abgeleitete Konformitätstests, bewahrt das System die Zertifizierungsgrenzen und minimiert die Notwendigkeit von Delta-Bewertungen.

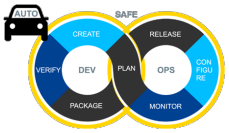
Skalierbarkeit ist ebenfalls in das Design eingebettet. Neue Betriebsmodi oder erweiterte ODD-Interpretationen können als konservative Verfeinerungen der semantischen Funktion f_{MM} realisiert werden. Da f_{MM} Eingabeströme auf Mengen zulässiger Ausgabeströme abbildet, vermeiden Erweiterungen, die das bestehende Verhalten beibehalten, die erneute Zertifizierung nicht verwandter Dienste. Zusätzliche Perzeptions- oder Lokalisierungskomponenten, die reichhaltigere Kontextinformationen bieten, werden über neue Eingabekanäle in I^{MM} aufgenommen, während ältere Bereitstellungen mit früheren Schnittstellenversionen kompatibel bleiben. Diese prinzipielle Weiterentwicklung unterstützt die schrittweise Hinzufügung neuer Fahrzeugfähigkeiten und erfüllt die Anforderungen an das funktionale Erweiterung, die in Abschnitt 3.2 in [26] beschrieben sind.

4.4.3 Resilienz gegenüber dem Betriebskontext

Die Resilienz in unterschiedlichen Betriebskontexten bleibt eine grundlegende Herausforderung für automatisierte Fahrsysteme. Fahrzeuge im realen Straßenverkehr müssen sich kontinuierlich an Unsicherheiten anpassen, die durch Sensorausfälle, unvorhersehbares Verhalten anderer Verkehrsteilnehmer und dynamische Umweltbedingungen entstehen. Über die Unsicherheit hinaus müssen automatisierte Fahrzeuge auch die Vielfalt offener Fahrszenarien bewältigen, was Mechanismen zur funktionalen Rekonfiguration und skalierbaren Adaption erfordert.

Robustes Laufzeitmonitoring ist entscheidend, um die Integrität und Sicherheit von Fahrzeugen im Betrieb sicherzustellen. Monitoring-Komponenten übernehmen dabei verschiedene Rollen. Zunächst stellen sie sicher, dass die grundlegenden Annahmen, die während der Designphase getroffen wurden, über den gesamten Lebenszyklus hinweg gültig bleiben. In einem zweiten Schritt werden diese Annahmen systematisch auf spezifische Machine-Learning-Komponenten heruntergebrochen, häufig in Form von Metriken, die potenzielle Risiken zur Laufzeit anzeigen. Monitoring von Machine-Learning-Komponenten ist entscheidend, da es kritische Einblicke in die Performance und den Zustand des Systems bietet.

Wenn Schlüsselindikatoren, die aus Sicherheitsannahmen oder Metriken abgeleitet wurden, ein Ereignis auslösen, kann dies den Bedarf an einem Softwareupdate signalisieren. Dennoch sind zusätzliche Maßnahmen erforderlich, um die Resilienz gegenüber dem Betriebskontext zu erhöhen und robuste, anpassungsfähige Systeme für das automatisierte Fahren zu ermöglichen. Das AutoDevSafeOps-Projekt hat dazu Lösungsansätze entwickelt, die die Systemrobustheit durch den Einsatz vielfältiger Sensorkonfigurationen, Ensemble Learning, adversarial Scenario Testing und dynamische Systemadaption verbessern.



Aufbauend auf der formalen SOA-Spezifikation aus Abschnitt 4.4.1 spezifizieren und synthetisieren wir Laufzeitmonitore auf Grundlage des contract-driven runtime monitor generation-Ansatzes aus [76]. Diese Monitore sind darauf ausgelegt, sicherzustellen, dass die tatsächlichen Laufzeitinteraktionen und Datenflüsse mit der intendierten Architektur und den Serviceverträgen übereinstimmen. Das Hauptziel ist es, funktionale Korrektheit und Betriebskonsistenz auch bei dynamisch integrierten oder sich entwickelnden Komponenten sicherzustellen.

Im nächsten Absatz wird der ODD-Handler beschrieben, der die Resilienz gegenüber dem Betriebskontext verwaltet und zusammen mit dem ADS-Mode-Manager einen Umschaltmechanismus für ODD-adaptive Fahrzeugfunktionen implementiert.

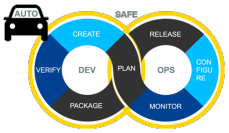
ODD Handler

Der ODD-Handler spielt eine grundlegende Rolle bei der Realisierung kontextbewusster Anpassungsfähigkeit innerhalb robuster Automobilarchitekturen. Er ist dafür verantwortlich, Umweltsignale und betriebliche Einschränkungen in eine explizite ODD-Klassifikation zu interpretieren, die das Verhalten höherer Systemfunktionen informiert. Funktional bewertet der ODD-Handler Laufzeitdaten anhand eines strukturierten Modells betrieblicher Annahmen, wodurch das System beurteilen kann, ob die aktuellen Fahrbedingungen mit seinen zertifizierten Fähigkeiten übereinstimmen.

Konzeptionell fungiert der ODD-Handler als Vermittlungsschicht zwischen Wahrnehmungseingaben und Modusentscheidungslogik. Er synthetisiert verschiedene Datenquellen in semantisch sinnvolle ODD-Zustände, wie z. B. die Identifizierung, ob sich das Fahrzeug in einer Baustelle befindet, ungewöhnlichen Straßenschildern ausgesetzt ist oder vorübergehenden Änderungen der Infrastruktur unterliegt. Durch die kontinuierliche Überwachung dieser kontextuellen Hinweise ermöglicht der ODD-Handler nachgelagerten Komponenten, entsprechend dem Betriebsrahmen zu reagieren, sei es durch Moduswechsel, Einleitung eines Fallbacks oder Anpassung der Funktionalität. Zum Beispiel würde der ODD-Handler beim Erkennen von Verkehrspylonen, die eine Baustelle abgrenzen, eine *vorübergehende Straßenstruktur* als Teil des aktuellen Betriebsbereichs auslösen. Das Folgende zeigt, wie dies gemäß ISO 34503 modelliert werden kann.

Innerhalb des zuvor eingeführten serviceorientierten Architekturparadigmas, wie in 4.4.1 beschrieben, dient der ODD-Handler der Modularität, Fehlerisolation und einem schnittstellengetriebenen Design. Er stellt eine klar definierte syntaktische Schnittstelle zum Empfangen von Wahrnehmungs- oder Infrastrukturdaten bereit sowie eine semantische Schnittstelle, die diese Eingaben in strukturierte ODD-Zustände abbildet. Dieses Design ermöglicht es, unabhängig von spezifischen Sensorkonfigurationen oder Fahrzeugplattformen zu bleiben, wodurch der ODD-Handler als wiederverwendbarer und erweiterbarer Dienst in verschiedenen Systeminstanzen fungieren kann. Seine Ausgaben können von Mode-Management-Komponenten, Sicherheitsmonitoren oder Entscheidungsschichten genutzt werden, ohne dass eine enge Kopplung erforderlich ist. Dadurch werden architektonische Ziele wie lose Abhängigkeiten und eine skalierbare Integration erreicht.

Die Rolle des ODD-Handler ist besonders kritisch, um die Herausforderungen dynamischer und offener Umgebungen zu bewältigen, wie in Abschnitt 5.5 in [26] diskutiert. Automatisierte Fahrsysteme arbeiten unter variierenden Straßenlayouts, Wetterbedingungen, Verkehrs-



verhalten und sich entwickelnder Infrastruktur. Der ODD-Handler unterstützt die Resilienz, indem er formalisiert, wie solche kontextuellen Variabilitäten erkannt und darauf reagiert wird. Dies umfasst nicht nur das Aktivieren oder Deaktivieren von Systemfähigkeiten basierend auf vordefinierten ODD-Grenzen, sondern auch die Unterstützung von Reklassifizierungs- und Minderungsstrategien bei Umweltabweichungen. Somit trägt der ODD-Handler sowohl zur Robustheit als auch zum kontrollierten Abbau bei und leistet damit einen Beitrag zur Gesamtsicherheit der autonomen Funktion.

4.5 Erweiterung des Sicherheitslebenszyklus zur Gewährleistung der Vertrauenswürdigkeit von KI-Funktionen über den gesamten Produktlebenszyklus hinweg

Die Integration von KI-Komponenten in sicherheitskritische Automobilsysteme bringt neue Herausforderungen mit sich, insbesondere in Bezug auf Zuverlässigkeit, Robustheit und regulatorische Konformität über den gesamten Lebenszyklus eines Fahrzeugs hinweg. Traditionelle Sicherheitsprozesse müssen angepasst und erweitert werden, um die inhärenten Unsicherheiten KI-basierter Funktionen zu adressieren. Ein umfassender KI-Sicherheitslebenszyklus muss nicht nur Spezifikation und Verifikation abdecken, sondern auch kontinuierliches Monitoring und Anpassung im Betrieb berücksichtigen.

Dieser Abschnitt beleuchtet zentrale Methoden zur Erweiterung des KI Sicherheitslebenszyklus, wie sie im ADSO-Projekt entwickelt wurden. Er behandelt kritische Aspekte wie fehlende Spezifikation von KI-Funktionalität und Verhalten, Sicherheitsargumentation, rigoroses Testen, Datensatzspezifikation, Trainingsprozesse sowie Erkennung und Umgang mit Data Drift. Durch die Integration strukturierter Sicherheitsnachweisansätze schaffen wir eine Grundlage für die Vertrauenswürdigkeit von KI-Funktionen – von der Entwicklung bis zum realen Einsatz und langfristigen Betrieb.

4.5.1 Modulare Neuronale Netze

Eine der größten Herausforderungen bei KI in sicherheitskritischen Systemen ist das Fehlen einer expliziten Spezifikation, wie genau eine Aufgabe gelöst werden soll. Dadurch wird Interpretierbarkeit weithin als grundlegendes Merkmal angesehen, um Vertrauen in Deep Learning-Systeme aufzubauen, insbesondere in sicherheitskritischen Bereichen [81, 80]. Um den Mangel an Transparenz von Deep Neural Network-Modellen zu adressieren, wurden verschiedene Techniken vorgeschlagen, um deren interne Entscheidungsfindung besser zu erklären und zu verstehen [67]. Unter diesen bieten konzeptbasierte Ansätze eine ganzheitliche Sichtweise, indem sie das Modellverhalten in Bezug auf für Menschen verständliche Konzepte ausdrücken.

Ein bemerkenswertes Beispiel ist die Concept Bottleneck Model-Architektur [53]. Concept Bottleneck Models führen eine modulare Architektur ein, in der eine interpretierbare Zwischenschicht Eingaben auf eine Menge vordefinierter, bedeutungsvoller Konzepte (z. B. „Form“ oder „Farbe“) abbildet, bevor die endgültige Vorhersage erfolgt. Wie in Abb. 4.10 zu sehen ist, wird bei Concept Bottleneck Models der Prozess in zwei Stufen unterteilt, anstatt Eingaben $x \in X$

direkt auf Labels $y \in Y$ abzubilden: ein Konzeptprädiktor $g : X \to C$, der Eingaben auf einen binären Konzeptraum $\tilde{\mathbf{c}} \in \mathbb{B}^k$ abbildet, und ein Labelprädiktor $f : \mathbb{B}^k \to \mathbb{R}$, der Konzepte auf Ausgaben abbildet. Der Datensatz $\mathcal{D}$ besteht somit aus Tupeln $(x, \tilde{\mathbf{c}}, y)$, wobei Konzeptannotationen für jede Eingabe bereitgestellt werden. Die Beziehung zwischen den Konzepten und der endgültigen Ausgabe wird so modelliert, dass das Verständnis der vom Modell getroffenen Entscheidungen gefördert wird.

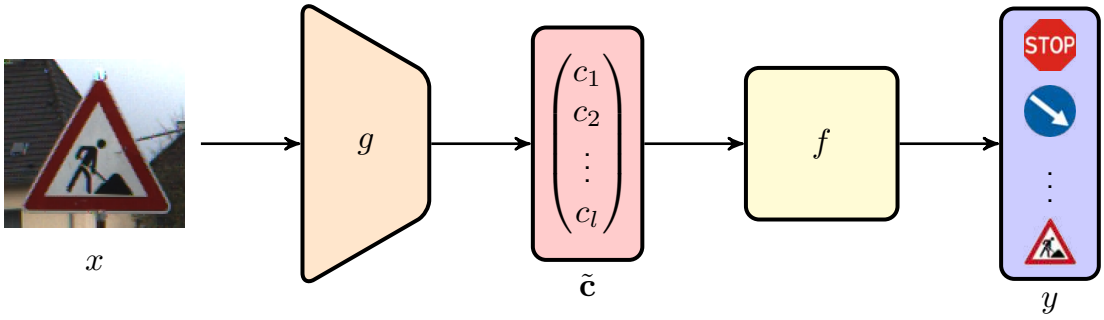


Abbildung 4.10: Architektur des Concept Bottleneck Models von Koh et al. [53], angepasst an die Aufgabe der Verkehrsschilderklassifikation: Die Eingabe x, bestehend aus Verkehrsschildbildern, wird in ein Convulutional Neural Network-Backbone g eingespeist, das die niedrigstufigen Konzepte in Form eines Konzeptvektors $\tilde{\mathbf{c}}$ vorhersagt, gefolgt von einem vollständig verbundenen Netzwerk f, das die Klasse der Verkehrsschilder vorhersagt und die endgültige Ausgabe y liefert.

Interpretierbarkeit wird besonders wertvoll, wenn es darum geht, Deep Neural Network-Modelle zu diagnostizieren und zu verbessern. In sicherheitskritischen Umgebungen ist die Absicherung ein fortlaufender, iterativer Prozess: Wenn ein Modell die Sicherheitsanforderungen nicht erfüllt, ist es entscheidend zu verstehen, *warum* es versagt hat. Traditionell beginnt die Fehleranalyse mit der Identifizierung von Eingaben, die zu falschen Ausgaben führen, wie Falsche Positive oder Falsche Negative im Binären Fall. Standardleistungsmetriken vermitteln jedoch nicht die Schwere einer Fehlklassifikation oder geben Einblicke in deren zugrunde liegende Ursache. Darüber hinaus wird der Vergleich von Modellfehlern, die durch unterschiedliche Eingaben verursacht wurden und als Falsche Positive oder Falsche Negative identifiziert wurden, ohne zusätzliche Einblicke zu einer mühsamen Aufgabe. Im Kontext von Concept Bottleneck Models können wir den interpretierbaren Konzeptraum für eine detailliertere Fehleranalyse nutzen.

Um zu quantifizieren, wie weit die Konzeptvorhersage eines Modells von der Ground Truth abweicht, führen wir die Hamming-Distanz als sicherheitsrelevante Metrik ein [33]. Sie misst die Anzahl der nicht übereinstimmenden Bits zwischen den vorhergesagten und den tatsächlichen Konzeptvektoren:

Definition 7 (Hamming-Distanz). Seien $\tilde{\mathbf{c}_1}$ und $\tilde{\mathbf{c}_2}$ zwei Konzeptvektoren im Konzeptraum

$\mathbb{C}$, dann ist die *Hamming-Distanz* h zwischen ihnen definiert als:

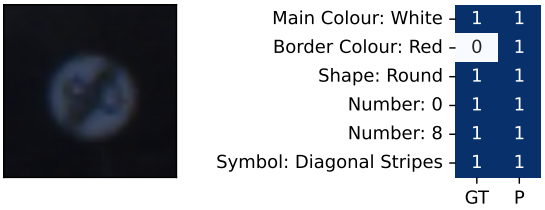
$$h(\tilde{\mathbf{c_1}}, \tilde{\mathbf{c_1}}) = \sum_{i=1}^{l} [\tilde{\mathbf{c_{1_i}}} \veebar \tilde{\mathbf{c_{2_i}}}],$$

wobei $[\mathbf{a} \veebar \mathbf{b}]$ die bitweise XOR-Operation zwischen den i-ten Elementen a_i und b_i der Vektoren $\mathbf{a}$ und $\mathbf{b}$ darstellt.

Höhere Hamming-Distanzen weisen auf größere Abweichungen vom erwarteten Verhalten hin und können Proben hervorheben, bei denen das Modell schlecht abschneidet. Dies ermöglicht eine sicherheitskritische Bewertung, entweder innerhalb jeder Klasse oder über gesamte Datensätze hinweg, und hilft, die kritischsten Fehlerfälle für die Untersuchung zu priorisieren.

Darüber hinaus bietet der vorhergesagte Konzeptvektor tiefere Einblicke in die Ursachen von Fehlvorhersagen. Sobald problematische Proben priorisiert wurden, wird es möglich, zu analysieren, welche spezifischen Konzepte falsche Ausgaben verursachen, welche konsequent falsch vorhergesagt werden und wie sie die endgültige Entscheidung beeinflussen. Diese detaillierten Einblicke sind entscheidend, um gezielte Maßnahmen zu ergreifen, wie z. B. das erneute Training von unterdurchschnittlich abschneidenden Konzeptprädiktoren oder die Überarbeitung von Konzeptdefinitionen, was letztendlich die Sicherheit und Robustheit des Modells in zukünftigen Iterationen stärkt.

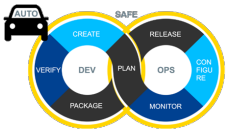
(a) Beispiel einer schwerwiegenden Fehlklassifikation, die mithilfe der Hamming-Distanz schnell identifiziert werden konnte.



(b) Beispiel einer korrekten Klassifikation, die jedoch auf einem falschen Satz vorhergesagter Konzepte basiert.

Abbildung 4.11: Beispiele für zusätzliche Informationen, die von Concept Bottleneck Models in einer Verkehrsschilderklassifikationsaufgabe bereitgestellt werden.

Abbildung 4.11 zeigt zwei Vorhersagebeispiele eines Concept Bottleneck Model, das auf dem German Traffic Sign Recognition Benchmark-Datensatz [82] trainiert wurde. In jeder Teilabbildung wird das Eingabebild x (linke Seite) zunächst vom Konzeptprädiktor verarbeitet, der das Vorhandensein oder Fehlen von 43 vordefinierten Konzepten bestimmt, darunter *Hauptfarbe: Weiß*, *Form: Rund* und *Symbol: Diagonale Streifen* usw. Dieser Prozess erzeugt einen binären Konzeptvektor $\tilde{\mathbf{c}}$, der auf der rechten Seite jeder Teilabbildung dargestellt ist. Zur Klarheit wird der Konzeptvektor in der Abbildung gekürzt, um nur eine Teilmenge der Konzepte anzuzeigen. Der Vektor $\tilde{\mathbf{c}}$ wird dann an den Labelprädiktor übergeben, der die endgültige Vorhersage des Modells ausgibt (unten in jeder Teilabbildung angezeigt).



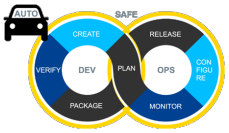
In Teilabbildung 4.11a führt die Vorhersage des Modells zu einer Hamming-Distanz von 7, was darauf hinweist, dass 7 Konzepte falsch vorhergesagt wurden. Dies unterstreicht den Mehrwert der Konzeptebene-Interpretierbarkeit: Anstatt die Ausgabe einfach als falsch zu markieren, erhalten wir Einblicke in das Ausmaß und die Art des Fehlers. In diesem Fall zeigt eine genauere Untersuchung, dass das Modell aufgrund der schlechten visuellen Qualität des Eingabebildes keine Konzepte korrekt identifizieren konnte.

Im Gegensatz dazu zeigt Teilabbildung 4.11b eine erfolgreiche Vorhersage trotz einiger Fehler auf der Konzeptebene. Hier wird das Schild korrekt klassifiziert, aber einige Konzepte (z. B. *Randfarbe: Rot*) werden falsch vorhergesagt. Diese Diskrepanz könnte auf den dunklen Hintergrund zurückzuführen sein, der das Modell dazu veranlasst, fälschlicherweise einen roten Rand zu erkennen, wo keiner existiert. Sowohl die Hamming-Distanz als auch die Konzept basierte Erkennung adressieren das Problem das es keine Spezifikation für das Verhalten von Neuronalen-Netzen gibt. Sowohl die Hamming-Distanz als auch die verbesserte Fehleranalyse können dabei direkt zu einer besseren Sicherheits-Argumentation beitragen, wie wir in [34] gezeigt haben.

Die Aufrechterhaltung von Leistung und Zuverlässigkeit in dynamischen Umgebungen erfordert jedoch die Auseinandersetzung mit weiteren Herausforderungen – allen voran **Data Drift**, eine der größten Herausforderungen beim Einsatz von Computer-Vision-(CV)-Modellen in sich verändernden Umgebungen. Data Drift bezeichnet systematische Veränderungen in den statistischen Eigenschaften und Merkmalen des Eingabebereichs über die Zeit hinweg, was die Leistung des CV-Modells erheblich beeinträchtigen kann [44].

Zur genaueren Untersuchung von Data Drift wurde ein Computer-Vision-Modell auf Basis von YOLO entwickelt und mit augmentierten Drift-Daten getestet. Der Trainingsdatensatz wurde mit Hilfe von *Car Learning to Act (CARLA)* [78] generiert. CARLA ermöglicht die Erstellung verschiedenster Szenarien – einschließlich solcher, die in der Realität schwer oder gefährlich nachzustellen sind, wie etwa extreme Wetterbedingungen oder komplexe Verkehrssituationen [51]. Dadurch wird die Robustheit der erzeugten Daten direkt erhöht, was sie für reale Anwendungen besonders geeignet macht.

Die Erfassung von Bilddaten aus CARLA erfolgte in zwei Phasen. Zunächst wurden Bilder typischer Straßensituationen aus verschiedenen städtischen und ländlichen Karten gesammelt, um einen Basis-Bilddatensatz zu erstellen. Anschließend wurden in der zweiten Phase diese Karten Drift-Umgebungen unterzogen, um einen „gedrifteten“ Datensatz zu erzeugen. Dadurch entstanden zwei Datensätze: einer ohne Drift-Daten und einer mit Drift-Daten. Die auf dem Datensatz ohne Drift trainierten Modelle wurden anschließend auf dem Datensatz mit Drift getestet. Wie in [44] hervorgehoben, kam es zu einem Leistungsabfall von fast 40 %, wenn das Modell nicht unter Driftbedingungen trainiert wurde, um Verkehrsschilder zu erkennen. Durch nachträgliches Training des Modells mit dem augmentierten Datensatz konnte die Genauigkeit auf über 90 % gesteigert werden. In einer weiteren Arbeit [4] haben wir zudem die Bedeutung des CV-Modells im ODD-Kontext von Baustellen hervorgehoben, indem ein synthetischer Datensatz mit Objekten wie Kegeln, Balken und Absperrungen erzeugt wurde. Auch dieses Modell erzielte bei Tests unter Driftbedingungen eine Genauigkeit von über 90 %. Diese Ergebnisse unterstreichen die kritische Auswirkung von Data Drift sowie die Wirksamkeit driftbewusster Trainingsstrategien zur Verbesserung von Bildverarbeitungsmodellen für sicherheitskritische Fahrzeugsysteme.



4.5.2 Data Drift Erkennung mit Zeitreihen Daten

Daten-Drifts beschränken sich nicht nur auf Bilddaten – sie können ebenso in Zeitreihendaten auftreten, ein Aspekt, der ebenfalls im ADSO-Projekt untersucht wurde. Die untersuchten Anwendungsfälle unterstreichen die Bedeutung einer effizienten Überwachung von ML-Modellen zur Laufzeit oder regelmäßig offline, um derartige Drifts frühzeitig zu erkennen, die auf sicherheitskritische Updates hinweisen können. Im Zusammenhang mit Data Drift in Zeitreihendaten, wie z. B. Sensordaten, war es das Hauptziel von MUL, regelmäßig Datenqualitätsprüfungen durchzuführen, indem das Problem der Datendrift-Erkennung gelöst wurde. Zu diesem Zweck wurden im ADSO-Projekt drei zentrale Technologiebausteine (Technology Bricks, TBs) implementiert, die im Folgenden aufgeführt sind:

- **Data Drift Injection** ist ein Werkzeug zur synthetischen Injektion von Daten-Drifts in einen Zeitreihendatensatz. Es dient der Simulation gedrifteter Bedingungen in Zeitreihen, z. B. durch Hardwarefehler wie Bit-Flips, Sensorausfälle usw.
- **Data Drift Detection** ist ein Tool, das aus einer Sammlung trainierter Algorithmen besteht, darunter: a) Confidence Distribution Batch Detection (CDBD) [68], b) Hellinger Distance Drift Detection Method (HDDDM) [35], c) PCA-basierte Änderungsdetektion (PCA-CD) [77], d) kdq-Tree Detection Method (kdq-Tree) [32], sowie e) Nearest Neighbor Density Variation Identification (NN-DVI) [69], die in der Lage sind, Daten-Drifts in Zeitreihendatensätzen zu erkennen.
- **Data Drift Analytics** ist ein Werkzeug zur Visualisierung der Ergebnisse des Data Drift Detection Tools in einem Grafana-Dashboard. Dargestellt werden z. B. erkannte Drifts, die Ground Truth der zugrunde liegenden Drifts sowie geschätzte KPIs für jeden Detektionsalgorithmus.

Zunächst einmal ist DDI ein Hilfsdienst, der uns dabei unterstützt, eine bestimmte Anzahl von Drifts mit bekannter Ground Truth (GT) synthetisch einzufügen, wenn wir einen Datensatz als Eingabe mit unbekannten oder keinen Drifts aus einem realen oder synthetischen Szenario erhalten. DDI ist ein wichtiger und nützlicher Dienst für die Trainings- und Bewertungsprozesse des nächsten TB, des DDD, der aus fünf bekannten Techniken zur Erkennung von Datenabweichungen besteht, wie zuvor aufgeführt. Diese zuvor erwähnten DDD-Methoden werden als unsere ML-Modelle von Interesse angesehen, die überwacht werden sollten, wenn ihre Leistung nachlässt, damit Maßnahmen ergriffen werden können, um diese DDD-Algorithmen neu zu trainieren. Sobald DDD eine der zugrunde liegenden Drifts erkannt hat, wird der DDA-Dienst aufgerufen, um die erkannten Drifts zusammen mit den berechneten KPIs pro Methode und der aktuell pro Methode verwendeten Parametrisierung zu visualisieren.

Für jeden DDD-Algorithmus werden zwei Hauptpipelines definiert: a) die Trainingsphase und b) die Testphase, wie in Abbildung 4.12 dargestellt. In der Trainingsphase werden mit Hilfe der Carla-Simulation generierte Zeitreihendatensätze verwendet, um jeden DDD-Algorithmus zu trainieren. In der Testphase kann jeder beliebige Zeitreihendatensatz, entweder aus Carla oder aus einem realen Szenario, in die Pipeline eingegeben werden, um getestet zu werden.

Abbildung 4.13 zeigt die kombinierten Trainings- und Testphasen des DDD-Tools in einem Flussdiagramm.

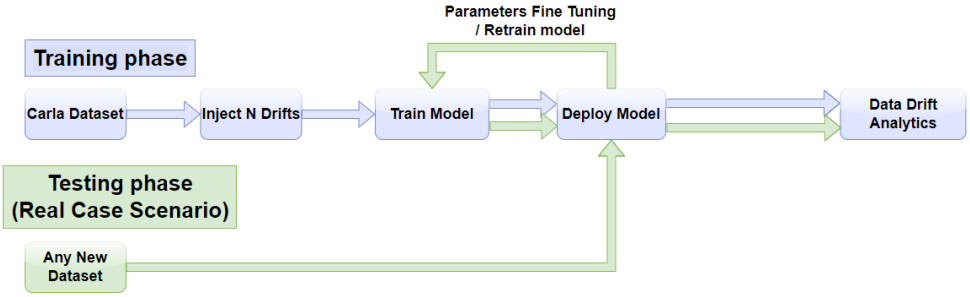


Abbildung 4.12: Zusammenfassung Flussdiagramme der Trainings- und Testphasen des Tools zur Erkennung von Datenabweichungen.

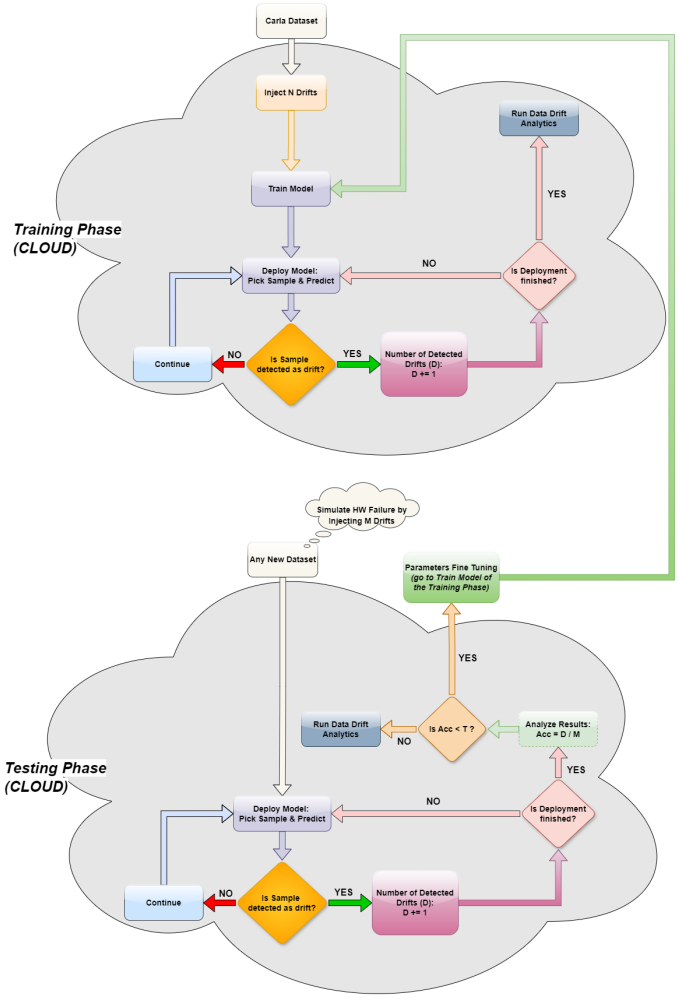
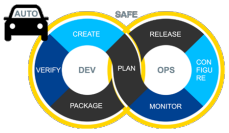


Abbildung 4.13: Detaillierte Flussdiagramme der kombinierten Trainings- und Testphasen des Tools zur Erkennung von Datenabweichungen.



Abschließend erfordert die sichere Integration und Aktualisierung von KI-Komponenten systematische Verifikationstechniken. Insbesondere werden in der Industrie Methoden zur Testfallegenerierung benötigt, um die Qualität und Robustheit dieser lernenden Komponenten zu bewerten. Im ADSO-Projekt haben wir uns gezielt auf automatisiertes Robustheitstesten konzentriert und den Eingaberaum systematisch untersucht, um adversarielle Beispiele zu finden. Ein adversarielles Beispiel ist eine kleine Störung einer Eingabe einer lernenden Komponente, die zu einem völlig anderen Ergebnis dieser Komponente führt – z. B. einer anderen Klassifikation. Das bedeutet, dass die lernende Komponente gegenüber solchen kleinen Störungen nicht robust ist.

Wir haben in zwei Domänen gearbeitet: der Bildklassifikation [52] und Black-Box-Spracherkennungssystemen [17]. Dabei haben wir evolutionäre Testfallegenerierungstechniken und das Konzept der übertragbaren adversariellen Beispiele verwendet, um solche kleinen Störungen gezielt zu finden.

4.6 Optimierung des Update-Prozesses durch frühe Validierung von Software-Updates in einer simulierten Umgebung und frühzeitige Ausführung im Schattenmodus

4.6.1 Adaptive Updates von Fahrzeugfunktionen

In sicherheitskritischen Systemen, insbesondere solchen mit automatisierten Fahrfunktionen, erfordert Updates von Softwarekomponenten eine strenge Validierung vor der Bereitstellung sowie streng kontrollierte Integrationsprozesse. Dies ist besonders relevant bei der Bereitstellung neuer oder modifizierter Funktionen, den sogenannten adaptiven Updates [9], unter operativen Einschränkungen.

Das Hinzufügen neuer Funktionalitäten, wie beispielsweise Funktion-on-Demand oder eines neuen KI-Modells, wurde durch die Integration der automatisierten Funktion „Construction Zone Assist“ (CZA) in die ADORe CI/CD-Pipeline validiert. Der CZA wurde als neue sicherheitskritische Softwarekomponente entwickelt. In einem ersten Schritt wurden die aus der STPA abgeleiteten Sicherheitsanforderungen zur Laufzeit mithilfe von APIKS [92] verifiziert. Zusätzlich wurde eine kontinuierliche Überwachung von Schlüsselmetriken implementiert, um die allgemeine Sicherheitsleistung zu verbessern.

Abbildung 4.14 zeigt, wie das virtuelle ADORe-Fahrzeug die gefahrene Trajektorie während der Fahrt durch eine Baustelle anpasst. Um kontrollierte adaptive Updates zu ermöglichen, wurde ein umfassender Validierungs- und Bereitstellungsansatz eingeführt [30]. Dieser Ansatz integriert Testautomatisierung, Vertragsverifizierung und Laufzeitkompatibilitätsprüfungen in eine strukturierte Pipeline, die speziell für die Bewertung und Installation von Softwaremodulen innerhalb der ADORe-Fahrzeugplattform entwickelt wurde.

Insgesamt erleichtert dieser Integrationsmechanismus strukturierte und sicherheitsorientierte Verbesserungen von Fahrzeugfunktionen in Automobilplattformen wie ADORe. Durch die Einbettung von Validierung, Zertifizierung und Integrationssicherung in die Softwarebereitstellungspipeline unterstützt er direkt das Ziel von ADSO, sichere und zuverlässige modulare

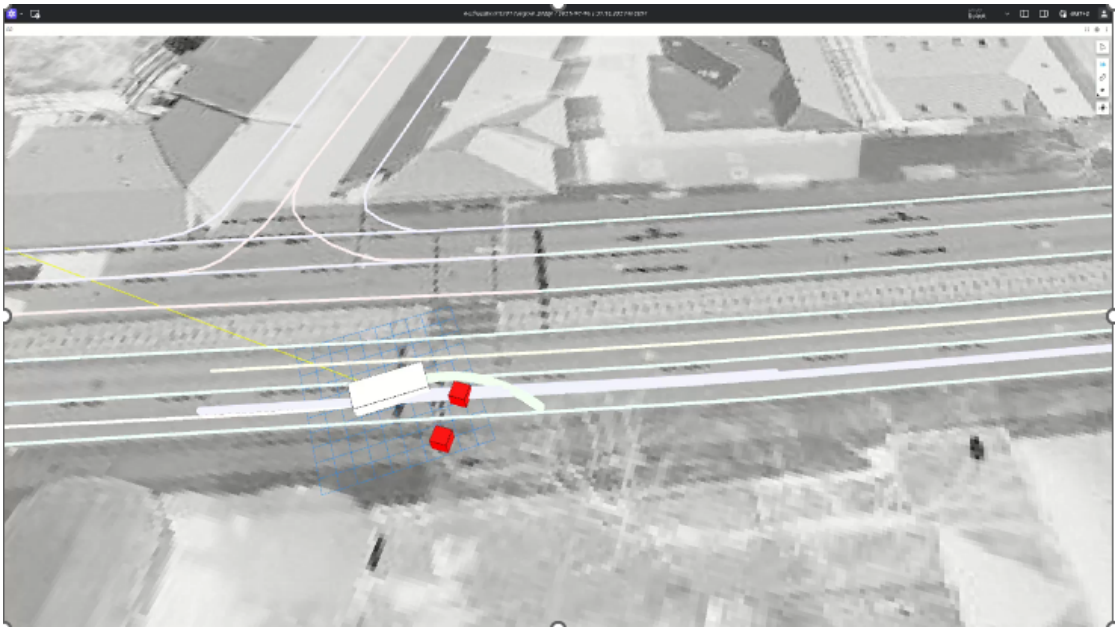
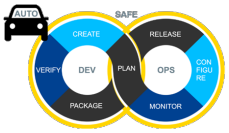


Abbildung 4.14: Trajektorienplaner für CZA, der in ADORe implementiert wurde.

Updates zu ermöglichen.

4.6.2 Virtueller Integrationstest für Timing und Ressourcen mittels Simulation

Um das korrekte zeitlicher Verhalten eines Updates sicherzustellen wird im Rahmen von UC3.2 eine Simulation des Zeitverhaltens der veränderten Funktionalität im Zusammenspiel mit den unveränderten Teilen des Systems vorgeschlagen. Hierfür werden Modellartefakten aus der Entwicklungsphase mit gemessenen Traces aus dem Fahrzeugbetrieb zu einem Simulationsmodell zusammengeführt. Dadurch werden implizite Projektion in den Artefakten aus der Entwicklung mit Messergebnisse aus der Praxis abgeglichen. Auf Basis der so verbesserten Modelle kann dann die Auswirkungen des Updates auf das Zeitverhalten des Gesamtsystem simulativ verifiziert werden. Eine Herausforderung bei der Extraktion von Modellinformationen aus gemessenen Traces liegt in der Berücksichtigung von beobachteten Korrelationen zwischen Modellelementen. Dabei kann es sich beispielsweise um Auswirkungen von Variablen auf das Laufzeitverhalten oder von Funktionsausführungen auf die Ausführung und die Laufzeit von Funktionen der Middleware handeln. Die Berücksichtigung solcher Korrelation im erzeugten Simulationsmodell haben Auswirkungen auf die Qualität des virtuellen Integrationstests des Updates. Im Projekt wurden Methoden entwickelt um mithilfe von statistischen Verfahren Korrelationen zu identifizieren und auch ihre Signifikanz auf Basis der verfügbaren Messdaten zu bestimmen. Ein wesentlicher Aspekt ist die Auswirkung des Updates auf das zeitliche Ende-zu-Ende Verhalten der zugrundeliegenden beobachtbaren Funktionalität (User Functions). Dabei kann das Update nur einzelne Teil der Wirkketten umfassen, wel-



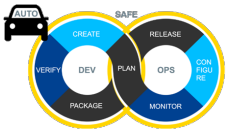
che die jeweilige Funktionalität realisieren. Hierfür wurde im Projekt mit den EventGraphen erweiterte Wirkkettenmodelle mit komplexen Anforderungen an die Ausführungsreihenfolge für die benötigten Timingmodelle entwickelt. Der EventGraph ist so angelegt, dass er auch auf die Analyse von messbasierten Traces angewendet werden kann, unabhängig von der Existenz eines Timingmodells welches den Trace vollständig erfasst. Für die Anwendung auf den Demonstrator in UC3.2 wurde weiterhin eine angepasste Tracinglösung für ROS2 und Tracekonvertierung auf Basis von Babeltrace [1] erstellt. Neben der reinen Simulation des Zeitverhaltens mit dem Simulator chronSIM [2], gibt es die Möglichkeit einer erweiterten Simulation, welche auch funktionale Aspekte und die Integration von konkreten Szenarien und ein Umgebungsmodell beinhaltet. Diese kann ebenfalls PC-basiert, also unabhängig von der Ziel-Hardwareplattform erfolgen und dabei auch von anderen Aspekten der Zielarchitektur abstrahieren. Hierfür wurden in UC3.2 Carla als Umgebungssimulator und für die Szenariendefinition verwendet, ADORE als die funktionale Zielimplementierung und chronSIM als Timing-Simulator. Das ADORE-System stellt dabei hauptsächlich eine Trajectorienplanung zur Verfügung welche das korrekte Fahrverhalten des Ego-Fahrzeugs in Carla auf Basis der von Carla erzeugen Sensorwerte steuert. Dabei tauschen ADORE in Carla über eine im Projekt angepasste Carla-ROS-Bridge die benötigten Informationen aus. Vom Timing-Simulator werden in einer Co-Simulation Verzögerungen in diese Kommunikation injiziert um für realistische Antwortzeiten zwischen Messung und Reaktion in Carla zu sorgen [14]. Mit diesen Maßnahmen ist ein ViT-Test möglich, welcher sowohl das funktionale als auch das zeitliche Verhalten umfasst.

4.6.3 Festkommaarithmetik zur Laufzeitoptimierung sicherheitskritischer Softwarekomponenten

Mit der wachsenden Rechenkomplexität automatisierter Fahrfunktionsmodule gewinnt die Reduzierung des rechnerischen Energieverbrauchs zunehmend an Bedeutung. Gleichzeitig bleibt eine echtzeitfähige Verarbeitung unverzichtbar, damit Fahrzeuge in dynamischen Umgebungen innerhalb weniger Millisekunden geeignete Entscheidungen treffen können.

Im Vergleich zur Gleitkommaarithmetik erfordert Festkommaarithmetik einfachere arithmetische Operationen und benötigt weniger Logikgatter und Transistoren. Dadurch sinken sowohl der Hardware-Ressourcenbedarf als auch der Energieverbrauch. Eine gezielte Verringerung der Wortbreite steigert darüber hinaus die Effizienz von SIMD-Berechnungen, was insbesondere für Trajektorienplaner von Vorteil ist. Allerdings gehen geringere Wortbreiten typischerweise mit einem Verlust numerischer Genauigkeit einher. Für sicherheitskritische Funktionen muss die resultierende Abweichung daher in einem klar definierten Toleranzband liegen.

Im Rahmen des Projekts wurde bestimmt, in welchem Umfang sich die Wortbreite reduzieren lässt, ohne die Sicherheitsanforderungen an die Trajektorienplanung zu verletzen. Zusätzlich wurde der Genauigkeitsverlust infolge einer kleineren Anzahl von Bits im Bruchteil der Festkommadarstellung quantifiziert. Die Untersuchung zeigt, dass eine 32-Bit Festkommadarstellung die Positionsberechnung von Trajektorienpunkten über eine Strecke von 50 m mit einer maximalen Abweichung von 2 cm ermöglicht. Diese Kennzahlen dienen als Leitplanke für die Auslegung FPGA-basierter ALUs mit angepasster Wortbreite und bilden die Grundlage für eine effizientere und energieoptimierte Trajektorienplanung, die nahtlos in die Simulations-



und Update-Pipeline von UC3.2 integriert werden kann.

4.6.4 Frühe Validierung von Software-Updates durch digitale Zwillinge und domänenspezifisches Modellieren

Die zunehmende Komplexität moderner Fahrzeugsoftware und die Anforderungen an eine schnelle, sichere und flexible Integration neuer Funktionen erfordern innovative Entwicklungs- und Testmethoden. Eine zentrale Rolle spielt dabei die virtuelle Steuergeräteumgebung (vECU), die als digitale Repräsentation eines realen Steuergeräts dient. Sie ermöglicht die frühzeitige Validierung und Integration von Softwarekomponenten unabhängig von physischer Hardware und schafft damit die Grundlage für eine modellbasierte, skalierbare und risikoarme Entwicklung. Die vECU unterstützt die Simulation realer Einsatzszenarien und erlaubt die kontinuierliche Verbesserung von Funktionen durch datengetriebene Analyse und iterative Tests.

Zur Unterstützung einer effizienten und fehlerfreien Integration von Fahrzeugfunktionen in die vECU wurde im Rahmen des Use Case 3.1 eine domänenspezifische Sprache auf Basis von Xtext entwickelt. Die DSL erlaubt es Entwickler:innen, die beabsichtigte Funktionalität mit domänenspezifischen, hochabstrakten Konstrukten wie Signalinteraktionen und zeitlichem Verhalten zu beschreiben. Durch die Erhöhung des Abstraktionsniveaus vom Low-Level-Code zu spezialisierten Modellen minimiert der Einsatz der DSL das Risiko von Integrationsfehlern.

Die Verwendung der DSL ermöglicht eine dynamische Bewertung von Sicherheitsaspekten und Zeitverhalten vor dem Deployment. Unbeabsichtigte zeitliche Verzögerungen, die beim Einspielen von Updates oder neuen Funktionen auftreten können und die Systemleistung beeinträchtigen, lassen sich so frühzeitig erkennen.

Wird eine Funktion mithilfe der DSL spezifiziert, kann sie automatisch in eine vECU integriert und in einer Digital-Twin-Umgebung getestet werden. Dies ermöglicht eine frühe Validierung des Verhaltens unter realistischen und extremen Szenarien ohne Risiko für physische Assets. Darüber hinaus kann dieselbe Funktion im *Schattenmodus* parallel zum laufenden System ausgeführt werden, sodass ein direkter Vergleich der Ausgaben möglich ist, um Regressionen oder Timing-Anomalien zu erkennen, bevor die Funktion vollständig aktiviert wird.

Um die Ausführung der digitalen Zwillinge im Laufzeit-Framework zu ermöglichen, müssen sie gemäß einer Reihe wohldefinierter Regeln ausgedrückt werden, die in der domänenspezifischen Sprache (DSL) festgelegt sind. Die Entwicklung der DSL begann aus einer abstrakten Perspektive, die sich auf die wesentlichen Eigenschaften der Konstrukte konzentriert. Auf diese Weise vermeiden wir unnötige Implementierungszwänge und bewahren gleichzeitig die Flexibilität der Sprache sowie Maschinenunabhängigkeit. Der verwendete Abstraktionsmechanismus ähnelt dem von Datenabstraktionen und Klassen in objektorientierter Programmierung. Die DSL ermöglicht die Erstellung eines Programms nach einem Regelsatz, der sicherstellt, dass ein Subjekt, das in einer simulierten Umgebung ausgeführt wird, nicht erkennen kann, dass es evaluiert wird, während es gleichzeitig Artefakte für die Komponenten der heterogenen modularen Plattform liefert.

Zur Versorgung der vernetzten Komponenten des Laufzeit-Frameworks erzwingt die Sprache eine explizite Deklaration von Ereignissen, die während der Ausführung ausgelöst werden, einschließlich deren Typen, um frühzeitig Abweichungen erkennen zu können. So ergibt die

Ausführung des digitalen Zwillings eine präzise formulierte Ereignisfolge, mit der die verknüpfte prädiktive Simulation vorhersagbare Artefakte liefern kann.

Im Gegensatz zur Kompilierung typischer objektorientierter Programmiersprachen, bei der Fehler bereits vor der Ausführung erkannt werden können, ermöglicht die DSL für digitale Zwillinge eine Fehlererkennung auf Basis der Ausführung, ergänzt durch oder parallel zur Konformitätsüberwachung und tatsächlichen Ausführung auf einer ECU.

Zustandsänderungen einer Softwarekomponente können durch die Ereignisse an ihrer Kommunikationsschnittstelle überwacht werden. Um die Überwachung interner Zustandsänderungen zu ermöglichen, müssen digitale Zwillinge diese Ereignisse bereitstellen. Unsere vorgesehene domänenspezifische Sprache erlaubt daher die Spezifikation zweier Arten von Überwachungsereignissen: *normale Ereignisse* und *Entscheidungsereignisse*. Während normale Ereignisse den Empfang von Eingabewerten, das Senden von Ausgaben oder die Interaktion mit anderen Komponenten beschreiben, sind Entscheidungsereignisse solche, die architektonische Grenzen überschreiten – etwa das Senden eines Befehls zur Geschwindigkeitssteigerung an eine andere Komponente.

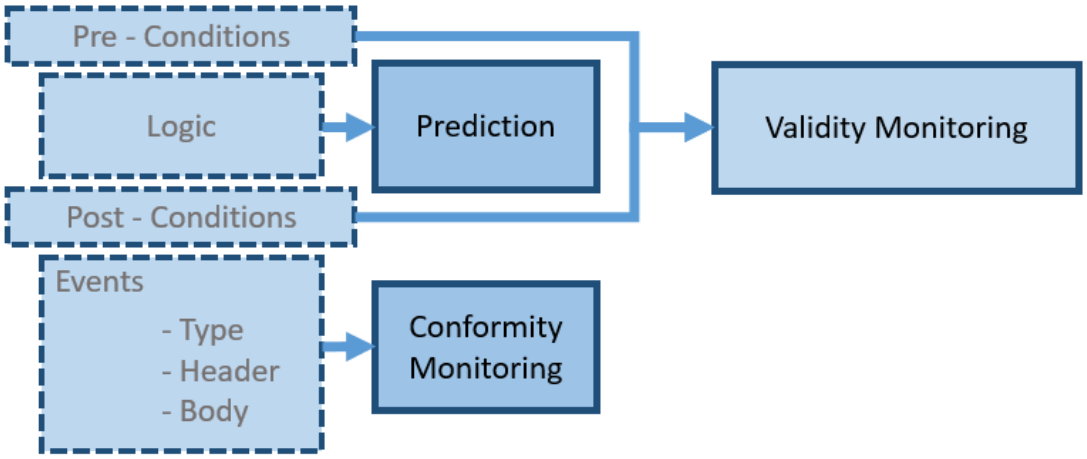
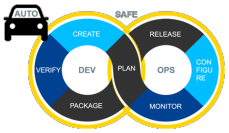


Abbildung 4.15: Hauptkonstrukte der domänenspezifischen Sprache

Die Konstrukte der Sprache ermöglichen die Funktionsspezifikation in Form von Artefakten, die von den Komponenten des Laufzeit-Frameworks weiterverarbeitet werden. Wie in Abbildung 4.15 dargestellt, liefert ein mit unserer DSL formulierter digitaler Zwilling Eingaben für zentrale Recheneinheiten des Laufzeit-Frameworks in Form von:

- Vorbedingungen und Nachbedingungen, die der Gültigkeitsüberwachung des Orakels dienen.
- Der Logik der Funktion, die Eingang für die Vorhersagekomponente des Orakels ist.
- Einer Liste von Ereignissen, die die Konformitätsüberwachung speisen.

Zur Unterstützung der frühzeitigen Validierung von Software-Updates in sicherheitskritischen Systemen wurde eine auf QEMU basierende Emulationsumgebung entwickelt, die

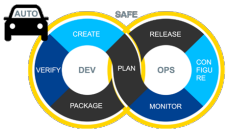


ADAS-Anwendungssoftware unter reproduzierbaren realistischen Szenarien ausführt. Der auf QEMU basierende emulierte Controller agiert als virtualisierte Repräsentation einer eingebetteten ECU und führt Anwendungs-Binaries aus, die aus modellbasierten Entwicklungsworkflows (z.B. Simulink) stammen. Diese Emulationsplattform ist mit einer Simulationsumgebung (CARLA) verbunden, die dynamische, szenariobasierte Eingangsdaten bereitstellt (z.B. Ego-Geschwindigkeit, Verhalten des vorausfahrenden Fahrzeugs, Relativabstand), die für ACC-, CACC- und Notbremsanwendungsfälle repräsentativ sind. Die Architektur integriert ein Digital-Twin-Setup, mit dem das Echtzeitverhalten der Software unter variierenden Umgebungs- und Verkehrssituationen überwacht werden kann. Laufzeit-Traces, einschließlich interner Zustandsübergänge und Ausgabesignale, werden aufgezeichnet und mit erwarteten Verhaltenssignaturen verglichen. Diese dienen als Referenz zur Erkennung von Anomalien oder Regressionen, die durch Softwarefehler oder Updates eingeführt wurden. Das Framework unterstützt Closed-Loop-Ausführung, wobei die Steuerungsausgaben des emulierten Controllers in die Simulation zurückgeführt werden, was eine Echtzeitüberwachung des Verhaltens und eine zeitabhängige Validierung unter dynamischen Bedingungen ermöglicht.

Zur Sicherstellung von Transparenz und Rückverfolgbarkeit im Validierungsprozess werden wichtige Ausführungsereignisse mit Zeitstempeln versehen und mit dem Szenarienverlauf in CARLA synchronisiert. Dies erlaubt sowohl funktionale Verifikation als auch die zeitliche Korrelation zwischen Stimuli und Systemreaktion – entscheidend zur Validierung von Steuerungsentscheidungen wie Bremsen oder Spurwechsel. Das Setup dient auch der Ausführung im Schattenmodus, in dem neue bzw. aktualisierte Softwareversionen parallel zu Basisversionen auf dem digitalen Zwilling ausgeführt werden können. So lassen sich Entscheidungen und Reaktionszeiten beider Versionen unter identischen Bedingungen vergleichen, was eine hochgenaue Detektion von Verhaltensabweichungen oder Nebeneffekten erlaubt. Abweichungen bei Aktuatorsignalen oder internen Zustandsübergängen können markiert und auf konkrete Softwareänderungen zurückgeführt werden. Das Gesamtdesign ist auf modulare Integration ausgerichtet – Simulation, Emulation und Überwachungskomponenten kommunizieren über leichtgewichtige Schnittstellen, wodurch das Framework plattformübergreifend portierbar ist. Die QEMU-Umgebung abstrahiert hardwarespezifische Details, sodass dieselbe Umgebung auf verschiedenen Zielplattformen eingesetzt werden kann, ohne die Zeitrealität zu beeinträchtigen. Der Ansatz ermöglicht so eine frühe, plattformunabhängige Validierung mit kontinuierlicher Verhaltenssicherung, wie sie für sichere, inkrementelle Updates im automobilen DevOps-Lifecycle erforderlich ist. Zudem gewährleistet die Nutzung quelloffener Simulations- und Emulationstechnologien Reproduzierbarkeit und Skalierbarkeit für den industriellen Einsatz.

4.7 Iterative und kontinuierliche Absicherung sicherheitskritischer Funktionen durch Synthese von Felddaten

Für die Zulassung und den Betrieb automatisierter Fahrsysteme (ADS) ist der Nachweis der Sicherheit von inhärenter Bedeutung. Dies geschieht durch den Aufbau einer strukturierten Argumentationkette von Sicherheitsbehauptungen, die durch Evidenz belegt werden. Die



Verwendung von sogenannten Safety Assurance Cases (SACs) ist zu einer gängigen Praxis geworden, um den unterschiedlichen Stakeholdern die Sicherheit des Systems zu demonstrieren, [6]. SACs sind ein "*structured argument, supported by evidence, aimed at justifying that a system and an activity are acceptably safe for a specifc application in a particular operating environment and comply with recognized good practices*" [83].

SACs und deren strukturierte Argumentationsketten sind nicht statisch anzusehen, sondern sollen vielmehr als „lebende Dokumente" verstanden werden, die über den gesamten Produktlebenszyklus hinweg den Erfordernissen entrsprechend angepasst angepasst und fortgeschrieben werden müssen. In traditionellen Systemen werden diese jedoch oftmals als statisch behandelt und demonstrieren die Sicherheit eines bestimmten Systems zum Zeitpunkt der Zulassung auf Basis von Nachweisen, die während der Entwicklungs- und Validationsphase des Systems erhoben wurden.

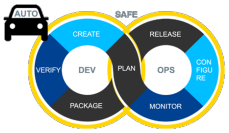
Da automatisierte Fahrfunktionen naturgemäß in einem offenen Welt-Kontext (open world context) agieren, sind sie ständigen Veränderungen der Umgebungsgegebenheiten und Bedingungen ausgesetzt, die eine Falsifikation des Sicherheitsnachweises zur Folge haben können, aufgrund unerwarteter Systemeigenschaften, Umweltunsicherheiten oder unzureichender Systemleistung. Selbst ein wohl definierter Sicherheitsnachweis unterliegt Unsicherheiten, da die von Experten erstellte Sicherheitsargumentation naturgemäß eine subjektive Komponente aufweist. Der sogenannte Bestätigungsfehler (Confirmation Bias) stellt dabei eine besondere Gefahr für die Validität und Umfänglichkeit des Sicherheitsnachweises dar [43].

Methoden der Feldbeobachtung spielen eine entscheidende Rolle bei der Überprüfung der Validität des Safety Assurance Cases nach der Inbetriebnahme; dies gilt in besonderem Maße für den Betrieb von ADS. Die zunehmende Verfügbarkeit von Echtzeit-Fahrzeugdaten eröffnet die Möglichkeit, potenzielles sicherheitskritisches Fehlverhalten, welches die Gültigkeit von SACs beeinträchtigt, frühzeitig festzustellen. Diese Daten können genutzt werden, um inkrementell sichere und leistungsfähigere Systeme zu entwickeln. Trotz dieses Potenzials fehlt es bisher jedoch an einem strukturierten Ansatz, um Felddaten inkrementell und kumulativ im Sicherheitsnachweis zu verwerten [62].

Mehrere Normen fordern bereits explizit die Feldbeobachtung für unterschiedliche Aspekte des Sicherheitsnachweises. ISO 26262 [47] schreibt einen Überwachungsprozess vor, um sicherheitsrelevante Vorfälle während des Fahrzeuglebenszyklus „Betrieb, Service und Außerbetriebnahme" zu verfolgen. Durch Sammlung von Felddaten sollen funktionale Sicherheitsprobleme erkannt und Maßnahmen wie Entscheidungsfindung, Festlegung von Rückruf- und Korrekturmaßnahmen sowie Berichterstattung an Beteiligte ausgelöst werden.

ISO 21448 [49] liefert Leitlinien zur Gewährleistung der Safety of the Intended Functionality (SOTIF), also zur Vermeidung unangemessener Risiken durch funktionale Unzulänglichkeiten. Sie verlangt die Definition eines Überwachungsprozess vor Freigabe, um Risiken im Betrieb zu bewerten. Werden neue Gefährdungen oder funktionale Unzulänglichkeiten entdeckt, erfolgt eine Neubewertung des Risikos und geeignete Maßnahmen werden implementiert. Sofortige oder langfristige Maßnahmen können erforderlich sein, um das System zu aktualisieren und erneut freizugeben.

Die Norm UL4600 [84] hat zum Ziel, eine umfassende Anleitung zur Entwicklung und Absicherung hochautomatisierter Fahrfunktionen zu bieten. Sie behandelt explizit die Sicherheit autonomer Systeme, d.h. Systeme die ohne direktes menschliches Eingreifen operieren. Zu-



dem wird die Felddatenüberwachung durch Einbindung von sogenannten Safety Performance Indicators (SPIs) im Sicherheitsnachweis gefordert.

SSystem-Theoretic Process Analysis"(STPA) [66] ist eine moderne, innovative Methode zur Gefährdungs- und Risikoanalyse für komplexe vernetzte und softwareitensive Systeme. Sie verfolgt einen systemtheoretischen Ansatz, um mögliche Gefährdungen und ihre Ursachen zu identifizieren und Transparenz über die komplexen Beziehungen zwischen Systemkomponenten und Gefährdungen zu schaffen. Die Feldüberwachung ist integraler Bestandteil des STPA-Prozessmodells und findet besondere Berücksichtigung bei den Kontrollaktionen sowie der Beobachtung von Frühindikatoren als Teil des operativen Sicherheitsmanagements (operational safety management). Durch die Forderung und Etablierung einer aktiven Feldbeobachtung zielt STPA darauf ab, potenzielle Gefährdungen und Abweichungen vom Sollverhalten proaktiv zu erkennen und zu vermeiden.

Safety Assurance Cases sind bisher vor allem für den menschlichen Nutzer gedacht, sei es zur Überarbeitung, Überwachung oder zur Sicherheitsbeurteilung. Aufgrund der wachsenden Komplexität der Systeme in sich ändernden Umgebungen und der Vielzahl an Annahmen, die überwacht werden müssen, sind sie jedoch oftmals schwer handhabbar. Ein datenbasierter Ansatz in Verbindung mit maschinenlesbaren Sicherheitsargumentationen und der Integration von SACs mit datengetriebenen Methoden kann hier durch Automation abhilfe schaffen. Der SafetyDevOps-Prozess 3 kann verwendet werden, um iterative Update-Zyklen in ADS zu unterstützen. Diese Updates lassen sich durch Feldbeobachtung effektiv stützen und ermöglichen zudem die Überwachung bestimmter Sicherheitseigenschaften zu Laufzeit, ein zentrales Element der Absicherung im Feld (operational safety assurance).

4.7.1 Systematischer Ansatz zur Identifikation nützlicher SPIs

Der systematische Ansatz besteht aus einer vierstufigen Methodik zur Gewährleistung der operativen Sicherheitsabsicherung. Der erste Schritt beinhaltet das Einholen von Einschätzungen von Sicherheitsexperten und deren Darstellung in Form von Entscheidungs- und Vertrauenswerten, jeweils auf einer Skala von 0 bis 1. Die Entscheidungsskala spiegelt die Neigung eines Experten zu einer Aussage wider, wobei 1 vollständige Zustimmung, 0 völlige Ablehnung und 0,5 Unentschlossenheit bedeutet. Die Vertrauensskala misst den Informationsgrad des Experten zur Stützung seiner Entscheidung, wobei 1 ausreichende Information und 0 unzureichende Information bedeutet [46]. Die gesammelten Daten werden in einer Evaluationsmatrix dargestellt, wie in Abbildung 4.16 gezeigt.

Die Entscheidungs- und Vertrauenswerten werden anschließend in Glauben (Bel), Zweifel (Disb) und Unsicherheit (Unc) umgewandelt, gemäß den folgenden Formeln:

$$\begin{aligned} \mathrm{Bel}(x) &= \frac{\mathrm{Conf}(x) - 1}{2} + \mathrm{Dec}(x) \\ \mathrm{Disb}(x) &= \frac{\mathrm{Conf}(x) + 1}{2} - \mathrm{Dec}(x) \\ \mathrm{Unc}(x) &= 1 - \mathrm{Bel}(x) - \mathrm{Disb}(x) \end{aligned} \tag{4.1}$$

Bei der Bestimmung der Werte muss die sogenannte „Josang-Bedingung“ in der Evaluationsmatrix berücksichtigt werden. Werte außerhalb des Dreiecks in der Matrix weisen auf

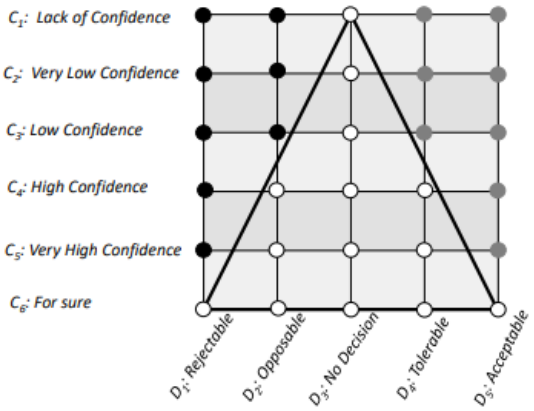


Abbildung 4.16: Evaluationsmatrix [46].

negative Glaubens- (schwarze Punkte) oder Zweifelwerte (graue Punkte) hin, die ungültig sein können. Dies erfordert zusätzliche Formeln zur Anpassung der Werte, um die „Josang-Bedingung“ zu erfüllen.

Wenn $\mathrm{Dec}(x) > \frac{1+\mathrm{Conf}(x)}{2}$, müssen die Entscheidungswerte wie folgt gesetzt werden:

$$\mathrm{Dec}(x) = \frac{1 + \mathrm{Conf}(x)}{2} \tag{4.2}$$

Wenn $\mathrm{Dec}(x) < \frac{1-\mathrm{Conf}(x)}{2}$, müssen die Entscheidungswerte wie folgt gesetzt werden:

$$\mathrm{Dec}(x) = \frac{1 - \mathrm{Conf}(x)}{2} \tag{4.3}$$

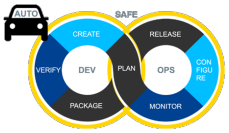
Der zweite Schritt besteht darin, die Bel-, Disb- und Unc-Werte entlang der SAC-Struktur zum obersten Ziel zu propagieren. Dabei ist der Typ der Argumentbeziehung entscheidend. Es werden drei Typen unterschieden: Konjunktives Argument (C-arg), Disjunktives Argument (D-arg) und Hybrides Argument (H-arg). In einem C-arg wird die Schlussfolgerung C nur dann unterstützt, wenn alle Prämissen (z. B. P1 und P2) zutreffen – ähnlich einem logischen UND-Gatter. Ein D-arg hingegen erlaubt, dass bereits eine Prämisse ausreicht – ähnlich einem logischen ODER-Gatter. Die Werte werden mittels Dempsters Kombinationsregel propagiert. Die Berechnungen unterscheiden sich je nach Argumenttyp:

Für ein D-arg gilt:

$$bel = 1 - \times_{i=1}^{n} (1 - bel_i) \tag{4.4}$$

$$disb = \times_{i=1}^{n} (disb_i) \tag{4.5}$$

Für ein C-arg gilt:



$$bel = \times_{i=1}^{n} (bel_i) \tag{4.6}$$

$$disb = 1 - \times_{i=1}^{n} (1 - disb_i) \tag{4.7}$$

Der dritte Schritt konzentriert sich auf die Bewertung einzelner Claims basierend auf der zuvor identifizierten Unsicherheit. Ein Fachexperte definiert einen spezifischen Unsicherheitsschwellenwert, der von Faktoren wie Anwendungsdomäne, SIL-Stufen, Kritikalität der Funktion oder Risikobewertungen abhängt. Claims, deren Unsicherheit über diesem Schwellenwert liegt, müssen mittels eliminativer Argumentation (EA) [39] behandelt werden. Dabei wird ein Claim durch Zweifel oder Einwände (Defeater) hinterfragt. Das Argumentationsmuster bleibt gleich, unabhängig davon, ob der Defeater widerlegend, untergrabend oder unterminierend ist. Neue Claims oder Nachweise können eingebracht werden, um Defeater zu entkräften. Inferenzregeln verknüpfen die Claims mit ihren Nachweisen, um Vertrauen aufzubauen. Kann ein Nachweis oder eine Inferenzregel nicht weiterentwickelt werden, bleibt ein „Residualzweifel“. Claims, die hinterfragt werden, müssen durch neue Nachweise gestützt werden. Manche Claims – ob ursprünglich oder durch EA ergänzt – bleiben möglicherweise unbeantwortet. Für diese werden spezifische Leistungsindikatoren (SPIs) definiert.

Der letzte Schritt besteht in der Definition von SPIs. Diese müssen realistisch, relevant und eindeutig auf sicherheitsrelevante Ziele ausgerichtet sein, unabhängig von ihrer Komplexität. Häufig ist eine Kombination mehrerer SPIs notwendig, um die Sicherheitsleistung korrekt zu beschreiben. Ein SPI ist ein Metrik-Schwellenwert-Paar, das einen Claim über den Vergleich mit einem Schwellenwert validiert. Ein SPI kann sich auf Produktleistung, Designqualität, Prozessgüte oder Konformität mit Verfahren beziehen. Der Kontext ist entscheidend – eine Metrik allein ist kein SPI [55].

Der entwickelte systematische Ansatz wurde exemplarisch anhand des Use Cases „Construction Zone Assist“ erläutert und in [61] detailliert beschrieben.

4.7.2 Ursachenanalyse nach Felddatensynthese

Innerhalb des Safety Case definieren Safety Performance Indicators (SPIs) die Laufzeitbedingungen, unter denen eine Aussage gültig bleibt. Wird eine SPI-Bedingung verletzt, wird der zugehörige Claim ungültig und muss im nächsten DevOps-Zyklus aktualisiert werden. Um herauszufinden, welche Systemartefakte die Verletzung verursacht haben, ist eine Ursachenanalyse erforderlich, ausgehend vom ungültigen Claim. Während dieser Analyse werden die Verknüpfungen zwischen den relevanten Entwicklungsartefakten traversiert, bis Artefakte gefunden wurden, die mögliche Ursachen für die Verletzung der Laufzeitbedingung darstellen. In komplexen Systemen wie ADS können diese relevanten Verknüpfungen zahlreich und stark verwoben sein, was eine manuelle Analyse zeitaufwendig und fehleranfällig macht.

Figure 4.17 zeigt eine beispielhafte Traversierung von verschiedenen Artefakte.

Zuerst wird der Safety Case vom Claim ausgehend bis zu den Nachweisen durchlaufen — typischerweise Entwicklungsartefakte (z. B. Ergebnisse aus der Fehleranalyse) —, die diesen stützen. Ein Beispiel: Ein Claim wie „die Fehlermodelle sind frei von Einzelfehlerursachen“ kann sich auf Ergebnisse einer Minimal-Cutset-Analyse stützen. Als Nächstes müssen alle

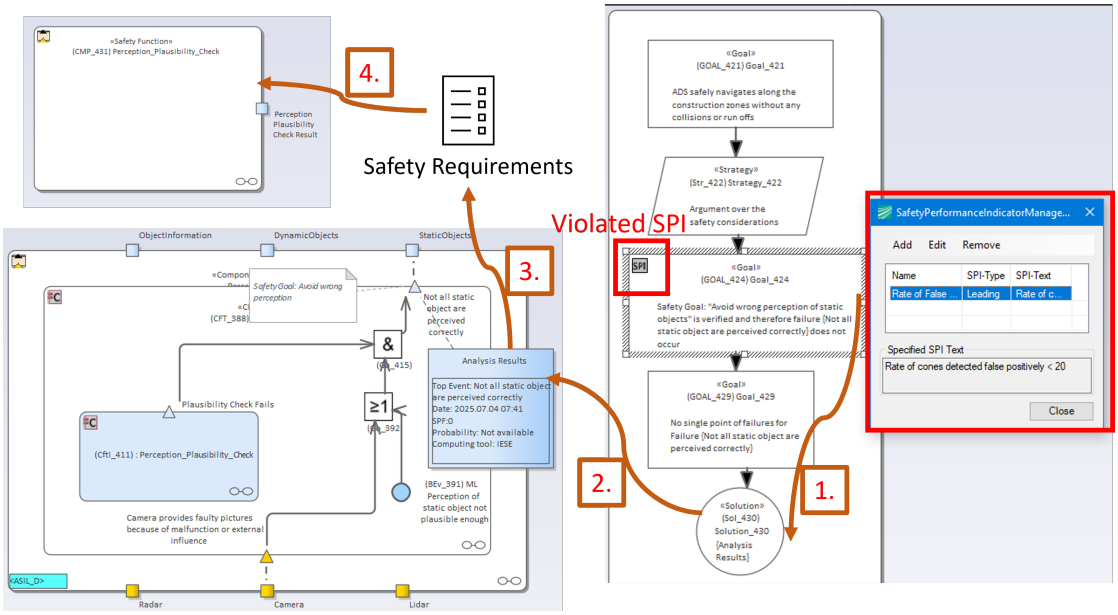


Abbildung 4.17: Beispiel Beispiel für Traversierung von Artefakten

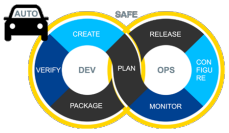
Entwicklungsartefakte identifiziert werden, die (direkt oder indirekt) mit diesen Nachweisen verknüpft sind. In dem abgebildetem Beispiel würden hierbei zuerst über die im Fehleranalyseergebnis aufgelisteten Fehler die entsprechend spezifizierten Sicherheitsanforderungen erfasst werden. Anschließend können die Sicherheitsfunktionenn aufgelistet werden, welche diese Sicherheitsanforderungen erfüllen und ggf. die Ursachen für die Verletzung der Laufzeitbedingung darstellen.

Zur Vereinfachung wurde die Anwendung Trace Viewer entwickelt, die die Rückverfolgung von Entwicklungsartefakten automatisiert. Sie basiert auf dem DDI-Framework und erfordert:

- Ein DDI-Modell mit den integrierten Artefakten,
- die ID des initialen interessierenden Elements (z. B. ein SPI oder ein MinimalCutsets-Ergebnis),
- eine JSON-formatierte Trace-Konfiguration mit den Referenznamen und Elementeigenschaften.

In der Konfiguration definieren Experten sogenannte „Trace-Pfade“ für jeden Elementtyp. Wenn der Elementtyp zum Beispiel „MinimalCutsets“ ist, könnte der Trace Viewer folgende Verbindungen verfolgen:

- Minimal Cutsets → zugewiesener Failure
- Failure → verknüpfte Safety Requirement
- Safety Requirement → Safety Function



Bei der Ausführung traversiert der Trace Viewer zuerst den GSN-Sicherheitsnachweisbaum vom SPI-Element aus über Strategien, Ziele und Lösungen bis zum Nachweis-Artefakt. Danach folgt er den definierten Trace-Pfaden im DDI-Modell und gibt alle besuchten Elemente aus. Existieren mehrere Lösungselemente, erstellt das Tool separate Berichte für jedes.

Im Beispiel-DDI-Modell (Figure 4.18) ist der SPI bezüglich der Falsch-positv-Rate bei der Leitkegelerkennung mit dem Ziel 429 verknüpft, das über das untergeordnete Ziel 429 und zu Lösung 430 führt, welche ein Ergebnis der Fehlermodellanalyse referenziert. Die Ausführung des Trace Viewers mit dieser Konfiguration ergibt eine Konsolenausgabe (Figure 4.19), die Folgendes zeigt:

- Den Pfad durch den GSN-Baum von SPI 1 zu Lösung 35,
- Die Abfolge der besuchten Entwicklungsartefakte gemäß der JSON-Konfiguration.

Durch die Aggregation dieser Pfade können Ingenieur*innen direkt erkennen, welche Artefakte (z. B. Fehler, Sicherheitsanforderungen oder Sicherheitsfunktionen) potenzielle Ursachen für die Verletzung eines SPIs darstellen. Dieser automatisierte Ansatz reduziert Zeitaufwand und Fehlerquellen der manuellen Ursachenanalyse und ermöglicht effizientere Updates zur Wiederherstellung der Gültigkeit von Claims. D4.2[28] bietet weitere Informationen und einen detaillierteren technischen Einblick der DDI Trace Viewer Komponente.

4.8 Innovative Lifecycle Modelle für den sicheren Einsatz Neuronaler Netze in CPS durch neuartige Modellstrukturen zu OOD Erkennung

Während konzeptbasierte Modelle zur Sicherheitsargumentation beitragen, bleibt die Erkennung von **Out-of-Distribution (OOD)**-Eingaben eine zentrale Herausforderung. Klassische DNNs neigen dazu, auf unbekannten Eingaben übermäßig selbstsichere Vorhersagen zu machen – besonders kritisch im Betrieb, wenn Fehler auf nachgelagerte Komponenten übergehen. Deshalb erfordert der Einsatz tiefer neuronaler Netze (DNNs) in sicherheitskritischen Cyber-Physical Systems (CPS) Mechanismen zur Laufzeitüberwachung, da Fehlvorhersagen schwerwiegende Folgen haben können. Klassische Metriken wie Accuracy sind in diesem Umfeld nicht ausreichend, da keine Ground-Truth-Labels während des Betriebs vorliegen. Besonders wichtig ist daher die Erkennung von Out-of-Distribution (OOD) und Open-Set-Samples (OSR), um unsichere Zustände zuverlässig zu detektieren.

AURORA (**A**uto-associative **U**niversal **R**eal-time **O**utlier **R**isk **A**ssessment) [95] begegnet dieser Herausforderung durch eine Kombination aus generativem Deep Learning und Unschärfemodellierung.

Bestehende Ansätze nutzen meist separate Modelle (z. B. Autoencoder), was zu zusätzlicher Latenz und erhöhtem Aufwand führt. Zudem sind viele Methoden auf spezifische Aufgaben zugeschnitten oder benötigen aufwendige Trainingsstrategien. Das vorgestellte AURORA-Netzwerk (Auto-associative Universal Real-time Outlier Risk Assessment) bietet eine skalierbare, anwendungsunabhängige Architektur, die OOD-Erkennung direkt in das Modell integriert. Kernidee ist die Nutzung eines Autoencoders (AE) bzw. Variational Autoencoders

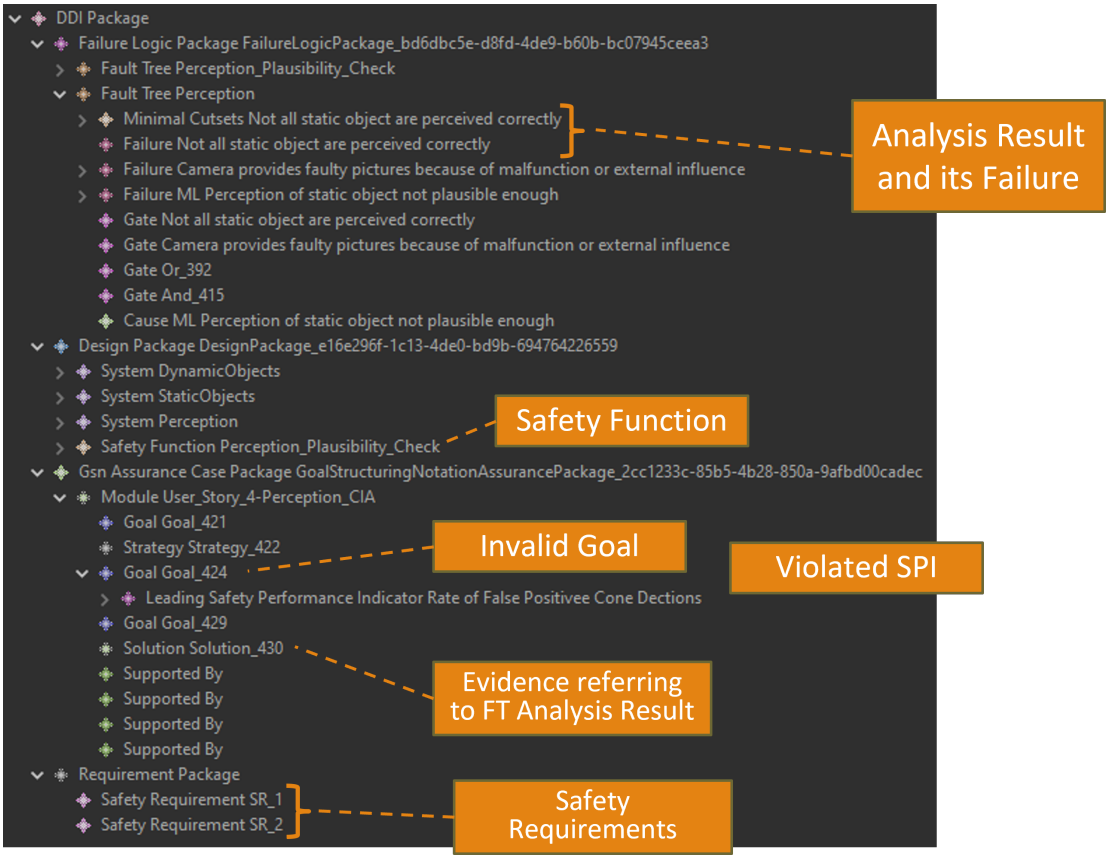


Abbildung 4.18: Beispiel für Trace Viewer-Eingabe-DDI

```
GSN Trace from root GSN element:
{Type=LeadingSafetyPerformanceIndicator; Name=Rate of False Positivee Cone Dections; Id=2f809
{Type=Goal; Name=Goal_424; Id=dd2895dd-de0d-4a5c-88cc-352189d63f77} ->
{Type=Goal; Name=Goal_429; Id=a892a79d-2192-432b-9367-6369eb60a322} ->
{Type=Solution; Name=Solution_430; Id=adb62723-304b-493f-91f1-6b559ec2b961}

Artifact Traces with starting Element of Interest Type: MinimalCutsets
- (MinimalCutsets) Name:Not all static object are perceived correctly | ID:64979ab2-32b1-471d
  - (Failure) Name:Not all static object are perceived correctly | ID:1c76af22-658f-470c-a1a6
    - (SafetyRequirement) Name:SR_1 | ID:b5a3f64a-78ea-4d89-9e57-9f885f4330ad
      - (SafetyFunction) Name:Perception_Plausibility_Check | ID:34ab06f7-a337-41c6-b63a-9778
    - (SafetyRequirement) Name:SR_2 | ID:852a8bfb-ee82-413f-9e48-d848529c2840
      - (SafetyFunction) Name:Perception_Plausibility_Check | ID:34ab06f7-a337-41c6-b63a-9778
```

Abbildung 4.19: Beispiel für Trace Viewer-Ausgabe

(VAE) als Feature-Extraktor und gleichzeitiger OOD-Detektor. Die probabilistische Natur des VAE sorgt für robuste Generalisierung und erlaubt die Erfassung epistemischer und aleatorischer Unsicherheiten. So kann ein separater Head für Aufgaben direkt im latenten Raum

trainiert und unabhängig ausgetauscht werden Dabei wird die Latenz gering gehalten, da keine zusätzlichen Modelle erforderlich sind. AURORA kombiniert drei Bausteine:

- **Latent-Space-Pretraining**: durch (variationale) Autoencoder, die eine strukturierte Repräsentation erzeugen.
- **Uncertainty Estimation**: über Monte-Carlo Dropout im Klassifikationskopf.
- **Reconstruction-based OOD Detection** wobei Decoder-Fehler zur Laufzeit ausgewertet werden.

Das fertige Modell weißt dann unabhängig von dessen genauer Umsetzung, auf jeden Fall folgende Struktur auf 4.20:

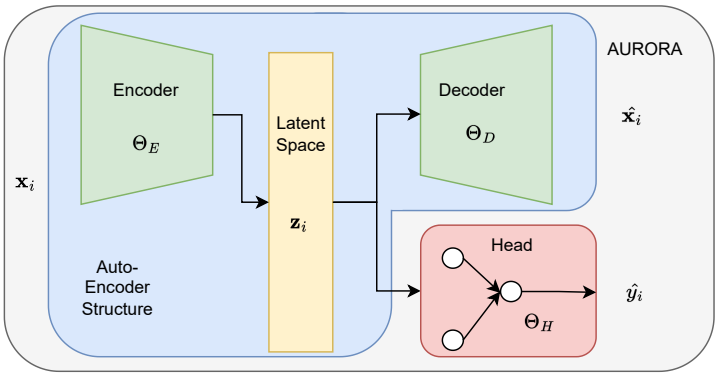


Abbildung 4.20: AURORA Inference Stage

Experimente auf MNIST und dem German Traffic Sign Recognition Benchmark (GTSRB) zeigen, dass AURORA mit State-of-the-Art-Methoden mithalten kann. Auf MNIST erreicht das System eine AUROC von 91% bei OOD-Detektion, auf GTSRB von 91 %. Besonders der VAE verbessert die Robustheit gegenüber Rauschen (z.B. Shot Noise: +30 % TPR gegenüber AE) und stabilisiert die Klassifikation (96 % Genauigkeit). Die Kombination von Unsicherheitsabschätzung und Rekonstruktionsfehlern reduziert Fehlklassifikationen und steigert die Erkennungsrate unbekannter Klassen.

Die Ergebnisse verdeutlichen, dass die vorgeschlagene Architektur eine effiziente und robuste Lösung für Laufzeitüberwachung in CPS darstellt. Sie ermöglicht gleichzeitige Klassifikation und OOD-Detektion mit geringem Overhead und hoher Generalisierbarkeit [95].

Fazit und Ausblick: AURORA eröffnet neue Perspektiven für die sichere Integration von DNNs in CPS. Die Fähigkeit des kontinuierlichen getrennten Retrainings im Head und Unsicherheitsbewertung unterstützt das ADSO-Paradigma iterativer Updates. Zukünftige Arbeiten sollen die Laufzeiteffizienz weiter optimieren, Vergleiche mit klassischen OOD-Methoden vertiefen und den Einsatz auf ressourcenbeschränkter CPS-Hardware untersuchen. Zudem ist geplant, OOD-Samples zur Ursachenanalyse von Fehlvorhersagen einzusetzen.

Der Einsatz von Machine-Learning-basierter Software in Cyber-Physical Systems (CPS) stellt jedoch nicht nur die eingesetzten Modelle selbst vor erhebliche Herausforderungen. Um

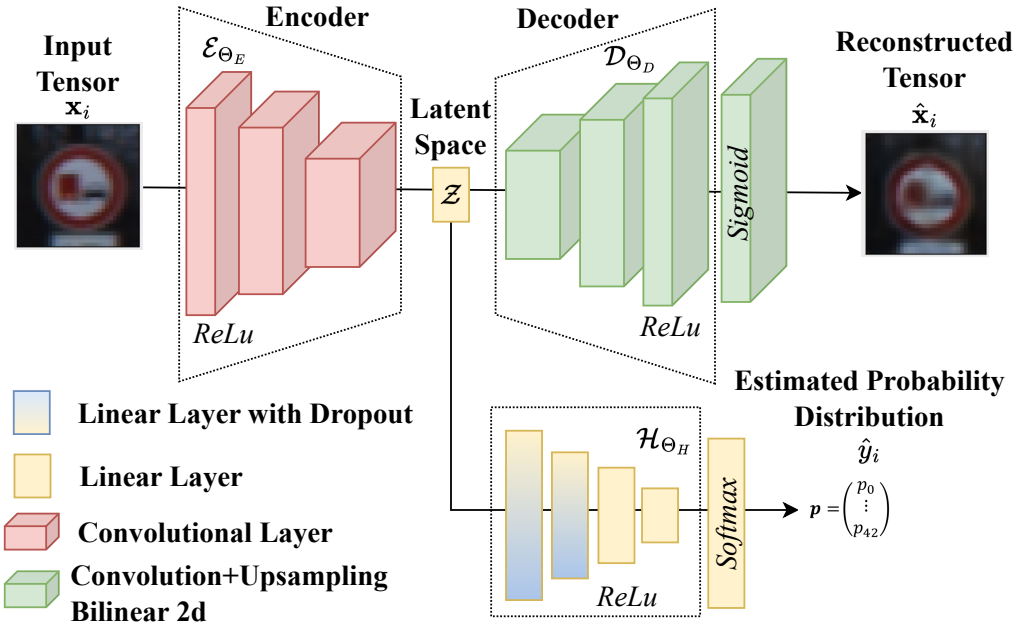
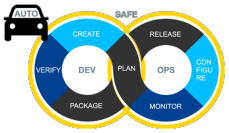


Abbildung 4.21: AURORA Gesamtsystem-Architektur

ein sicheres Verhalten in realen Anwendungen gewährleisten zu können, sind neuartige Teststrategien erforderlich. Da jedoch nicht alle Edge- und Corner-Cases im Vorfeld bekannt sein können und ein durchgängiges Komponentenverständnis fehlt, wenn Funktionalität auf Basis zuvor bereitgestellter Daten erlernt wird, entsteht, wie bereits eingeführt, der Bedarf nach einer Online-Überwachungskomponente, welche in einen komplett neugedachten Lebenszyklus eingebettet werden muss. Damit müssen auch neuartige Entwicklungs- und Wartungsprozesse berücksichtigt werden.

4.8.1 Lebenszyklusmodell

Das ARTEMIS-Modell (Adaptive Response and Tracking of External Model Inputs in Safety-Critical Systems) [96] adressiert eine zentrale Herausforderung beim Einsatz von neuronalen Netzen in sicherheitskritischen CPS: den systematischen Umgang mit Out-of-Distribution (OOD) Daten über den gesamten Systemlebenszyklus hinweg. Während AURORA die technische Lösung für die Laufzeitüberwachung bereitstellt, etabliert ARTEMIS den strukturierten Prozessrahmen für kontinuierliche Adaptation und Retraining. Traditionelle Entwicklungsprozesse wie das V-Modell erweisen sich als ungeeignet für ML-basierte CPS, da sie nicht für sich entwickelnde Umgebungsbedingungen und neu auftretende Datenverteilungen konzipiert wurden. Diese Limitierung wird besonders in sicherheitskritischen Anwendungen wie dem automatisierten Fahren problematisch, wo eine Lücke zwischen der Notwendigkeit kontinuierlichen Lernens und den strikten Sicherheitsanforderungen entsteht. Der ARTEMIS-Prozess gliedert sich in einen geschlossenen Regelkreis aus vier Hauptphasen. In der initialen Entwicklungsphase wird zunächst die Operational Design Domain (ODD) formal spezifiziert, die die zulässigen



Umgebungsbedingungen und Betriebsparameter definiert. Basierend auf dieser Spezifikation erfolgt die Sammlung des initialen Datensatzes sowie die Hyperparameter-Optimierung zur Minimierung der Verlustfunktion. Vor dem Deployment wird eine Sicherheitsmonitoring-Komponente wie AURORA integriert. Während der Betriebsphase führt das System kontinuierliche OOD-Detektion durch, wobei Eingangsdaten in Echtzeit mittels einer Detektionsfunktion bewertet werden. Bei erkannten OOD-Samples werden die entsprechenden Vorhersagen systematisch verworfen und die verworfenen Daten zusammen mit Kontextinformationen wie GNSS-Position, Sensorzustand und Zeitstempel in einer Cloud-Datenbank gespeichert. Diese Datensammlung bildet die Grundlage für die nachfolgende adaptive Lernphase. In der adaptiven Lernphase erfolgt ein gezieltes Sample-Retrieval kritischer OOD-Daten aus der Cloud-Datenbank. Diese Samples durchlaufen einen manuellen Annotationsprozess durch Fachexperten, um die erforderliche Datenqualität für sicherheitskritische Anwendungen zu gewährleisten. Der erweiterte Trainingsdatensatz wird anschließend für das Retraining verwendet, wobei vollständige Versionskontrolle und Nachvollziehbarkeit aller Änderungen sichergestellt wird. Die finale Phase umfasst das Retraining und Redeployment, das entweder periodisch oder bedarfsgesteuert bei erkannter Performancedegradation ausgelöst wird. Nach umfassender Sicherheitsvalidierung erfolgt eine kontrollierte, stufenweise Ausrollung, wobei historische Modellversionen für mögliche Rollback-Szenarien erhalten bleiben. Der ganze Prozess lässt sich auch in einem UML Diagram darstellen 4.22:

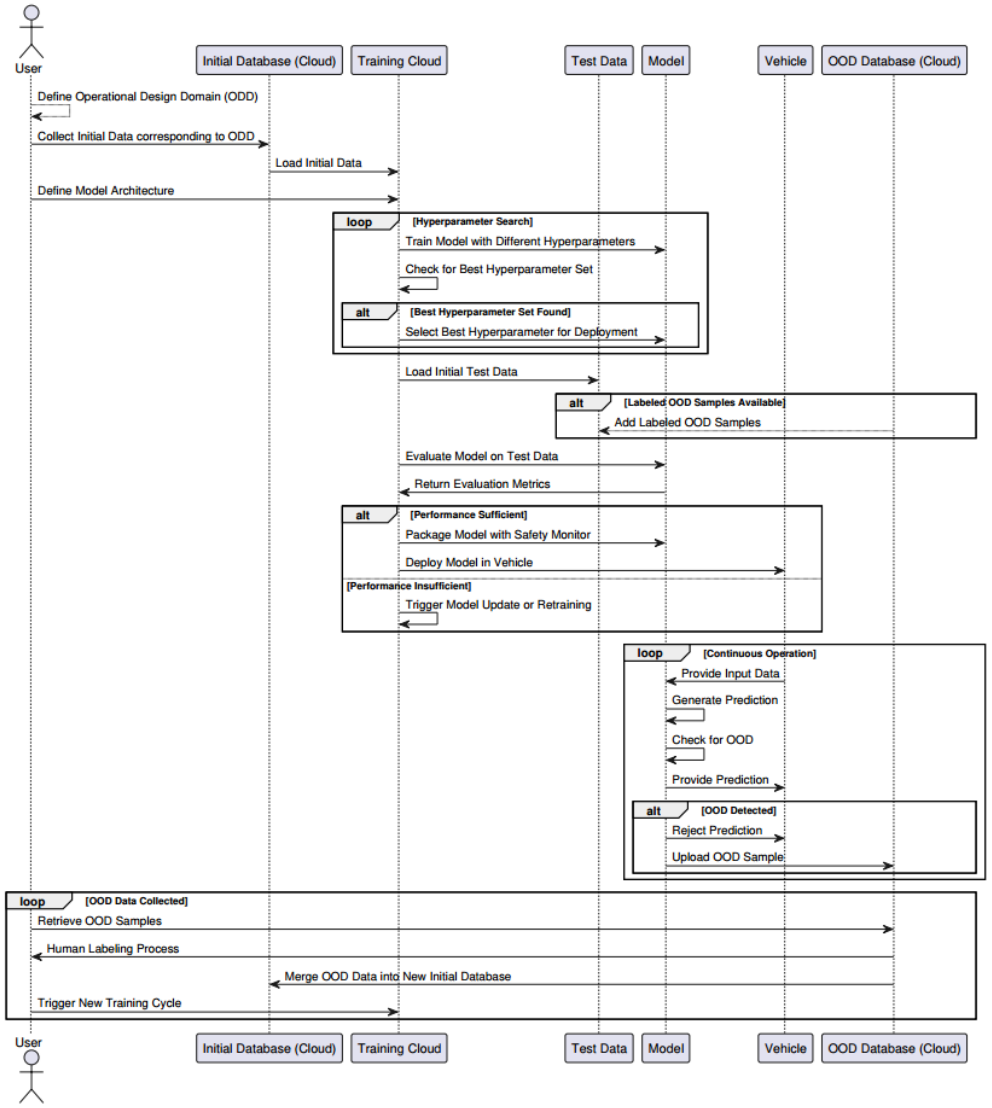


Abbildung 4.22: ARTEMIS Modell

Die Cloud-Infrastruktur fungiert als zentraler Enabler für diesen Prozess, indem sie skalierbare Datenspeicherung für große Datenmengen aus Fahrzeugflotten, dynamische Ressourcenallokation für rechenintensive Trainingsprozesse und vereinfachte Homologation durch zentrale Update-Verteilung ermöglicht. Besonders wichtig ist die Gewährleistung fleet-weiter Konsistenz durch simultane Updates aller Fahrzeuge. Das Monitoring erfolgt über zwei komplementäre Strategien. Für Aktuator-gekoppelte Systeme mit direkter Nutzerinteraktion eignet sich der Shadow-Mode, bei dem Modellvorhersagen mit tatsächlichen Aktuator-Werten verglichen und bei Abweichungen neue Trainingsdaten generiert werden. Für Perzeptionsaufgaben ohne direktes Nutzerfeedback ist hingegen eine eingebaute OOD-Detektion wie AURORA erforderlich, die präventiv Fehler vor der Weiterverarbeitung verhindert. Theoretisch fundiert

ist das Modell durch die Unterscheidung zwischen Near-OOD (geringfügige Verteilungsverschiebungen innerhalb der ODD) und Far-OOD (signifikante Abweichungen außerhalb der ODD), wobei formale Grenzwerte für die Bewertung der Verteilungsdistanz etabliert werden. Die Integration der ODD-Konzepte ermöglicht eine systematische Behandlung der Current Operational Domain (COD) und deren Abgrenzung zur spezifizierten ODD, wie folgende Abbildung zeigt 4.23:

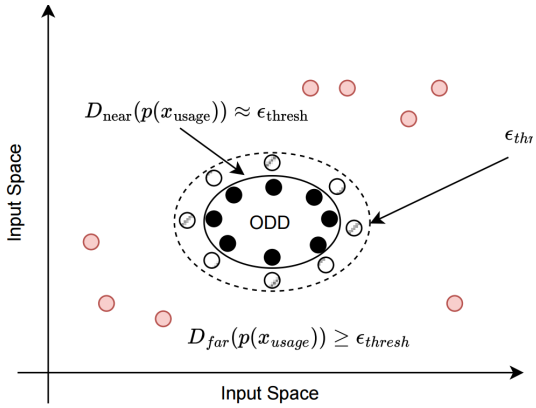


Abbildung 4.23: OOD Formalismus in Zusammenspiel mit der ODD

ARTEMIS positioniert sich als kritischer Baustein für die Entwicklung einer neuen Generation adaptiver, sicherheitskritischer CPS. Das Modell überbrückt erfolgreich die Lücke zwischen akademischer OOD-Forschung und industrieller Anwendung durch strukturierte Prozesse für den Umgang mit Unsicherheit, compliance-konforme Update-Strategien und skalierbare Lösungen für Fahrzeugflotten. Die proaktive Herangehensweise ermöglicht kontinuierliche Systemverbesserung statt reaktiver Fehlerbehebung und ebnet den Weg für langfristig sichere, zuverlässige und anpassungsfähige ML-enabled CPS.

4.9 Monitoring Strategien

In Cyber-Physical Systems (CPS) bildet adaptives Monitoring eine essenzielle Grundlage, da es die kontinuierliche Anpassung von Überwachungsmechanismen an sich verändernde Systemzustände und Umweltbedingungen ermöglicht und somit sowohl die Zuverlässigkeit als auch die Sicherheit der Systeme gewährleistet. Im Rahmen des Projekts sind mehrere Monitoring Strategien erforscht und entwickelt worden, um besonders im automobilen Kontext die Sicherheit unterschiedlicher Systeme in Echtzeit überwachen zu können. Hierbei spielt die holistische Sicht auf die Sicherheit (Saftey und Security), und dementsprechend auch ganzheitliches Monitoring Strategien, eine entscheidene Rolle, um auch in unvorhergesehenen Zustände jeweils Mitigationsstrategien auswählen und nutzen zu können.

4.9.1 Adaptives Monitoring

Mit zunehmenden Anspruch von Softwareumfängen in automatisierten Fahrzeugen, etwa durch Service-orientierte Architekturen und Virtualisierung, wächst auch die Notwendigkeit für eine kontinuierliche Überwachung. Klassische, statisch konfigurierte Monitoring-Lösungen stoßen hier schnell an ihre Grenzen: Wird zu viel Information gesammelt, entstehen unnötige Datenlasten und Ressourcenverbrauch; wird zu wenig erfasst, können sicherheitsrelevante Vorfälle unentdeckt bleiben. Potentiell safety-kritische Zustände in Komponenten sollten dennoch vor dem eintreten etwaig nachgelagerter Problematiken zu erkennen sein.

Adaptives Monitoring [40] setzt genau an diesem Punkt an. Es erlaubt die flexible Anpassung der Überwachung an den aktuellen Systemzustand oder die vorherrschenden Risiken. So kann im Normalbetrieb eine ressourcenschonende Datensammlung erfolgen, während im Verdachtsfall gezielt zusätzliche Informationen aktiviert werden. Dadurch wird nicht nur die Effizienz erhöht, sondern auch die Reaktionsfähigkeit auf Fehlverhalten und Anomalien deutlich verbessert.

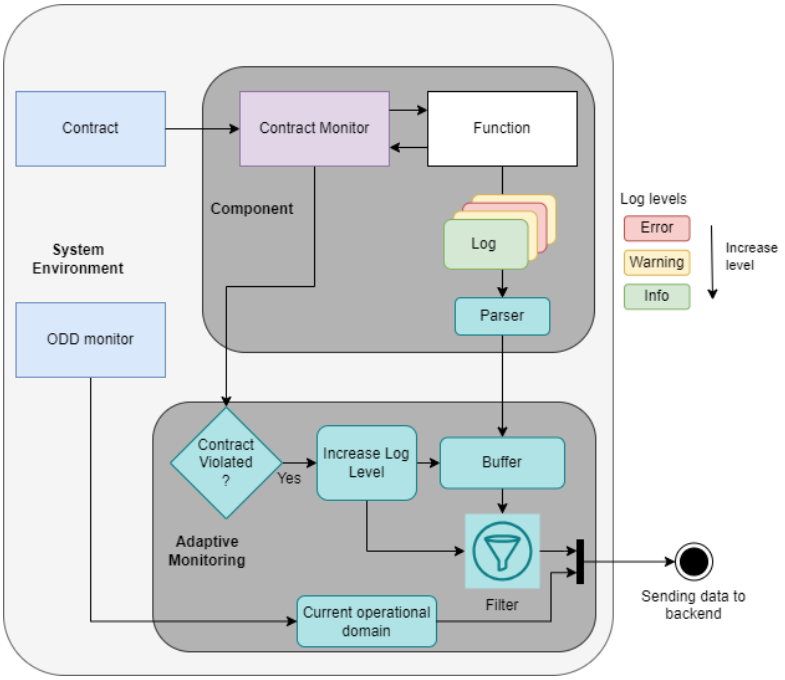
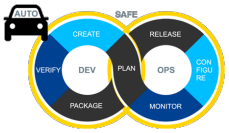


Abbildung 4.24: Konzept des adaptiven monitorings

Als Reaktionsauslöser zum schalten der verschiedenen Monitoringstufen können entweder relevante Kontextdaten dienen, oder Verletzungen von Contracts, welche eine Nichterfüllung der Komponentenspezifikation implizieren (siehe Abb. 4.24). Safety-Kontrakte können mit verschiedenen Spezifikationssprachen formuliert werden, wie z.B. Signal Temporal Logic (STL) oder Linear Temporal Logic (LTL). Diese Kontrakte definieren die erwarteten Verhaltensweisen und Sicherheitsanforderungen der Komponenten. Contracts bestehen aus Annahmen über die Umgebungsbedingungen und Garantien über das Verhalten des Modells. Bei einer Verletzung



dieser Kontrakte kann das Monitoring-System automatisch in einen höheren Überwachungsmodus wechseln, um detailliertere Daten zu erfassen und potenzielle Probleme frühzeitig zu erkennen.

Ein weiteres Kernelement des adaptiven Monitorings ist das Puffern von aktuell nicht als relevant betrachteten Daten. Diese werden zunächst lokal gespeichert und können bei Bedarf, sobdald das Log-level erhöht wird, zur nachträglichen Analyse herangezogen werden. Dies ermöglicht eine retrospektive Untersuchung von Vorfällen und unterstützt die Ursachenanalyse, ohne dass ständig große Datenmengen übertragen werden müssen.

Insgesamt trägt adaptives Monitoring entscheidend dazu bei, die Balance zwischen Sicherheit und Effizienz in CPS zu gewährleisten. Es schafft die Grundlage für eine nachhaltige Absicherung anspruchsvoller Systeme, indem es sowohl den steigenden Anforderungen der Normen und Standards als auch den begrenzten Ressourcen in eingebetteten Umgebungen gerecht wird.

Ein weiterer Beitrag zu Monitoring ist die zentralisierten Überwachung des aktuellen Betriebsbereichs, der Current Operational Domain (COD). Eine spezielle Komponente kondensiert hierbei die relevanten Systemaspekte und Umgebungsbedingungen, was zu einer harmonisierten Implementierung von Operational Design Domains (ODDs) führt.

Die Grundlage hierfür bildet eine gemeinsame Taxonomie, z. B. abgeleitet aus ISO 34503 und ISO 34504, die von Funktionsentwicklern und -Verantwortlichen zur Spezifikation der ODDs für die Funktionen genutzt werden. Die Feststellung der COD ist davon separiert, wodurch widersprüchliche Auffassungen über zu aktivierende Funktionen vermieden werden. Die zentralisierte und vereinheitlichte Überwachung der COD / ODD dient auch als allgemeine Beschreibung der aktuellen Situation, was zum Verständnis von Ausfällen und kritischem Verhalten von Komponenten im Fahrzeug beitragen kann. Wenn sich eine Komponente nicht wie gewünscht verhält, kann es hilfreich sein zu wissen, unter welchen Bedingungen sie verwendet wurde. Wenn sich Fehler oder kritische Situationen in bestimmten Betriebsbereichen häufen, könnte dies ein Hinweis darauf sein, dass eine Aktualisierung erforderlich ist. Entsprechend wurde das ODD/COD-Monitoring mit dem adaptiven Sammeln von Informationen verknüpft.

Shadow Mode

Darüber hinaus kann insbesondere während eines Updateprozesses eine Multikomponentenanalyse sinnvoll sein, um den geeignetsten Updatekandidaten zu herauszuarbeiten und das Einführen von Fehlverhalten durch Updates zu verhindern.

Hierbei muss auch insbesondere Augenmerk darauf gelegt werden, wenn sicherheitskritische Funktionen durch Machine Learning (ML)-Modelle realisiert werden. Da klassische Testverfahren nicht alle möglichen Betriebsszenarien abdecken können, wurde ein Ansatz herausgearbeitet, bei dem neue Modellversionen zunächst im Shadow Mode mit Hilfe unterschiedlicher Softwaremonitore getestet werden können.

Hierbei laufen die Shadow Versions (SVs) parallel zur aktuell aktiven Version (Active Version, AV), beeinflussen jedoch nicht die Steuerung des realen Systems. Dieser Ansatz ermöglicht es, reale Umgebungsbedingungen zu nutzen, ohne die Betriebssicherheit zu gefährden. Kern des Konzepts ist die Multi-Version Execution (MVE), welche parallele Ausführungen verschiedener Komponentenzustände erlaubt, welche dann auch selektiv einer Sicherheitsbewertung

durch eine zuvor getroffene Komponentenspezifikation unterzogen werden können. Das System ist im folgenden dargestellt 4.25:

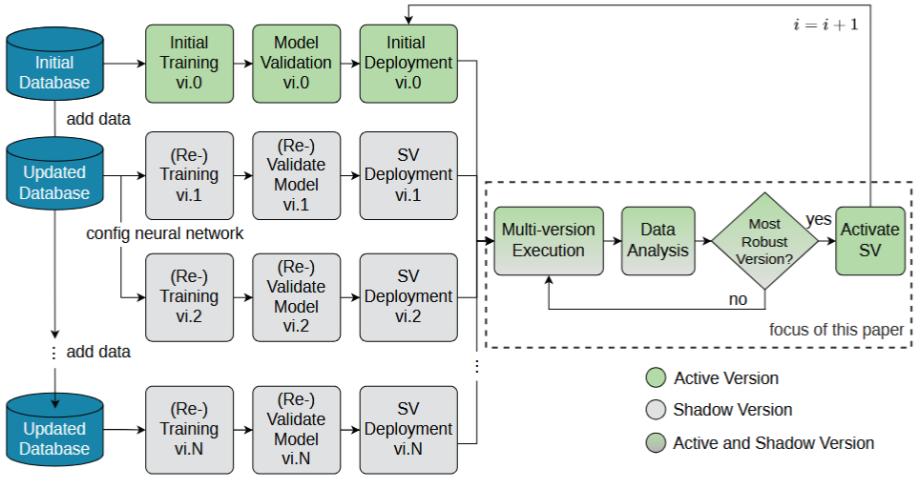


Abbildung 4.25: Shadowmode Runtime

Ein zentrales Element stellt die Monitoring-Funktion dar:

Die Funktionsüberwachung basiert auf den genannten Spezifikationen (Contracts), die in diesem Projekt mittels STL formalisiert werden. Das Monitoring überprüft zur Laufzeit, inwieweit diese Contracts eingehalten werden. Dabei wird die relative Robustheit als Bewertungsmaß eingesetzt. Sie erlaubt eine quantitative Aussage darüber, wie weit ein Modell von der Erfüllung bzw. Verletzung der spezifizierten Eigenschaften entfernt ist.

Diese Robustheitswerte werden kontinuierlich berechnet und in Histogrammen zusammengeführt. So entsteht sowohl eine On-board Analyse (lokale Bewertung auf dem Zielsystem, inklusive Deaktivierung fehlerhafter SVs) als auch eine Off-board Analyse (Zusammenführung von Monitoring-Daten vieler Systeme in einer Cloud-Datenbank). Die Off-board Analyse ermöglicht zudem, durch die gewonnenen Daten kritische Situationen speziell für ML-basierte Komponenten in die Trainingsdatenbank aufzunehmen und so die Modelle gezielt weiterzuentwickeln.

Die vorgestellte Methode integriert sich nahtlos in DevOps/MLOps-Workflows und damit auch in den ADSO Prozess und bietet eine Grundlage für kontinuierliche Sicherheitsbewertungen von ML-basierten Funktionen. Besonders die Monitoring-Funktion stellt hierbei den Schlüssel dar, da sie einen laufenden, vergleichenden Nachweis der Vertragserfüllung ermöglicht und so eine objektive Entscheidungsgrundlage für Updates schafft.

Darüber hinaus kann eine Multikomponentenanalyse während des Updateprozesses hilfreich sein, um verschiedene Bewertungsdimensionen (z. B. Robustheit, Stabilität, Smoothness) gemeinsam zu betrachten und den geeignetsten Updatekandidaten systematisch auszuwählen [42].

Datennutzung und Auswertung

Moderne Fahrzeuge sind zunehmend softwaredefiniert und vernetzt, wodurch große Mengen an Daten aus unterschiedlichen Quellen anfallen – sowohl aus der Entwicklung (Dev) als auch aus dem Betrieb (Ops). Um diese Daten sinnvoll zu nutzen, und ein holistisches Monitoring-Konzept zu ermöglichen, das Daten systematisch nach Kontexten und Zuständen gliedert, benötigt man hierfür ein aussagekräftiges Datenmodell.

Aus den Fahrzeugen gesammelte Daten sollten dabei nicht isoliert betrachtet werden, sondern insbesondere im Zusammenhang mit den relevanten Aspekten aus der Entwicklungszeit. Folgende Sichten auf die relevanten Daten können unterschieden werden:

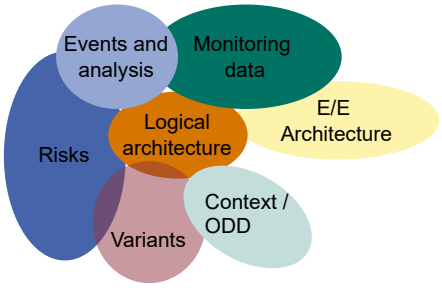
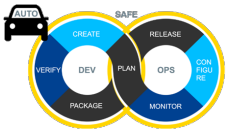


Abbildung 4.26: Schematische Darstellung der Informationsverknüpfung in einem graphbasierten Datenmodell

- **Konfiguration und Architektur:** Fahrzeugvarianten, Softwareversionen, logische Softwarekomponenten und E/E-Architektur bilden die Grundlage, um Monitoring-Daten einordnen und analysieren zu können.
- **Kontext und Operational Domain:** Fahrzeugstatus und weitere Attribute der aktuellen COD bzw. der ODD liefern den aktuellen Kontext der Systeme.
- **Risiken:** Maßnahmen und bekannte Risiken, z.B. aus einer Analyse in der Entwicklungszeit lassen sich den jeweiligen Systemen und Komponenten zuordnen.
- **Monitoring Daten, Events und Analyse:** Gesammelte Daten aus Fahrzeugen und externe Reports, zum Beispiel aus Vertragswekstätten, ergänzen den Kontext / COD und ermöglichen die Analyse hinsichtlich Problemen (z.B. SOTIF-Events) und eine Verknüpfung zwischen internem Zustand und externen Beobachtungen.

Zur Verknüpfung dieser unterschiedlichen Datenquellen und Zustände wird ein graphbasiertes Datenmodell verwendet [41] (siehe Abb. 4.26). Dieses Modell bildet alle Informationen und deren Beziehungen in einer einheitlichen, ontologie-ähnlichen Struktur ab. So können operative Daten (z.B. Logs) mit Entwicklungsinformationen (z.B. Softwareversionen) verknüpft werden, um Muster, Zusammenhänge oder Abhängigkeiten zu erkennen. Ziel ist die ganzheitliche Einordnung und Verknüpfung von Daten, sodass Entscheidungen oder Analysen immer



im richtigen Kontext stattfinden und auch dem jeweiligen Betriebsbereich des Fahrzeugs der Operational Design Domain (ODD) zugeordnet werden können.

Integration in den ADORe-Stack

Im Rahmen der Kooperation zwischen KIT und DLR konnten die im Projekt entwickelten Konzepte erfolgreich in einem praxisnahen Use Case demonstriert werden. Dabei stand insbesondere der Contract-Monitor im Fokus, der Spezifikationsverletzungen zur Laufzeit erkennt und bei Bedarf automatisch das Log-Level hochsetzt. Auf diese Weise werden sicherheits- und sicherheitsrelevante Abweichungen nicht nur frühzeitig detektiert, sondern auch durch eine adaptive Erhöhung der Monitoring-Granularität besser nachvollziehbar gemacht. Ergänzend wurde die aktuelle Operational Design Domain (ODD) aus den Kontextinformationen berechnet, sodass die Bewertung des Systemverhaltens stets in Bezug auf den konkreten Einsatzbereich erfolgen konnte.

Für die Umsetzung wurde das ADORe-Stack des DLR genutzt, das eine leistungsfähige Plattform für automatisierte Fahrfunktionen bietet, in welche nun auch teile der Monitoring Konzepte eingeflossen sind. Durch diese enge Verzahnung entstand ein gemeinsamer Demonstrator auf Basis eines automatisierten Fahrzeugs, der nicht nur die technische Machbarkeit, sondern auch den Mehrwert eines holistischen, adaptiven Monitorings im automobilen Kontext sichtbar gemacht hat. Die gesammelten Informationen wurden mithilfe des Telemetrie-Systems von ADORe erfasst und dargestellt, um eine umfassende Analyse des Systemverhaltens zu ermöglichen. Spezifische Dashboards zeigen den aktuellen Zustand der Contracts und die ODD-Informationen an, sodass Bediener jederzeit den Überblick über die Systemintegrität behalten.

4.9.2 Monitoring in ADORe

Überblick und Ziele

Ein Schwerpunkt liegt auf der Erweiterung von ADORe (https://github.com/eclipse/adore) und ROS (Robot Operating System), um ein umfassendes Framework für die Forschung, Entwicklung und Bereitstellung autonomen Fahrens bereitzustellen, das den ADSO-Updateprozess unterstützt. Für dieses Projekt wurden mehrere Tools entwickelt, in ADORe integriert und als Open Source veröffentlicht, um Kernfunktionen bereitzustellen, die die ADSO-Ziele erreichen und gleichzeitig eine Grundlage für weitere Forschung in diesem Bereich schaffen.

Ein solches Tool, das mit Unterstützung dieses Projekts und seiner Partner als Open Source bereitgestellt wird, ist der ROS2 Observer (https://github.com/DLR-TS/ros2_observer). Der ROS2 Observer bietet Tracing- und Beobachtungsfunktionen, die Kernel-Level-Tracing über LTTng und das offizielle ROS-Projekt ros_tracing (https://github.com/ros2/ros2_tracing), Echtzeit-Knoten- und Programminspektion sowie strukturierte Protokollierungssysteme umfassen, um Online-Einblicke in das Systemverhalten und die Leistung in verteilten Automotive-Computing-Umgebungen zu liefern.

Online-Überwachung mit dem ADORe Model Checker

Ein weiteres Open-Source-Tool, das mit Unterstützung dieses Projekts und seiner Partner veröffentlicht wurde, ist der ADORe Model Checker (https://github.com/DLR-TS/adore_model_checker). Der ADORe Model Checker ist ein hybrides CTL-Modellprüfungstool zur formalen Verifikation von Sicherheitseigenschaften. Der Model Checker ist erweiterbar, sodass zusätzliche Sicherheitsvorschläge hinzugefügt werden können, um zukünftige Forschung zu unterstützen. Er unterstützt die Online-Überwachung von Live-ROS2-Daten von Topics und die Offline-Analyse von Bag-Dateien mit konfigurierbaren Sicherheitseigenschaften und flexiblen Quellen.

Die Leistungsfähigkeit des Model Checkers wird durch formale Verifikationsergebnisse demonstriert, die in Abbildung 4.27 aus einem Online-Szenariolauf stammen. Diese Ergebnisse veranschaulichen die Fähigkeit des Tools, Sicherheitseigenschaften in verschiedenen Szenarien zu analysieren und klare Verifikationsergebnisse für Anwendungen im autonomen Fahren zu liefern.

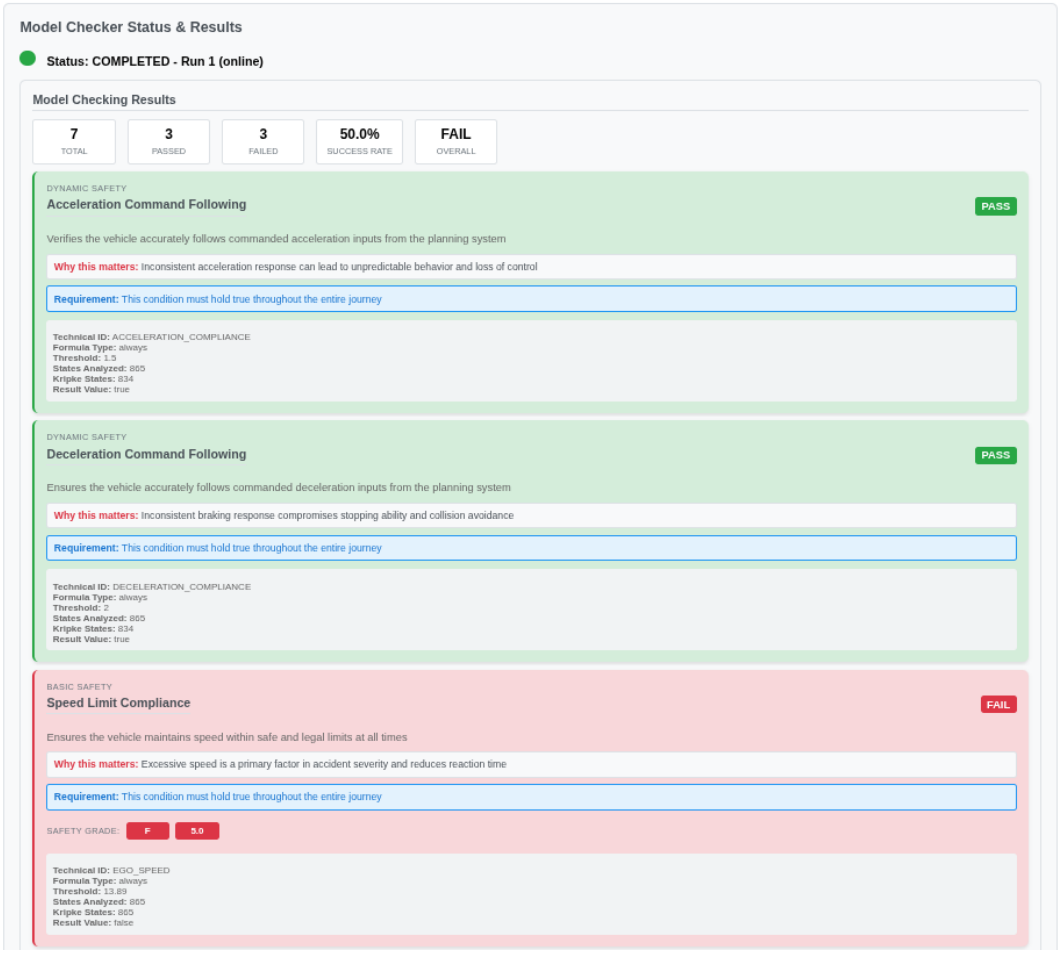
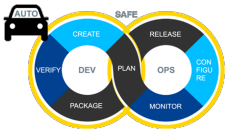


Abbildung 4.27: ADORe Model Checking Ergebnisse

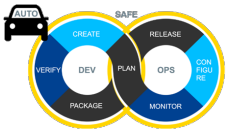


Die in Abbildung 4.28 gezeigte Ausgabe des ADORe Model Checker demonstriert die umfassenden Verifizierungsfunktionen des Tools.

```
============================================================
DYNAMIC VEHICLE MONITORING RESULTS
============================================================
SUMMARY:
  Total Propositions: 7
  Analyzed: 6
  Passed: 3
  Failed: 3
  Success Rate: 50.0%
  Overall Result: FAIL
DETAILED RESULTS:
------------------------------------------------------------
  EGO_SPEED                      : FAIL     (865 states, 865 Kripke states)
                                     Max velocity: 9.91 m/s, Avg velocity: 3.37 m/s
                                     Speed threshold: 13.89 m/s
                                     States with data: 834, without data: 31
                                     Valid evaluations: 865, failed evaluations: 0
  NEAR_GOAL                      : FAIL     (865 states, 568 Kripke states)
                                     Goal position: (606471.04, 5797161.11)
                                     Final vehicle position: (606764.68, 5797183.07)
                                     Min distance to goal: 117.81m
                                     Final distance to goal: 294.46m
                                     Distance threshold: 5.00m
                                     States with data: 0, without data: 297
  ACCELERATION_COMPLIANCE        : PASS     (865 states, 834 Kripke states)
                                     Max error: 0.100 m/s², Avg error: 0.009 m/s²
                                     Max measured: 1.01 m/s², Max commanded: 1.01 m/s²
                                     Avg measured: 0.64 m/s², Avg commanded: 0.64 m/s²
                                     Acceleration events: 327, Violations: 0
                                     Compliance rate: 100.0%
  DECELERATION_COMPLIANCE        : PASS     (865 states, 834 Kripke states)
                                     Max error: 0.075 m/s², Avg error: 0.004 m/s²
                                     Max measured: -1.50 m/s², Max commanded: -1.50 m/s²
                                     Avg measured: -0.64 m/s², Avg commanded: -0.64 m/s²
                                     Deceleration events: 451, Violations: 0
                                     Compliance rate: 100.0%
  LANE_KEEPING                   : FAIL     (865 states, 565 Kripke states)
                                     Max distance: 15.990 m, Avg distance: 0.596 m
  TIME_TO_COLLISION              : NO_DATA  (0 states, 0 Kripke states)
                                     No TTC calculatable
  SMOOTH_STEERING                : PASS     (865 states, 834 Kripke states)
                                     Max steering rate: 0.592 , Min steering rate: 0.000
                                     Violations:
```

Abbildung 4.28: Die Verifizierungsausgabe des ADORe Model Checker zeigt eine detaillierte Analyse der Sicherheitsvorschläge in einem menschenlesbaren Format mit umfassenden Metriken für jede bewertete Eigenschaft.

Der ADORe Model Checker liefert umfassende Ergebnisse in menschenlesbaren und ma-



schinenlesbaren Formaten, um unterschiedlichen Anwendungsfällen und Integrationsanforderungen gerecht zu werden. Die menschenlesbare Ausgabe präsentiert die Verifizierungsergebnisse in einem strukturierten, leicht verständlichen Format mit klaren Statusanzeigen (PASS/FAIL/NO_DATA), detaillierten Kennzahlen für jeden Sicherheitsvorschlag und zusammenfassenden Statistiken, einschließlich Erfolgsraten und Gesamtsystemstatus. Dieses Format ermöglicht es Stakeholdern, die Systemleistung schnell zu bewerten und spezifische Problembereiche zu identifizieren. Gleichzeitig generiert das Tool eine maschinenlesbare JSON-Ausgabe mit denselben Verifizierungsdaten in einem strukturierten Format, das sich für die automatisierte Verarbeitung, die Integration in kontinuierliche Integrationspipelines wie den ADSO-Updateprozess sowie die Speicherung und Weiterleitung an Telemetriedatenbanken eignet. Dieser duale Ausgabeansatz stellt sicher, dass die Verifizierungsergebnisse sowohl für menschliche Bediener sofort umsetzbar als auch in automatisierte Sicherheitsüberwachungs- und Berichtssysteme integriert werden können.

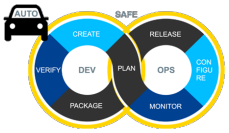
Zusätzlich kann die propositionsbasierte Methode mit einem Sicherheitsscore ergänzt werden. Der Sicherheitsscore berücksichtigt ähnliche Metriken wie Statusindikatoren, bietet aber zusätzliche Funktionen, die ihn im Vergleich zum Booleschen Wert (PASS/FAIL) flexibler machen. Basierend auf intensiven Tests und dem verfügbaren Wissen über ein System werden akzeptable Werte des Systems an einen normalisierten Wert von 1,0 für ausgezeichnete Sicherheit bis 0,0 für unzureichende Sicherheit gebunden. Darüber hinaus implementiert die Methode Schutzbedingungen. Durch die Verwendung von Schutzbedingungen fällt das System nicht direkt aus, wenn die Grenzwerte überschritten werden. Vielmehr können die Sicherheitsbedingungen basierend auf dem jeweiligen System und seinen Anforderungen definiert werden. Darüber hinaus können die Sicherheitsbedingungen in Cluster unterteilt werden. Für diese Cluster und deren Kombination können unterschiedliche Regeln definiert werden. Diese Methode ermöglicht die Definition komplexer Kombinationen und bietet somit mehr Möglichkeiten als die propositionsbasierte Methode. Durch die Quantifizierung von Tests können verschiedene Algorithmen gegeneinander bewertet werden. Derzeit überschreibt der Sicherheitsscore nur einen einzelnen Testtyp. Die Zusammenführung verschiedener Tests basiert weiterhin auf Propositionen.

Online-Überwachung mit dem ROS2 Observer

Wie bereits erwähnt, bietet der ROS2 Observer (https://github.com/DLR-TS/ros2_observer) Tracing auf Kernel-Ebene, Echtzeit-Knoten- und Programminspektion sowie strukturierte Protokollierungssysteme. Diese Funktionen liefern Einblicke in das Systemverhalten und die Leistung in verteilten Automotive-Computing-Umgebungen. Der ROS2 Observer ermöglicht die Beobachtung von Themenkommunikation, Nachrichtenflüssen und Knotenabhängigkeiten. Die Integration von LTTng-Tracing in die ROS-Knotenbeobachtung unterstützt detaillierte Analysen durch die Konvertierung von LTTng-Trace-Formaten.

Erfassung und Weiterleitung von Telemetriedaten mit Rsyslog

Wie in D5.2 und D6.2 erläutert, bietet ADORe die Rsyslog-Integration über einen ROS-Knoten namens `ros2_rsyslog` (https://github.com/eclipse-adore/adore/tree/develop/ros2_



workspace/src/ros2_syslog). Der ros2_rsyslog-Knoten bildet eine Brücke zwischen den von ROS generierten Datenmengen und aussagekräftigen Telemetrieergebnissen. Das Rsyslog-basierte Telemetriesystem kann außerdem Überwachungsdaten und Berichte filtern und weiterleiten, die vom ROS2-Observer, dem ADORe Model Checker und anderen Überwachungstools generiert wurden.

Diese Telemetriedaten dienen sowohl der Echtzeit-Systemüberwachung als auch der langfristigen Systemverbesserung. Bei der Online-Überwachung ermöglicht der kontinuierliche Strom von Verifizierungsergebnissen, Leistungskennzahlen und Sicherheitsbewertungen die sofortige Erkennung von Anomalien, Sicherheitsverletzungen oder Leistungseinbußen. So können Bediener Korrekturmaßnahmen ergreifen oder automatisierte Sicherheitsmaßnahmen auslösen, bevor kritische Fehler auftreten. Die Echtzeitüberwachung ist unerlässlich für die Aufrechterhaltung der Betriebssicherheit in dynamischen autonomen Fahrumgebungen, in denen sich die Bedingungen schnell ändern können. Gleichzeitig erstellen die erfassten und gespeicherten Telemetriedaten einen umfassenden Verlaufsbericht, der analysiert werden kann, um Muster, Fehlermodi und Leistungstrends über längere Zeiträume zu identifizieren. Diese gesammelten Daten sind für Initiativen zur Systemverbesserung von unschätzbarem Wert und ermöglichen es Forschern, Sicherheitsschwellenwerte zu verfeinern, Algorithmen zu optimieren, neue Funktionen anhand realer Betriebsdaten zu validieren und robustere Verifizierungseigenschaften zu entwickeln. Die Kombination aus sofortiger Überwachung und langfristiger Datenanalyse schafft eine Feedbackschleife, die die Systemzuverlässigkeit und Sicherheitsleistung kontinuierlich verbessert und den ADSO-Aktualisierungsprozess realisiert.

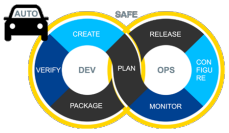
4.10 Kontinuierliche Integrationspipeline für System-of-Systems im Automobilbereich

Ziel war die Entwicklung einer CI-Pipeline zur simulativen Validierung von Automobilsystemen, insbesondere für SoS-Architekturen. Die Pipeline orientiert sich an den Anforderungen aus [75] und integriert bewährte CI-Praktiken aus [37] und [45]. Sie fördert kollaborative Entwicklung, kontinuierliche Tests und gewährleistet technische Robustheit sowie die Einhaltung von Industriestandards.

Die CI-Pipeline folgt klassischen Phasen der Build-Verifizierung und des Softwaretests. Builds starten lokal (private Builds) und laufen später gemeinsam, analog zum Entwicklerworkflow. Leichte Tests folgen direkt für schnelles Feedback, schwere Tests erfolgen später, meist über Nacht, um Entwicklungsblockaden zu vermeiden. Die Trennung optimiert Ressourcennutzung und Feedback, wie in [37] empfohlen. Leichte Tests prüfen Einzelbeiträge, schwere bewerten das Systemverhalten.

Die Pipeline erfüllt Anforderungen der Automobilindustrie, etwa aus Automotive SPICE und dem V-Modell. Sie unterstützt Bottom-up-Tests von Unit- bis Systemebene. Eine vom ITIV gehostete GitLab-Instanz dient als zentrales Versionskontrollsystem für verteilte Teams. Das in [75] vorgestellte "Create-Package-Commit"-Muster standardisiert die Artefaktbehandlung: Entwickler erstellen, verpacken (z. B. in Containern) und committen ihre Ergebnisse ins gemeinsame Repository.

Das Testobjekt (engl. Subject-under-Test, SUT) ist ein automatisches Spurhaltesystem



(ALKS) gemäß UN-Regelung Nr. 157 [85], bestehend aus dem Adaptivem Abstrandsregeltempomat (engl. Adaptive Cruise Control, ACC) für die Längsführung, dem Notbremssystem (engl. Automated Emergency Brake System, AEBS) zur Kollisionsvermeidung und dem Spurhalteassistent (engl. Lane Keeping Assistant, LKA) für Querführung.
Die Systeme sind als ROS 2-Knoten implementiert und kommunizieren über Themen (z. B. "Ego-Geschwindigkeit" = 60 km/h). Der LKA-Knoten läuft isoliert in einem Docker-Container auf Basis eines ROS 2 Humble-Images.
Das System folgt teils dem vertragsbasierten Design: Verträge (Annahme-Garantie) formalisieren Verhalten, Schnittstellenprüfungen sichern dessen Umsetzung im Code.

Unterschiedliche Testtypen sind Teil der CI-Pipeline. Unit-Tests validieren Parameter wie den maximalen Lenkwinkel des LKA und nutzen ggf. simulierte Abhängigkeiten. Komponententests prüfen die Knotenkommunikation. Der LKA und ein Testknoten sind containerisiert; letzterer sendet realistische Daten. Code-basierte Systemtests überprüfen die Gesamtfunktionalität per Unittest-Framework. Szenario-basierte Systemtests nutzen CARLA [36], ScenarioRunner [18] und Szenarien aus [13] zur Ausführung. Ein weiterer Beitrag [74] beschreibt darüber hinaus eine Strategie zur Auswahl relevanter Tests. Das Paper präsentiert ein sicherheitsorientiertes Klassifikationsframework für Software-Updates in Fahrzeugen, das durch gezielte Auswahl von Regressionstests die Time-to-Market verkürzt, ohne die funktionale Sicherheit zu gefährden.

Abbildung 4.29 zeigt die CI-Pipeline-Struktur. Die Phasen sind:

1. Bereinigung (engl. Cleanup): Entfernt veraltete Container, Images und Netzwerke.
2. System-Builds (engl. System-level Builds): Erstellt Container für Teilsysteme (z. B. LKA).
3. Statische Tests (engl. Static Tests): Prüfen Schnittstellen auf Vertragskonformität.
4. Unit-Tests auf System-Ebene (engl. System-level Unit Tests): Für jedes Teilsystem.
5. Komponententests (engl. Communication Tests): Validieren Kommunikation zwischen LKA und Testknoten.
6. SoS-Build: Erstellt das ALKS-Gesamtsystem.
7. Code-basierte Systemtests auf SoS-Level (engl. System-of-Systems Level: Code-based System Tests): Mit Unittest-Framework.
8. Szenario-basierte System Tests (engl. System-of-Systems Level: Scenario-based System Test): Simulation mit CARLA und ScenarioRunner.

Die Implementierung der CI-Pipeline erfolgte schrittweise: Zunächst wurden Testartefakte wie Unit-Tests und Schnittstellenprüfungen entwickelt. Anschließend wurde die CI-Infrastruktur aufgebaut, inklusive Serverzugriff und Integration der Tests zur automatisierten Ausführung und Berichterstellung.

Folgende Phasen wurden im Rahmen des Projekts umgesetzt: Bereinigung, System-Build für LKA, Statische, Schnittstellenprüfung, Unit-Tests für ACC, AEBS und LKA, Komponententest für LKA und Code-basierter Systemtest für ALKS. In Zukunft sollten darüber hinaus

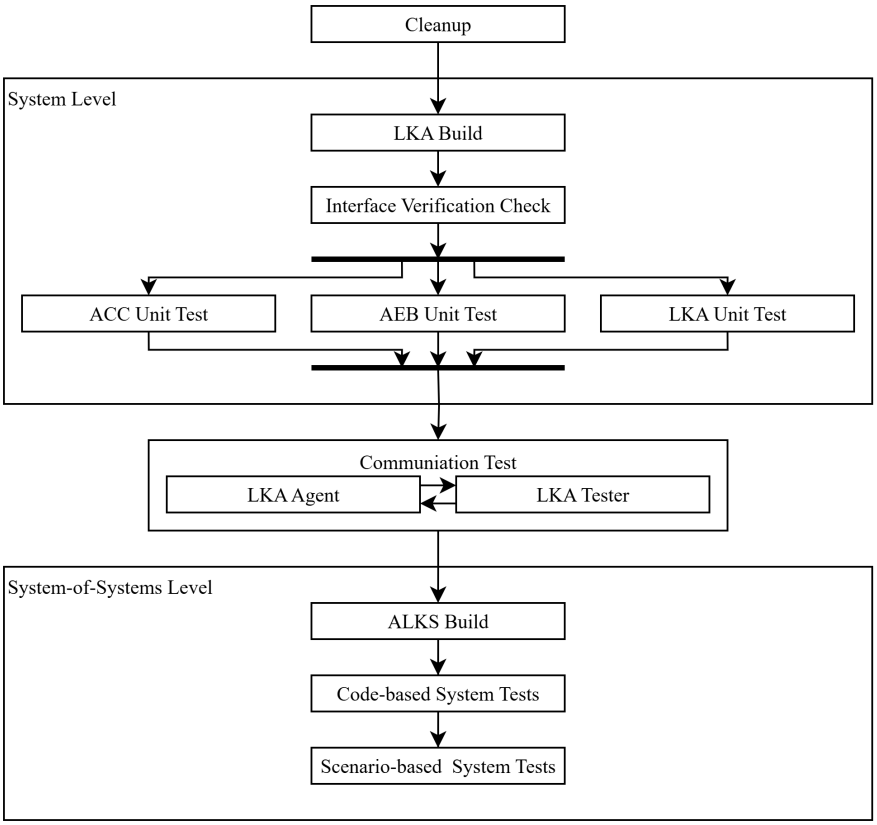
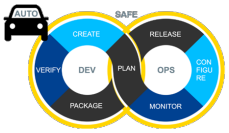


Abbildung 4.29: GitLab CI-Pipeline zur Testung des ALKS als System-of-Systems



der SoS-Build und szenariobasierte Systemtests untersucht werden. Ein Ansatz hierzu ist die Ausführung von CARLA und ScenarioRunner in separaten Containern, wobei hier die Kommunikation zwischen den Containern noch eine Herausforderung darstellt.

In diesem Projektteil wurde eine CI-Pipeline für die Verifizierung von System-of-Systems im Automobilbereich entworfen. Sie integriert Industrie-Standards, unterstützt verteilte Teams und erfüllt Automotive SPICE. Die Pipeline ermöglicht strukturierte Tests, nutzt vertragsbasierte Designprinzipien und Simulationen realistischer Szenarien. Das Ergebnis ist eine robuste, skalierbare Infrastruktur für sichere und effiziente Entwicklung.

4.11 Standardisierungsbedarfe

Der tiefgreifende Wandel in der Entwicklung sicherheitskritischer automobiler Systeme schlägt sich naturgemäß auch in den zugehörigen Regularien und Standards nieder, die der Typzulassung (Homologation) unterliegen. Normen und Regularien sind dabei gesetzliche oder gesetzähnliche Vorschriften, die die Anforderungen an Sicherheitsnachweise[1] definieren. Diese Normen und Regularien werden typisch in einer streng hierarchischen Art und Weise erlassen, ausgehend von (nahezu) weltweit vereinbarten Regularien die typisch innerhalb der Wirtschaftskommission der Vereinten Nationen wie UNECE (United Nations Economic Commission for Europe) vereinbart und dann schrittweise über das Europarecht in die Rechtsrahmen der Mitgliedstaaten übernommen werden.

Solche Regularien und Normen setzen dabei das 'Was' der Fahrzeugtypzulassung fest, d.h. sie definieren Anforderungen an Sicherheitsnachweise, die vom Hersteller eines Fahrzeugs erbracht werden müssen, damit die Typzulassung erfolgen kann. Standards, auf der anderen Seite, definieren das 'Wie' dieser Nachweise, d.h. sie legen technische und prozedurale Methoden fest, mit denen diese Nachweise erbracht werden können. Ideal stellt ein Standard den aktuellen Stand der Technik dar, d.h. die dort beschriebenen Verfahren und Methoden sind im Idealfall so vollständig und ausgereift, dass keine besseren Verfahren für diese Sicherheitsnachweise existieren. Aufgrund der langwierigen Einigungsprozesse die für die Erstellung eines Standards notwenidg sind und insbesondere angesichts der durch die höheren Autonomiegrade und die Möglichkeiten KI-basierter Systeme rasant steigenden Anforderungen an und Funtionsumfängen von modernen Fahrzeugen, sind Standards oft erweiterungsbedürftig, d.h. sie müssen an neue Technologien und Methoden für die Nachweisführung angepasst werden, um weiterhin aktuelle und einheitliche 'best-in-class' Verfahren für die Homologation zu erbringen.

Der durch höhere Automationsgrade und den Einsatz von KI-Methoden ermöglichte steigenden Funktionsumfang von Fahrzeugen bedingt offensichtlich Änderungen der damit verbundenen Sicherheitsnachweise. Die zugehörigen Anforderungen wurden bereits in einer Reihe von neuen Regularien und Normen aufgenommen, wie beispielsweise den UNECE Regularien UN R155 und UN R156, die Anforderungen zur Cybersecurity und zu Update-fähigen Systemen beinhalten. UN R157 (Äutomated Lane Keeping Systems) sowie die neuen EU ADS Regularien ('Automated Driving Systems') stellen seit Beginn der 2020er Jahre die ersten

[1] Mit dem deutschen Wort 'Sicherheit' ist hier sowohl funktionale Sicherheit (safety) als auch Cybersicherheit (Security) gemeint.

Regularien für hochautomatisierte Systeme bereits. Regularien für KI-basierte Systeme, wie beispielsweise der EU AI Act sind nicht zentral auf die Automobilindustrie und den Homologationsprozess ausgerichtet, betreffen diesen aber auch. Eine Fülle von Standards ist bereits erschienen oder in Vorbereitung, um auch die technische Umsetzung der in den Regularien geforderten Umsetzungen zu vereinheitlichen. Diese Standards sind jedoch zur Zeit noch oft generisch und konkrete Methoden, die den Stand der Technik darstellen, fehlen oft.

Ein Ziel des ADSO Projekts war die Identifikation von sogenannten Lücken in Standards, d.h. die Identifikation solcher Technologien, die für die Entwicklung und insbesondere die Sicherheitsnachweise moderner Fahrzeuge notwendig sind, die aber in den aktuellen Versionen der Standards noch nicht berücksichtig wurden und somit einen Erweiterungsbedarf dieser Standards aufzeigen, damit diese wieder dem Stand der Technik entsprechen. Der hierbei verfolgte Ansatz bestand darin, die im Projekt entwickelten Technolgoiebausteine auf die einzelnen, durch bestehende Standards erfolgten Nachweisverfahren abzubilden – d.h.einer Zuordnung von Technologien zu den Prozessschritten und/oder zu den durch die Regularien und Standards vorgegebenen Nachweisschritten, in denen diese Technologien einen Beitrag leisten – und einer folgenden Analyse, ob diese in den Standards abgebildet werden. Die dabei identifizierten Lücken sind Hinweise auf eine zu erfolgende Erweiterung der jeweiligen Standards.

4.11.1 'Assessors View' und 'Developers View'

Um diese Fragestellung vollständig zu bearbeiten wurden im Projekt zwei Sichtweisen festgelegt, die eine 'aus Sicht der Standards' (sogenannte Gutachter-Sicht bzw. Assessors View) und eine 'aus Sicht des Entwicklungs- und Nachweisprozesses' (sogenannte Entwicklersicht bzw. Developers View). In beiden Fällen wurden die für Sicherheitsnachweise hauptsächlich verwendeten vier sogenannten 'primary Standards' als Grundlage verwendet, die Verfahren sind jedoch auf nahezu jeden existierenden Standard anwendbar.

Die vier betrachteten primären Standards sind

- ISO 26262:2018 - Road Vehicles - Functional Safety.
- ISO 21448:2022 - Road Vehicles - Safety of the Intended Functionality (SOTIF)
- ISO/SAE 21434:2021 – Road Vehicles – Cybersecurity Engineering
- ISO 24089:2023 – Road Vehicles – Software Update Engineering

Im sogenannten 'Assessors View' wurden zunächst durch den Projektpartner SGS-TÜV sogenannte 'Work Product Lists' erstellt. Hierbei handelt es sich um Listen von durch die jeweiligen Standards verlangten Nachweisen (Daten, Argumentationen, Formulare, Artefakte aus dem Entwicklungsprozess, Testergebnisse, etc.) in den einzelnen zur Nachweisführung notwendigen Kategorien. Die in ADSO entwickelten Technologiebausteine wurden auf diese Listen abgeildet, d.h. es erfolgte eine Zuordnung, welche Technologiebausteine zu welchem Schritt einen Beitrag leistet.

Abbildung 4.30 zeigt einen Auszug aus der Zuordnung von Technolgoiebausteinen zu den Schritten der Work Product List des Standards ÏSO 26262:2018 - Road Vehicles - Functional Safety". Diese Zuordnung wurde für alle Technologiebausteine und jeweils alle vier

Work Product Mapping List

ISO 26262:2018 Part	Clause	Work Product	Technology Bricks contributing to this Work Product
6 Product Development at the Software Level	6-8 Software unit design and implementation	**6-8.5.1** Software unit design specification	Partly: System Modeling (see 6-8.5.2 Software unit implementation); also in 4-6.5.4, 5-7.5.1, 6-6.5.2, 6-8.5.1)
		6-8.5.2 Software unit implementation	Partly: System Modeling (TB05 (Modular Neural Networks), TB10 (Integration Platform), TB17 (DSL for Models), TB28 (Code Generation), TB31 (Development Update View (incl. Modes), TB43 (Integration of development artifacts into a formal model using different modeling tools); TB441 (Assured Dataset Creation), TB55 (DDI Tool adaptor), TB56 (Development of concepts of automotive SoA specification for innovative functions and learned/learning behaviors); TB58 (Specification of resiliant architectures); TB60 (AS Mode Manager) Partly: System Synthesis (TB 30 (Synthesis of Strategies/Plan); (TB332 (ADORE CD/CI)); TB 54(Optimization of paralellism); TB 59 (Synthesis of adaption strategies) Partly: Monitor Synthesis (TB04 (Code generation for [...] monitors); TB07 (Runtime Metrics for NN); TB442 (Neural Network Architecture with XAI-based ODD Detction); TB57 (Contract based Runtime Monitoring Specification and Analysis).
	9 Software unit verification	**6-9.5.1** Software verification specification	
		6-9.5.2 Software verification	Partly: Simulation (TB 15: Data Drift Injection); TB18 (OEMU Interfaces); TB21 (vECU); TB441 (Co-Simulation); TB61 (Ross-

Abbildung 4.30: Auszug aus der Zuordnung von Technologiebausteinen zur 'Work Product List' des Standards ISO 26262 Funktionale Sicherheit

primären Standards angewandt. Dabei wurde (a) davon ausgegangen, dass die in ADSO entwickelten Technologiebausteine softwaretechnisch weiterentwickelt und für den Einsatz in der Entwicklung und dem Sicherheitsnachweis notwendigen Qualifizierungen erhalten und (b) das Verständnis zugrunde gelegt, dass die Technologiebausteine einen Beitrag, nicht notwendig die vollständige Lösung, zur Erstellung des entsprechenden Work-Products leisten.

Im sogenannten 'Developers View' wurden ausgehend auf der in Arbeitspaket 2.4. vorgenommenen Modellierung der Use-Case spezifischen Instanzen des ADSO Prozesses in den vier Anwendungsfällen, die Technologiebausteine auf diese Prozesse abgebildet, so dass ein Use-Case spzeifisches Bild der dort in den einzelnen Schritten der Entwicklung und des Sicherheitsnachweises notwendigen und genutzten Bausteine entstand. Die entsprechende Prozessmodellierung wurde dabei im Werkzeug TCA (Tool-Chain-Analyzer) des Projektpartners Validas vorgenommen. Die hierzu notwendigen Annahmen zur Toolqualifikation sowie zur Vollständigkeit des durch einen einzelnen Baustein gelieferten Nachweise für den entsprechenden Prozess- bzw. Nachweisschritt sind dabei die gleichen wie im 'Assessors View'. Abbildung 4.31 zeigt ein Beispiel für die in TCA durchgeführte Modellierung.

Zusammenfassend ermöglichen die beiden Sichten ein umfängliches Bild über die Zuordnung der Technologiebausteine zu den verschiedenen in den Regularien und Standards geforderten Nachweisschritten und damit die Analyse, welche dieser Bausteine und Technologien bisher

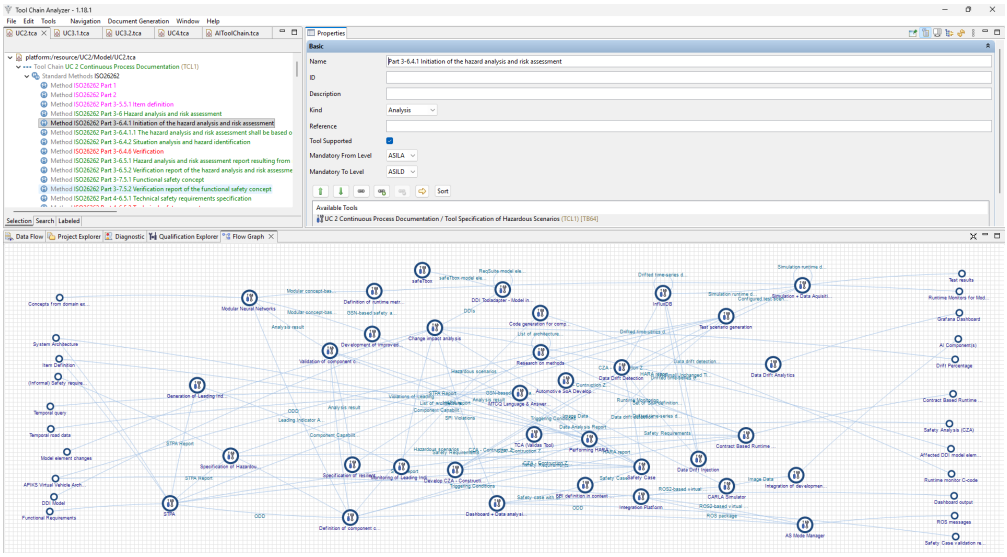


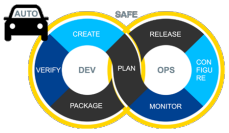
Abbildung 4.31: Auszug aus den in TCA modellierten Flüssen zwischen Methoden/Tools und Artefakten

nicht in den einzelnen Standards abgebildet wurden.

4.11.2 Erweiterungsbedarf der Standards

Die der oben dargestellten Zuordnung von Technologiebausteinen zu durch Prozess- und Nachweisschritten folgende Analyze des Erweiterungsbedarfs von Standards hat insbesondere die folgenden Bereiche identifiziert:

- Eine Sicherheitsargumentation für KI-basierte Komponenten wird aktuell in den Standards wenig berücksichtigt. Insbesondere in den primären Standards ISO 26262 (Funktionale Sicherheit) and ISO 21448 (SOTIF) sind entsprechende Argumente zwar in aktuellen Erweiterungen vorgesehen, ihre Umsetzung in konkrete für die Nachweise verlangte Methoden und Prozesse ist jedoch noch unzureichend. Relevante, jedoch zu erweiternde Standards und Erweiterungen sind:
  - ISO/PAS 8800: Road vehicles — Safety and artificial intelligence. ISO/PAS 8800 ist der erste Standard der die Prinzipien der funktionalen Sicherheitsanalyse direkt auf KI-basierte Lösungen anwendet. Er enthält Richtlinien, Prozesse und Methoden für die Validierung von ML-Modellen, Management der Datenqualität, Monitoringverfahren für KI-Komponenten und Ansätze für die Behandlung von kontinuierlichen Lernverfahren.
  - ISO 26262: Road vehicles — Functional safety, behandelt KI Module als sicherheitsrelevante Komponenten mit den entsprechenden Anforderungen an die zugehörigen Sicherheitsnachweise; entsprechende technische Maßnahmen und Methoden zum Nachweis dieser Eigenschaften fehlen jedoch fast völlig.



  - ISO 21448 (SOTIF): Road vehicles — Safety of the intended functionality, bietet Verfahren zur Minimierung des Vorkommens von unsicheren sowie unbekannten Situationen (Szenarien). Bezüglich KI ist dieses insbesondere für Perzeptionskomponenten relevant. Bezüglich technischer Maßnahmen zur Umsetzung dieser Minimierung sind ebenfalls kaum KI-spezifische bzw. für KI-Systeme notwendige Mechanismen zu finden.

  Die offenen Fragen bezüglich der Behandlung von KI-basierten Komponenten sind dabei insbesondere das nicht-determinisitische – oder besser: das stochastische – Verhalten solcher Komponenten und System, die Qualität der Lerndaten kombiniert mit der offenen Frage der relevanten sicherheitsmetriken für diese Daten, die 'Coverage' der Daten und die Behandlung von Unsicherheiten. Die entsprechenden in ADSO entwickelten Verfahren können hier (Teil-)lösungen bieten, insbesondere die Technologiebausteine für die Kontinuierliche Integration und Updates, Monitoring im Feld, Shadow-mode basiertes Testen und Validieren und frühzeitigem Testen von Updates in virtuellen (Simulations-)Umgebungen. Als neben den obigen Betrachtungen weiter zu vertiefende und zu lösende Fragestellungen empfehlen sich hierbei

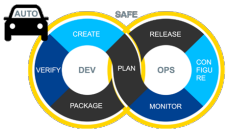
  - Die Erweiterung der HARA (Hazard and Risk Analyis) zur Behandlung von KI-spezifischen Risiken
  - Die Erweiterung der in ISO 26262 beschriebenen Ïtem Definitionfür KI Komponenten.
  - Die vertiefte Nutzung von Safety Performance Indikatoren.

- Standards wie die ISO 345xx Reihe befassen sich bereits mit der definition von ODDs (Operational Design Domains) und dem scenarien-basierten, virtuellen Testen. Es fehlt weiterhin eine ausreichend gute Methodik für das testen von System-Fähikgeiten (Capabilities) gegen diese ODDs – insbesondere die sogenannte Aktuelle OD, Current Operational Domainßur Laufzeit, die entweder in diese Standardreihe ode rin der ISO 21448 (SOTIF) inkludiert werden könnte. Auch hier können die in ADSO (weiter-)entwickelten Technologien eine gute Basis bilden.

- Die Herausforderung, in einem offenen Kontext festzustellen, inwieweit gewählte Testszenarien diesen Konmtext abdecken, ist weiterhin nicht abschließen beantwortet. Die automatisierten, geleitete Generierung von Testszenarien, wie sie in ADSO verwendet und weiterentwickelt wurde, sollte entsprechend vertiefter im SOTIF und/oder der ISO 345xx Reihe behandelt werden, um insgesamt einen Äbdeckungs-Begriff"(Coverage) und den Begriff der äusreichenden Abdeckungfestzulegen.

Als zusätzlicher Erweiterungsbedarf wird empfohlen, eine Integration der primären Sicherheitsstandards ISO 26262 (Funktionale Sicherheit), ISO 21448 (SOTIF) und ISO/SAE 21434 in einem STPA (Systems-Theoretic Process Analysis) basierten Ansatz in ein gemeinsames Sicherheitsframework durchzuführen. Die ersten, hierzu durchgeführten Arbeiten sind vielversprechend, müssen jedoch in Folgeaktivitäten weiter untersucht und analysiert werden.

Als Ausblick auf weitere mögliche Arbeiten empfehlen wir die Behandlung der Fragestellung, inwieweit das Potential von KI-basierten Verfahren nicht nur in den Systemen, sondern



insbesondere auch im Entwicklungs- und Testprozess genutzt werden kann. Das Fehlen entsprechender Standards und Qualifizierungstechniken für solche KI-basierten Entwicklungs- und Testwerkzeuge muss dringend behoben werden, um die Möglichkeiten dieser Technologie in Bezug auf Effizienz- und Funktionalitätssteigerungen auszuschöpfen, gleichzeitig aber die damit entwickelten Systeme den gleichen hohen Sicherheitsanforderungen genügen zu lassen, wie dies heute der Fall ist.

5 Zusammenfassung

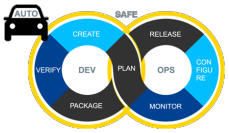
Das Verbundprojekt MANNHEIM–AutoDevSafeOps (ADSO) hat gezeigt, dass sich funktionale Sicherheit und DevOps-Prinzipien erfolgreich in einem konsistenten, industriell anwendbaren Entwicklungs- und Betriebsrahmen für automobile Softwaresysteme vereinen lassen. Damit wurde eine zentrale Herausforderung der künftigen Fahrzeugentwicklung adressiert: die sichere und normkonforme Weiterentwicklung komplexer, KI-basierter Systeme über den gesamten Lebenszyklus hinweg.

Mit der Konzeption und Umsetzung des ADSO-Prozesses wurde ein generischer, sicherheitsorientierter DevOps-Lebenszyklus geschaffen, der die kontinuierliche Integration sicherheitsrelevanter Aktivitäten, Nachweise und Prüfungen in Entwicklung und Betrieb ermöglicht. Der Prozess stellt eine methodische Brücke zwischen den normativen Anforderungen der funktionalen Sicherheit (u. a. ISO 26262) und der Safety of the Intended Functionality (ISO 21448) einerseits sowie den agilen, iterativen Entwicklungspraktiken der modernen Softwaretechnik andererseits dar.

Im Rahmen des Projekts wurden mehr als 80 Technologiebausteine entwickelt, die eine praxisgerechte Umsetzbarkeit des ADSO-Prozesses aufzeigen. Sie adressieren wesentliche Themenfelder wie adaptive und resiliente Softwarearchitekturen, automatisierte Sicherheits- und Compliance-Nachweise, Verfahren zur kontinuierlichen Sicherheitsbewertung, daten- und KI-basiertes Monitoring sowie die Integration von Operational Design Domains (ODD) in formale Sicherheitsverträge. Durch die Kombination dieser Bausteine entsteht eine modulare und skalierbare Grundlage für ein sicheres, automatisiertes und kontinuierliches Software-Deployment im Automobilbereich.

Die Validierung in vier industrienahen Anwendungsfällen hat die Praxistauglichkeit des entwickelten Ansatzes überprüft. Dabei konnte gezeigt werden, dass der ADSO-Prozess die Effizienz und Transparenz sicherheitskritischer Entwicklungsprozesse erhöht, den Aufwand für Sicherheitsnachweise und Zertifizierungsaktivitäten reduziert und die Wiederverwendung vorhandener Sicherheitsartefakte ermöglicht. Gleichzeitig trägt er zur Steigerung der Systemresilienz bei, da unsicheres Verhalten frühzeitig erkannt und durch gezielte Anpassungen korrigiert werden kann. Der zugehörige Erweiterungsbedarf bestehender Standards wurde identifiziert; somit kann darauf hingearbeitet werden, dass diese Verfahren auch aus regulatorischer, normativer Sicht einsatzfähig werden.

Insgesamt liefert MANNHEIM–AutoDevSafeOps einen maßgeblichen Beitrag zur Etablierung sicherer, kontinuierlicher Softwareentwicklungsprozesse in der Automobilindustrie. Der entwickelte Ansatz stellt einen wichtigen Schritt auf dem Weg zu softwaredefinierten, updatefähigen und verlässlich sicheren Fahrzeugen dar und bietet eine tragfähige Grundlage für künftige Forschungs- und Industrievorhaben im Bereich sicherheitskritischer cyber-physischer Systeme.



Zukünftige Arbeiten sollten sich auf die Operationalisierung des ADSO-Prozesses in industriellen Entwicklungsumgebungen konzentrieren. Dabei stehen die Anpassung an bestehende Toolchains, die Automatisierung der Nachweisführung sowie die Integration in unternehmensweite DevOps- und Continuous-Delivery-Pipelines im Vordergrund.

Ein weiterer Schwerpunkt sollte auf der Integration von Künstlicher Intelligenz in sicherheitskritische Systeme liegen. Hierfür müssen Verfahren zur kontinuierlichen Validierung und Zertifizierung lernender Systeme (weiter-) entwickelt werden, um die Vertrauenswürdigkeit von KI-Funktionen über den gesamten Produktlebenszyklus sicherzustellen.

Darüber hinaus bietet die Verknüpfung mit digitalen Zwillingen und datengetriebenem Systems-Engineering großes Potenzial, sicherheitsrelevante Zustände in Echtzeit zu überwachen und prädiktiv zu bewerten. Langfristig kann so ein integriertes Safety-DevOps-Ökosystem entstehen, das den gesamten Lebenszyklus softwaredefinierter Fahrzeuge unterstützt – von der Entwicklung bis zum sicheren Betrieb.

Literaturverzeichnis

[1] Babeltrace. https://babeltrace.org/. 30-06-2025.

[2] chronsuite. https://www.inchron.com/chronsuite/. 30-06-2025.

[3] Irune Agirre, Irune Yarza, Imanol Mugarza, Jacopo Binchi, Peio Onaindia, Tomasso Poggi, Francisco J. Cazorla, Leonidas Kosmidis, Kim Grüttner, Patrick Uven, Mohammed Abuteir, Jan Loewe, Juan M. Orbegozo, and Stefania Botta. Safe and secure software updates on high-performance mixed-criticality systems: The up2date approach. *Microprocessors and Microsystems*, 87:104351, 2021.

[4] Abu Shad Ahammed, Md Shahi Amran Hossain, and Roman Obermaisser. A computer vision approach for autonomous cars to drive safe at construction zone. In *2025 IEEE 6th International Conference on Image Processing, Applications and Systems (IPAS)*, pages 1–6. IEEE, 2025.

[5] Ahmad Alnafessah, Alim Ul Gias, Runan Wang, Lulai Zhu, Giuliano Casale, and Antonio Filieri. Quality-aware devops research: Where do we stand? *IEEE access*, 9:44476–44489, 2021.

[6] Anaheed Ayoub, Jian Chang, Oleg Sokolsky, and Insup Lee. Assessing the overall sufficiency of safety arguments. 2013.

[7] Nasreen Azad and Sami Hyrynsalmi. Devops critical succes factors—a systematic literature review. *Information and Software Technology*, page 107150, 2023.

[8] Claude Baron and Vincent Louis. Towards a continuous certification of safety-critical avionics software. *Computers in Industry*, 125:103382, 2 2021.

[9] Yosab Bebawy, Houssem Guissouma, Sebastian Vander Maelen, Janis Kröger, Georg Hake, Ingo Stierand, Martin Fränzle, Eric Sax, and Axel Hahn. Incremental contract-based verification of software updates for safety-critical cyber-physical systems. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2020.

[10] Jan Steffen Becker, Björn Koopmann, Ingo Stierand, and Lukas Westhofen. Providing evidence for correct and timely functioning of software safety mechanisms. In *Software Engineering 2023 Workshops*, pages 66–77. Gesellschaft für Informatik e.V., Bonn, 2023.

[11] Gerd Behrmann, Agnes Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Uppaal-tiga: Timed games for everyone. In *Nordic Workshop on Programming Theory (NWPT'06)*, 2006.

[12] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto L. Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. Contracts for system design. *Found. Trends Electron. Des. Autom.*, 12(2-3):124–400, 2018.

[13] BMW AG. ALKS scenario interpretation in openscenario, 2025.

[14] Wolfgang Böhm, Manfred Broy, Cornel Klein, Klaus Pohl, Bernhard Rumpe, and Sebastian Schröck. *Model-based engineering of collaborative embedded systems: Extensions of the spes methodology*. Springer Nature, 2021.

[15] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems - Focus on Streams, Interfaces, and Refinement*. Monographs in Computer Science. Springer, 2001.

[16] Frank Budinsky, Ed Merks, Marcelo Paternostro, and Dave Steinberg. *EMF: Eclipse modeling framework*. The eclipse series. Addison-Wesley, Upper Saddle River, NJ [u.a.], 2. ed., rev. and updated. edition, 2011.

[17] Hanbo Cai, Pengcheng Zhang, Hai Dong, Lars Grunske, Shunhui Ji, and Tianhao Yuan. Adversarial example-based test case generation for black-box speech recognition systems. *Softw. Test. Verification Reliab.*, 33(5), 2023.

[18] CARLA. CARLA scenariorunner, 2025.

[19] AutoDevSafeOps Consortium. Deliverable D6.1 'requirements from the use cases'. Technical report, MANNHEIM-AutoDevSafeOps Project, 2022.

[20] AutoDevSafeOps Consortium. Deliverable D2.1 'initial version overall concept and process'. Technical report, MANNHEIM-AutoDevSafeOps Project, 2023.

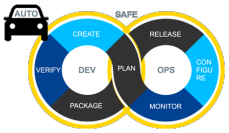
[21] AutoDevSafeOps Consortium. Deliverable D3.1 'initial version of the autodevsafeops foundation'. Technical report, MANNHEIM-AutoDevSafeOps Project, 2023.

[22] AutoDevSafeOps Consortium. Deliverable D4.1 'initial version assurance methods and technologies'. Technical report, MANNHEIM-AutoDevSafeOps Project, 2023.

[23] AutoDevSafeOps Consortium. Deliverable D5.1 'initial version middleware, communication and distributed systems'. Technical report, MANNHEIM-AutoDevSafeOps Project, 2023.

[24] AutoDevSafeOps Consortium. Deliverable D1.2 Final Version – Closing the Gap with respect to standards and regulations. Technical report, MANNHEIM-AutoDevSafeOps Project, 2025.

[25] AutoDevSafeOps Consortium. Deliverable D6.2 'instantiation of the use cases'. Technical report, MANNHEIM-AutoDevSafeOps Project, to appear 2024.



[26] AutoDevSafeOps Consortium. Deliverable D2.2 'final version overall concept and process'. Technical report, MANNHEIM-AutoDevSafeOps Project, to appear 2025.

[27] AutoDevSafeOps Consortium. Deliverable D3.2 'final version of the autodevsafeops foundation'. Technical report, MANNHEIM-AutoDevSafeOps Project, to appear 2025.

[28] AutoDevSafeOps Consortium. Deliverable D4.2 'final version assurance methods and technologies'. Technical report, MANNHEIM-AutoDevSafeOps Project, to appear 2025.

[29] AutoDevSafeOps Consortium. Deliverable D5.2 'final version middleware, communication and distributed systems'. Technical report, MANNHEIM-AutoDevSafeOps Project, to appear 2025.

[30] AutoDevSafeOps Consortium. Deliverable D6.3 'demonstrators for the use cases'. Technical report, MANNHEIM-AutoDevSafeOps Project, to appear 2025.

[31] StepUp!CPS Consortium. Deliverable D2.2 'concept and process for modular updates'. Technical report, StepUp!CPS Project, 2019.

[32] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. An information-theoretic approach to detecting changes in multidimensional data streams. *Interfaces*, 01 2006.

[33] João Paulo Costa de Araujo, Balahari Vignesh Balu, Eik Reichmann, Jessica Kelly, Stefan Kugele, Núria Mata, and Lars Grunske. Applying concept-based models for enhanced safety argumentation. In *ISSRE*, pages 272–283. IEEE, 2024.

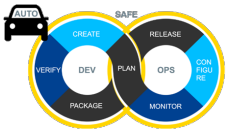
[34] João Paulo Costa de Araujo, Balahari Vignesh Balu, Eik Reichmann, Jessica Kelly, Stefan Kugele, Núria Mata, and Lars Grunske. Applying concept-based models for enhanced safety argumentation. In *35th IEEE International Symposium on Software Reliability Engineering, ISSRE 2024, Tsukuba, Japan, October 28-31, 2024*, pages 272–283. IEEE, 2024.

[35] Gregory Ditzler and Robi Polikar. Hellinger distance based drift detection for nonstationary environments. In *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*, pages 41–48, 2011.

[36] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[37] Paul M. Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Signature Series. Addison-Wesley, Upper Saddle River, N.J., 2007.

[38] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 1 2017.



[39] John B Goodenough, Charles B Weinstock, and Ari Z Klein. Eliminative argumentation: A basis for arguing confidence in system properties. *Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2015-TR-005*, 2015.

[40] Daniel Grimm, Moritz Zink, Marc Schindewolf, and Eric Sax. Adaptive cybersecurity monitoring for resilient vehicular architectures. In *2023 IEEE Vehicular Networking Conference (VNC)*, pages 41–48, 2023.

[41] Daniel Grimm, Moritz Zink, Marc Schindewolf, and Eric Sax. Cyber situational awareness in vehicle security operations: Holistic monitoring and a data model. In *2024 20th International Conference on Network and Service Management (CNSM)*, pages 1–7, 2024.

[42] Houssem Guissouma, Moritz Zink, and Eric Sax. Continuous safety assessment of updated supervised learning models in shadow mode. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 301–308, 2023.

[43] Magnus Gyllenhammar, Gabriel Rodrigues de Campos, and Martin Törngren. Holistic perspectives on safety of automated driving systems-methods for provision of evidence. *Authorea Preprints*, 2023.

[44] Md Shahi Amran Hossain, Abu Shad Ahammed, Divya Prakash Biswas, and Roman Obermaisser. Impact analysis of data drift towards the development of safety-critical automotive system. In *2024 International Symposium ELMAR*, pages 325–330. IEEE, 2024.

[45] Jez Humble and David G. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. A Martin Fowler Signature Book. Addison-Wesley, Upper Saddle River, NJ, 7th printing 2013 edition, 2011.

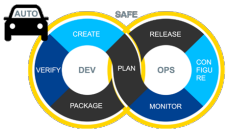
[46] Yassir Idmessaoud, Didier Dubois, and Jérémie Guiochet. Uncertainty elicitation and propagation in gsn models of assurance cases. In *SAFECOMP 2022*, pages 111–125. Springer, 2022.

[47] International Organization for Standardization. ISO 26262: Road vehicles- Functional Safety. Standard, 2018.

[48] International Organization for Standardization. Road vehicles — Safety and cybersecurity for automated driving systems – Design, verification and validation (ISO/TR 4804). Technical report, ISO, 2020.

[49] International Organization for Standardization. ISO 21448: Road vehicles – Safety of the intended functionality. Standard, 2022.

[50] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is devops? a systematic mapping study on definitions and practices. In *Proceedings of the scientific workshop proceedings of XP2016*, pages 1–11, 2016.



[51] Jong-Chan Kim Jaesung Jang, Hyeongyu Lee. Carfree: Hassle-free object detection dataset generation using carla autonomous driving simulator. *Applied Sciences*, 12(1):281, Dec 2021.

[52] Shunhui Ji, Changrong Huang, Bin Ren, Hai Dong, Lars Grunske, Yan Xiao, and Pengcheng Zhang. Taefuzz: Automatic fuzzing for image-based deep learning systems via transferable adversarial examples. *ACM Trans. Softw. Eng. Methodol.*, January 2025. Just Accepted.

[53] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. *CoRR*, abs/2007.04612, 2020.

[54] Dimitrios Kolovos, Louis Rose, Richard Paige, and A Garcia-Dominguez. *The Epsilon Book*. Eclipse, 2010.

[55] Philip Koopman. *How Safe Is Safe Enough?: Measuring and Predicting Autonomous Vehicle Safety*. 3 edition.

[56] Janis Kröger and Martin Fränzle. Mode management in contract-based design. In *SE 2024-Companion*, pages 31–42. Gesellschaft für Informatik eV, 2024.

[57] Janis Kröger, Björn Koopmann, Ingo Stierand, and Martin Fränzle. Contract-based specification of mode-dependent timing behavior. *Innovations in Systems and Software Engineering*, pages 1–17, 2023.

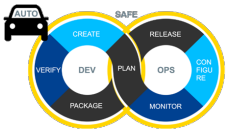
[58] Janis Kröger, Ingo Stierand, and Martin Fränzle. Ensuring integration conditions during the update of cyber-physical systems. In *Formal Methods for Industrial Critical Systems: 30th International Conference, FMICS 2025, Aarhus, Denmark, August 27–28, 2025, Proceedings*, page 203. Springer Nature, 2025.

[59] Janis Kröger and Martin Fränzle. Updates at runtime for cyber physical systems. a game theoretic approach. In *Software Engineering 2023 Workshops*, pages 54–65. Gesellschaft für Informatik e.V., Bonn, 2023.

[60] Stefan Kugele, Philipp Obergfell, Manfred Broy, Oliver Creighton, Matthias Traub, and Wolfgang Hopfensitz. On service-orientation for automotive software. In *2017 IEEE International Conference on Software Architecture, ICSA 2017, Gothenburg, Sweden, April 3-7, 2017*, pages 193–202. IEEE Computer Society, 2017.

[61] Nishanth Laxman, Joel Varghese Jiby, Ioannis Sorokos, Joshua Frey, and Jan Reich. From uncertainty representation to safety performance monitoring for operational safety assurance-a systematic approach. 2025.

[62] Nishanth Laxman, Markus Schweizer, and Jan Reich. Employing field monitoring and runtime safety verification for cumulative operational safety assurance in a safetydevops process. In *2024 8th International Conference on System Reliability and Safety (ICSRS)*, pages 545–549. IEEE, 2024.



[63] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.

[64] Nancy Leveson. A systems approach to risk management through leading safety indicators. *Reliability Engineering & System Safety*, 136:17–34, 2015.

[65] Nancy Leveson. *Engineering a safer world*. MIT Press, 2017.

[66] Nancy G Leveson and John P Thomas. Stpa handbook. *Cambridge, MA, USA*, 2018.

[67] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1):18, 2021.

[68] Patrick Lindstrom, Brian Mac Namee, and Sarah Jane Delany. Drift detection using uncertainty distribution divergence. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 604–608, 2011.

[69] Anjin Liu, Jie Lu, Feng Liu, and Guangquan Zhang. Accumulating regional density dissimilarity for concept drift detection in data streams. *Pattern Recognition*, 76:256–272, 2018.

[70] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. Adopting devops in the real world: A theory, a model, and a case study. *Journal of Systems and Software*, 157:110384, 2019.

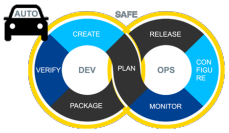
[71] Lucy Ellen Lwakatare, Terhi Kilamo, Teemu Karvonen, Tanja Sauvola, Ville Heikkilä, Juha Itkonen, Pasi Kuvaja, Tommi Mikkonen, Markku Oivo, and Casper Lassenius. Devops in practice: A multiple case study of five companies. *Information and Software Technology*, 114:217–230, 2019.

[72] Bart Meyers, Klaas Gadeyne, Bentley Oakes, Matthias Bernaerts, Hans Vangheluwe, and Joachim Denil. A model-driven engineering framework to support the functional safety process. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 9 2019.

[73] Peter Munk and Markus Schweizer. DevOps and safety? SafeOps! towards ensuring safety in feature-driven development with frequent releases. In *Lecture Notes in Computer Science*, pages 145–157. Springer International Publishing, 2022.

[74] Ann-Therese Tabea Nägele and Eric Sax. Smart testing for smart vehicles: Conditional regression strategies for software updates. In *2025 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*, 2025. accepted for publication.

[75] Ann-Therese Tabea Nägele, Marc Schindewolf, and Eric Sax. Collaborative continuous testing of automotive services (coco test). In *2025 IEEE International Systems Conference (SysCon)*, pages 1–8, Montreal, QC, Canada, April 2025. IEEE.



[76] Fazli Faruk Okumus, João-Vitor Zacchi, Maike Salfeld, Markus Schweizer, Núria Mata, and Stefan Kugele. Runtime monitor synthesis for automotive software architectures. In *Software Architecture - 19th European Conference, ECSA 2025*, LNCS. Springer, 2025. to appear.

[77] Abdulhakim A. Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. KDD '15, page 935–944, New York, NY, USA, 2015. Association for Computing Machinery.

[78] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Martins Mozeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. LGSVL simulator: A high fidelity simulator for autonomous driving. pages 1–6, 2020.

[79] Thomas Rosenstatter, Kim Strandberg, Rodi Jolak, Riccardo Scandariato, and Tomas Olovsson. REMIND: A framework for the resilient design of automotive systems. In *IEEE Secure Development, SecDev 2020, Atlanta, GA, USA, September 28-30, 2020*, pages 81–95. IEEE, 2020.

[80] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.*, 1(5):206–215, 2019.

[81] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *CoRR*, abs/2103.11251, 2021.

[82] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: A multi-class classification competition. In *IJCNN*, pages 1453–1460. IEEE, 2011.

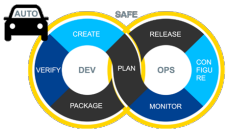
[83] The British Standards Institution. PAS 1881:2022 Assuring the operational safety of automated vehicles – Specifi cation. Specification, 2022.

[84] Underwriter Laboratories. ANSI/UL 4600 - Standard for Evaluation of Autonomous Products. Standard, 2022.

[85] United Nations. Uniform provisions concerning the approval of vehicles with regard to automated lane keeping systems. Technical Report E/ECE/TRANS/505/Rev.3/Add.156, United Nations, New York, 2021.

[86] Sebastian Vost and Stefan Wagner. Keeping continuous deliveries safe. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 5 2017.

[87] Ran Wei, Tim P. Kelly, Xiaotian Dai, Shuai Zhao, and Richard Hawkins. Model based system assurance using the structured assurance case metamodel. *Journal of Systems and Software*, 154:211–233, 2019.



[88] Lukas Westhofen, Jean Christoph Jung, and Daniel Neider. Temporal conjunctive query answering via rewriting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(14):15221–15229, Apr. 2025.

[89] Lukas Westhofen, Christian Neurohr, Jean Christoph Jung, and Daniel Neider. Answering temporal conjunctive queries over description logic ontologies for situation recognition in complex operational domains. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–187, Cham, 2024. Springer Nature Switzerland.

[90] Lukas Westhofen, Christian Neurohr, Jean Christoph Jung, and Daniel Neider. Topllet: An optimized engine for answering metric temporal conjunctive queries. In Nathaniel Benz, Divya Gopinath, and Nija Shi, editors, *NASA Formal Methods*, pages 314–321, Cham, 2024. Springer Nature Switzerland.

[91] Danny Weyns. *An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective*. 02 2021.

[92] João-Vitor Zacchi, Edoardo Clementi, and Núria Mata. Apiks: A modular ros2 framework for rapid prototyping and validation of automated driving systems. *arXiv preprint arXiv:2502.20507*, 2025.

[93] Marc Zeller. Towards continuous safety assessment in context of DevOps. In *Lecture Notes in Computer Science*, pages 145–157. Springer International Publishing, 2021.

[94] Marc Zeller, Ioannis Sorokos, Jan Reich, Rasmus Adler, and Daniel Schneider. Open dependability exchange metamodel: A format to exchange safety information. In *2023 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–7, 2023.

[95] Moritz Zink, Daniel Grimm, and Eric Sax. Aurora networks: Auto-associative universal real-time outlier risk assessment networks. In Martin Törngren, Barbara Gallina, Erwin Schoitsch, Elena Troubitsyna, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security. SAFECOMP 2025 Workshops*, pages 499–510, Cham, 2026. Springer Nature Switzerland.

[96] Moritz Zink, Luca Seidel, Daniel Baumann, Daniel Grimm, and Eric Sax. Artemis - adaptive response and tracking of external model inputs in safety-critical systems. 2025. Accepted at 2025 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE).