A Changing Landscape

A Changing Landscape: On Safety & Open Source in Automated and Connected Driving

Sangeeth Kochanthara

A Changing Landscape: On Safety & Open Source in Automated and Connected Driving

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens,
voor een commissie aangewezen door het College voor Promoties,
in het openbaar te verdedigen op vrijdag 17 maart 2023 om 13:30 uur

door

Sangeeth Kochanthara

geboren te Vattamkulam, Kerala, India

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. E.R. van den Heuvel
promotor:	prof.dr. M.G.J. van den Brand
copromotoren:	dr. Y. Dajsuren EngD, dr.ir. L.G.W.A. Cleophas
leden:	Prof.Dr.Rer.Nat.Habil. A. Wortmann (Universität Stuttgart)
	dr. E. Poll (Radboud Universiteit)
	prof.dr. H. Nijmeijer
	prof.dr. M. Staron (Chalmers University)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

A Changing Landscape: On Safety & Open Source in Automated and Connected Driving

Sangeeth Kochanthara

The work in the thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics) and was financed by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), i-CAVE project (14897 P14-18).
IPA dissertation series 2023-01

Printed by: ADC Nederland

Cover: Designed using DALL-E2 by Ayushi Rastogi & Sangeeth Kochanthara
Finishing touches and layout by Jinan Sekhar Edathara `www.jinansekhar.com`

A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-5673-1

An electronic version of this dissertation is available at `https://research.tue.nl/`

For my parents
— *Sathidevi* and *Chandran*

When we had no computers, we had no programming problem either.
When we had a few computers, we had a mild programming problem.
Confronted with machines a million times as powerful,
we are faced with a gigantic programming problem.

Edsger Dijkstra

Contents

Summary

Automobiles are the backbone of modern civilization. We depend on automobiles directly for daily transport and indirectly for everything we use, including goods and services. Any progress in the automotive industry can potentially improve the lives of almost all humans.

The twenty-first century automotive industry has become a software-intensive industry. The industry landscape is changing with the paradigm shift to automated and connected driving, predominantly enabled by software. This shift to automated driving, where a vehicle drives itself, and connected driving, where a set of vehicles exchange information and adapt their driving strategy for collective traffic optimizations, releases humans partially or entirely from the driving task. The dependence on software for the driving task makes the software that enables these technologies safety critical. This thesis focuses on the changes in the safety landscape due to automated and connected driving and the landscape shift to open source automotive software development.

Safety in automotive starts from safety requirements. Over the years, many processes and techniques to elicit safety requirements have been proposed. Any systematic and informed safety requirement elicitation, especially in the automated and connected driving context, requires consolidation and synthesis of the knowledge of these processes and techniques. The thesis presents a review of the state-of-the-art process and techniques for eliciting safety requirements in the automotive domain, compares the processes, and presents taxonomies of the techniques. The systematic literature review of 102 primary studies found that despite the vast literature, there is a need for real-world case studies for automated driving and safety requirement elicitation processes considering emergent behaviors in connected driving.

The thesis introduces a new process to address the lack of a safety requirement elicitation process for connected driving systems. The new process views the connected driving system as a system of systems to identify the emergent behaviors (and hence the additional safety requirements) visible only in the system of a systems view. Our case study on a connected driving prototype developed at Eindhoven University of Technology (TU/e) demonstrates the feasibility and ability of the new process to capture safety requirements on collective behaviors that are invisible at an individual system level.

Safety requirements are as good as their usage in the various product life cycle stages from design to deployment. This thesis presents how to use safety requirements in the design stage (of software and system) in connected driving systems. While design analysis methods have matured and evolved over the years, their applicability for quality

attribute safety in the automotive context was not studied in the literature. We extend the most mature software design assessment method, the Architecture Trade-off Analysis Method (ATAM), for system and software safety assessment in the automotive context. The proposed method describes how to use different abstractions of the software and its architecture effectively to analyze and assess the design against the safety requirements. A proof of concept implementation of the proposed method is presented on the connected driving prototype from TU/e. The designers of the prototype validated the results and usability of the method.

Safety considerations in the design of the perception system are crucial in bringing automated driving to actual roads. The perception system software, comprising traditional and machine learning components, is the part of automated driving software that is responsible for understanding a vehicle's environment and the relative position of surrounding objects. The safety aspects of perception systems within the context of completely automated driving are underexplored. The thesis presents the first study on the safety assessment of the design of the perception system of a mature automated driving software from the industry, Apollo, for its use on Dutch highways. The study showed that while all requirements relating to traditional software are fulfilled, most requirements specific to machine learning-based components are not.

The half-a-century-old landscape of automotive software and its development is shifting from proprietary to open source. We offer a first glimpse into the automotive software landscape in open source via an exploratory study through mining automotive software repositories in GitHub. The study shows high participation from organizations and software companies like Baidu, Microsoft, and start-ups leading the open-sourcing path for automotive software development. The thesis characterizes open source automotive software and its development style and potential implications across different dimensions, including educators, new entrants, and stakeholders like car makers, safety certification bodies, and vehicle users.

This research emphasizes the need for safety in automated and connected driving. The processes, techniques, and methods discussed in this thesis are steps toward better and safer automotive software, while the observations on the open source landscape can contribute to better development of such software.

Acknowledgments

My Ph.D. journey began with an interview with Mark van den Brand and Yanja Dajsuren. Loek Cleophas joined the supervising team immediately after I started at TU/e. First of all, I thank my promotor, Mark. I admire how you made me see the big picture, found creative ways to pivot ideas, and facilitated whatever I needed from day one and throughout my Ph.D. journey. I am often surprised by how you always find time for me from your busy schedules, sometimes at short notice. Thanks to my co-supervisor, Loek. This thesis is built on the many discussions, and the sheer amount of time you invested in me beyond the usual working hours. I can't thank you enough for your constant support and ability to keep a realistic perspective on my (often) unrealistic deadlines. Thanks to my co-supervisor Yanja, without whom the Ph.D. work package itself would not exist. You were always available whenever I needed and your knowledge in automotive, architecture and soft-skills have been of tremendous help to me. The freedom, comfort, guidance, and support for my ambitions that you three gave me made this journey enjoyable.

I thank the manuscript committee: Andreas Wortmann, Erik Poll, Henk Nijmeijer, and Miroslaw Staron, for reviewing the manuscript and for feedback. Your constructive comments have made this thesis better.

In the early stages of my Ph.D., I was mentored by Reinder Bril, Venkatesh Vinayakarao, and Jeroen Keiren. Thanks to Reinder for the discussions and pointers during the first months of my Ph.D., which informed early decisions. Thanks to Venkatesh for continuous encouragement and initial discussions on formal work (which is not a part of this thesis). Our discussions shaped my writing (and initial paper structuring) process. Thanks to Jeroen for coming in at a crucial stage in my Ph.D. I have learned most of my knowledge of systematic thinking, formulation, and presentation of mathematical proofs from you.

Thanks to my collaborators and colleagues from the industry, Arash Khabbaz Saberi (also a former Ph.D. in the SET group) and Alexandru Forrai. It was a pleasure working with you, and the success of these projects forms part of this thesis.

I want to thank the master's students I supervised: Tajinder Singh and Niels Rood, whose persistence and motivation extended beyond their master's thesis, and their work has contributed to this dissertation. I also want to acknowledge the bachelor students I mentored: Thijs Wester, Jari Martens, Sabine Havermans, Tim van Lankveld, Kevin Malkow, Priyatam Sai, and Kumar Tej, who (indirectly) contributed to the connected driving prototype at TU/e.

Thanks to Magiel Bruntink, Alex Serban, Tom van der Sande, Frans Hoogeboom, and Robbin van Hoek for being part of the team, the discussions, and creating the connected driving demonstrator which was used in this thesis.

Thanks to my master supervisors Rahul Purandare, Geoffrey Nelissen, and David Periera. Without them, I would not have entered the research track in my career in the first place. The amount of feedback and mentoring I received formed the basis of all decisions I took during my Ph.D. journey. Special thanks to Geoffrey for his valuable advice beyond the Ph.D. and the interesting conversations.

To the amazing colleagues who made my time in Eindhoven fun, colorful, and enjoyable. Special thanks to Rick Erkens; I always enjoyed the coffee chats, your cooking, teaching me whatever I know about coffee brewing, and helping with the teaching. Thanks to Lina Ochoa Venegas, Nan Yang, Mauricio Verano Merino, Daniela Girardi, Felipe Ebert, Miguel Botto Tobar, Tukaram Muske, and Kousar Aslam for being so kind to me. I will always remember our tons of chats, dinners, walks, and making me taste authentic cuisines. Thanks to Thomas Neele for your special pancakes; you were my go-to person to navigate the Dutch system. Thanks to Maurice Laveaux, you were my go-to person for my questions on typesetting the thesis and many Dutch translations.

I also thank current and former Ph.D. candidates and Postdocs of SET, FSA, and security groups, including Önder Babur, Wessley Silva Torres, Nathan Cassee, David Manrique Negrin, Mahdi Saeedi Nikoo, Hossain Muhammad Muctadir, Priyanka Karkhanis, and Sowmya Ravidas. It was a pleasure sharing office with some of you, attending multiple official and casual events together, SubSET meetings. You always made me feel home at Eindhoven.

Thanks to Gökhan Kahraman, Mohamoud Talebi, Mark Bouwman, Rodin Aarssen, Josh Mengerink, Jouke Stoel, Omar Alzuhaibi, Dana Zhang, Yaping Luo, Sander de Putter, Alexandar Fedorov, Ana-Maria Sutii, Ulyana Tikhonova, Muhammad Osama, Olav Bunte, Anna Stramaglia, Tom Franken, Ferry Timmers, Fei Yang, Jan Martens, Flip van Spaendonck, Lars van den Haak, and Satrio Rukmono for the fun interactions at SET and FSA colloquiums and Ph.D. meetings. I also thank colleagues from the PhD-PDEng council with whom I co-organized several events, including a marathon and Christmas gifts.

Thank you Bas Luttik, Ion Barosan, Tim Willemse, and Erik de Vink for the many corridor chats. Assisting in the teaching of Bas's Logic and Set Theory course taught me a new way of thinking and making proofs.

Thanks to the group secretaries Margje Mommers-Lenders, Agnes van den Reek, and Désirée van Oorschot for helping with organizational tasks.

Alexander Serebrenik has been very kind to me and always offered a new perspective from an empirical and statistical standpoint. Thanks to Eleni Constantinou, our chats and the road trip in Cyprus, and your amazing company; you are one of the super cool people I have ever met. Thanks to Gunnar Kudrjavets, your special way of wittiness while at the same time being profound and how you care for others have been something I cannot wrap

my head around yet. Tijs van der Storm, although we had fewer conversations, I really enjoyed them. Thanks to Sampi and Joseph, our night outs and chats on job search was comforting.

Thanks to the dear ones Rifat, Gitika, Subin, Akhisha, Vineesh, Devika, Kiran, Shilpa, Amal, Vishnu, Arunitha, Subash, Babu, Shaneesh, Rishija, Vipin, Alseena, and Shafeeq. With all the trips, countless dinner parties, badminton, card games, and help I received in difficult times, I know I can always lean on you folks. When I arrived in Eindhoven for the first time without a place to stay, Subin offered to share his place until I got a place. Rifat, you have been so supportive and encouraging and our get-togethers, and regular chats, helped me keep sane throughout the Ph.D. journey. It was an honor to be your paranymph and best man. Kiran, you have been a great friend, and I have learned a lot from your suggestions and view of life. I admire how Vishnu handles the worst of situations with a laugh. Vineesh, I don't understand how you manage to be so calm and carry yourself without worrying about anything. The tons of advice from Shaneesh and Babu and helping hands at crucial moments at short notice by Alseena and Shafeeq always made me feel at home in the Netherlands. I thank all my friends whose names I could not mention explicitly. You all have been a source of constant support and encouragement and I know that you always will be!

What I missed most in the Netherlands was my family. The two people who changed it are my *"Dutch Parents"* Thea and Cris. Any form of thanks would not be enough to convey my gratitude and admiration for both of you and the way you both supported me, showing me a different side of life, love, and patience. Thanks to Corry, Ronald, Marjan, and Jaap Jan for the tea parties and for bearing with us and our super broken Dutch.

None of this could have been possible without the constant support, trust, faith, and everything that Amma, Achan, and Kavitha have done for me. I would have been doing some mundane job and would not have landed in the Netherlands itself. You all have encouraged and supported me so much throughout my life that no amount of payback will ever be enough. I am grateful for your support with my choices, many of which I know are against your intuitions, and your unconditional love is irreplaceable in the entire world. Thanks to my in-laws (Mummy and Papa) and Didi, who are always supportive and accommodating, gave me a helping hand whenever I needed it; and went beyond their belief system for me. I hope that I will be able to make you all proud.

Last but not least, to my better half Ayushi, I don't know where to start! You stood with me during happy and challenging times, instilling confidence, encouraging me, supporting my ambitions, and making me stable. If it were not for you acting as a supporting pillar beyond what I could ask for, I would not have survived the Ph.D. journey. A gazillion thanks will not be enough.

Sangeeth
Assen, January 2023

1

Introduction

Automobiles are an inevitable part of modern human civilization. An estimated 1.4 Billion vehicles were on roads around the globe in 2019.[1] A cumulative 71 billion human hours are spent on roads in a year in the United States alone.[2] Thus, any advancement in the automotive domain can potentially improve the lives of almost every human being.

The automotive industry, worth 3 trillion dollars[3] with 66 million+ cars sold every year in the past decade,[4] has transitioned from an electro-mechanical to a software intensive one. In 2020, the software in a car and hardware it runs on is estimated to cost from US$4,800 up to US$10,650.[5] By 2030, this cost is expected to double forming an estimated 50% of the total car cost, as shown in Figure 1.1.[6]

Software-centered business models are emerging and finding success for the first time in the automotive industry. An example is Tesla Motors, which provides its advanced driver-assistance software (called Autopilot) as a service and separately bills based on the features opted by the user, above the cost for the car itself (such services may contribute up to 20% of the total cost). In the 2020s, the success of a vehicle in the market depends more on innovations in software than on the mechanical side, with 90% of all upcoming innovations in the industry expected from software and its engineering [1]. This thesis focuses on automotive software and its engineering.

The history of automotive software dates back to 1978 with the introduction of the first-ever electronic control unit (special purpose computer) in a car by General Motors [2].

[1] https://drivetribe.com/p/how-many-cars-are-there-in-the-dqbpAzrATLOOSgDfRrgkjQ?iid=B8Prt7paR3G7Rf82CnLLTA

[2] https://wtop.com/dc-transit/2019/02/highway-stars-survey-shows-americans-spent-71-billion-hours-on-the-open-roads-in-a-year/

[3] https://www.washingtonpost.com/business/cars-are-suddenly-worth-3-trillion-and-its-not-all-tesla/2021/12/29/0d9deaf2-68a8-11ec-9390-eae241f4c8b1_story.html

[4] https://www.statista.com/statistics/200002/international-car-sales-since-1990/

[5] https://www.eetimes.com/projections-for-rising-auto-software-cost-for-carmakers/

[6] https://www.statista.com/statistics/277931/automotive-electronics-cost-as-a-share-of-total-car-cost-worldwide/

1

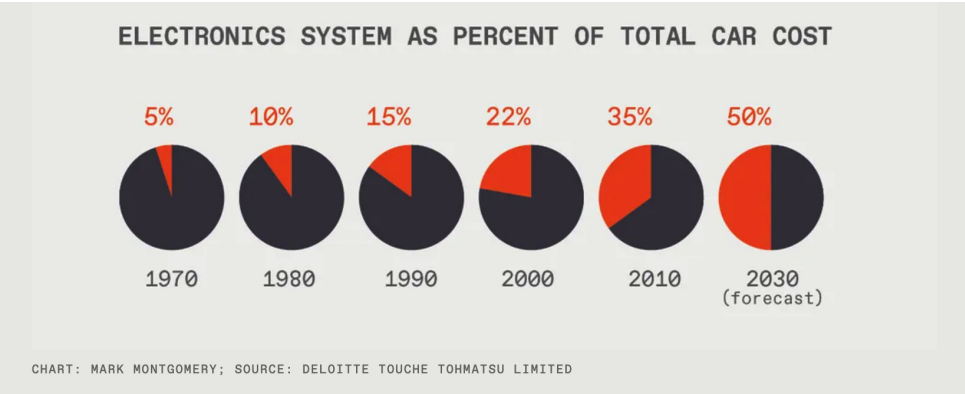


Figure 1.1: Cost of electronic systems in a vehicle. Figure credits: "https://spectrum.ieee.org/software-eating-car"

Fast forwarding to the 2020s, high-end cars like those in the BMW 7 series might contain 150 electronic control units [1] (see an indicative figure in Figure 1.2). A pickup truck like the Ford F-150 runs on 150 million lines of code[7]. Even lower end vehicles are nearing 100 million lines of code. With such code volumes, today's cars surpass modern airplanes, the Large Hadron Collider, the Android OS, and Facebook's front-end software in code size by a considerable margin.[8]

The industry is undergoing arguably the most significant paradigm shift since its inception with the advent of automated and connected driving. In automated driving, the vehicle takes over part of or the complete driving task from humans. In connected driving, the collective traffic behavior of a set of vehicles is optimized by communication among the vehicles and smart traffic infrastructure. In this context, this dissertation explores two primary directions:

1. the safety of automated and connected driving software, further detailed in Section 1.1;

2. a new landmark shift in automotive software development with the move to open source, elaborated in Section 1.2.

[7] https://cmte.ieee.org/futuredirections/2016/01/13/guess-what-requires-150-million-lines-of-code/

[8] Note that a higher code volume does not necessarily mean higher software complexity (the most advanced fighter jet–F35–has 35 million lines of code, but is arguably more complex). Some reasons for the magnitude of code volumes in automobiles might be: (a) less strict quality requirements leading to unreachable code; (b) the current style of automotive software development where the car maker does not have access to the source code of individual ECUs leading to sub-optimal code; (c) thin profit margins, the higher speed to market and the shorter cycle time of software features [3].

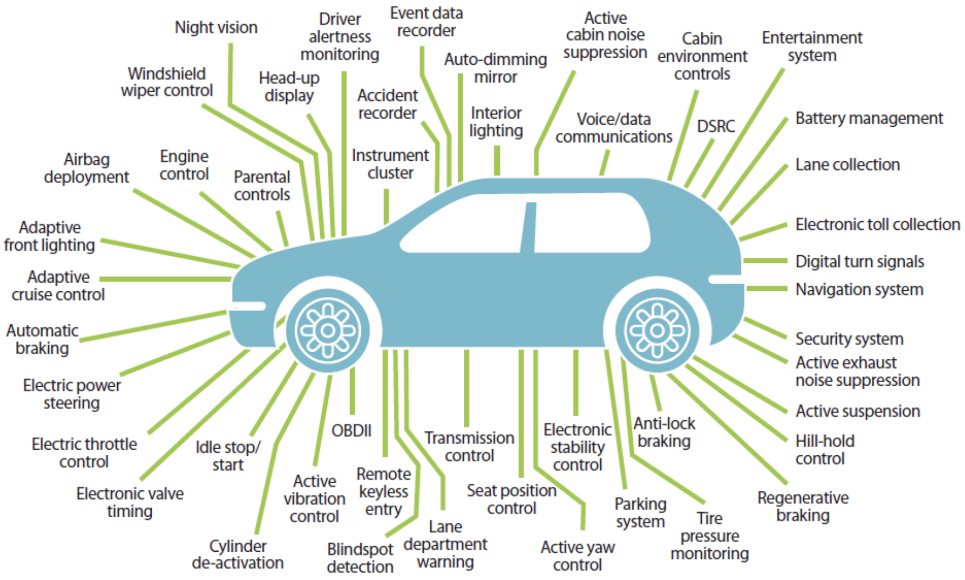


Figure 1.2: Electronic control units (special purpose computers) and their function. Figure credits: [4]

1.1 Safety of automated and connected driving software

Automobiles are safety-critical systems, meaning that a failure or malfunction may result in death or injury to users and other traffic participants and can damage infrastructure. For instance, software not functioning as intended led to the death of a pedestrian in the infamous Uber self-driving car crash.[9] Defects and failure of automotive parts are estimated to be the reason for 120,000 accidents every year.[10]

In the 2020s, one of the primary reasons for vehicle safety-related issues is the failure or malfunction of software and hardware that runs it. Recalls exemplify this trend. Recalls are requests from a manufacturer to return a product, typically after discovering safety issues. 2018 was a record-setting year when defects related to software and electronics

[9] https://www.ntsb.gov/news/press-releases/Pages/NR20191119c.aspx
Note that the investigation by the National Transportation Safety Board of the United States concluded that the distracted emergency driver did not take over the driving task and such a take over could have avoided the incident. Uber or their autonomous driving system was not charged. However, the automated driving stack could not accomplish its task of either stopping or maneuvering the vehicle to avoid the crash.

[10] https://www.peakefowler.com/vehicle-accidents-caused-by-defective-auto-parts/

that run the software formed the reason for the highest proportion ever of all reasons for recalls in the United States (17.8 million out of 47.2 million total recalled vehicles). [11]

Every year since 2018 software related defects formed one of the top 3 reasons in the United States for recall.[12] Just in the ten months July 2021–April 2022, 390+ crash incidents (leading to 6 deaths and 5 serious injuries) were officially reported to be linked to advanced driver assistance systems and their software.[13]

Software related safety issues can be caused by multiple factors. Some factors are:

- System or sub-system failures [5];
- Performance limitations (e.g., due to weather) or insufficient situational awareness [6];
- Insufficiencies of specification [6] and deficiencies in specified driving behavior [7];
- Incorrect and inadequate human-machine interface design (leading to inappropriate user situational awareness, e.g., user confusion, user overload, user inattentiveness) [6, 8]; and
- Attack exploiting vehicle security vulnerabilities [9, 10].

This thesis focuses on the first two types of safety issues. Each of the above factors can be considered in different stages of the product life cycle including requirements, design, development, validation & verification, and deployment. This thesis considers the first two stages: requirements and design.

1.2 Automotive software development: A landmark shift

In its 40+-year-old history, automotive software has been primarily developed in closed source. Closed source software, also known as proprietary software, in the context of this thesis, refers to software where the source code of the software is not made available publicly or not developed publicly. Thus, the access to automotive software to academics, educators, beginners, and even within a company was limited. This thesis shows that for the first time in history, automotive software, at scale, across the industry, is starting to be developed in open source.

With software being at the center stage of the automotive industry, the move to open source can revolutionize the industry and has potential implications across the board, including on vehicle users, educators, researchers, practitioners, companies, and safety certification bodies.

[11] https://www.recallmasters.com/2018-recalls/

[12] Based on the recall data from https://www.recallmasters.com

[13] https://www.nytimes.com/2022/06/15/business/self-driving-car-nhtsa-crash-data.html

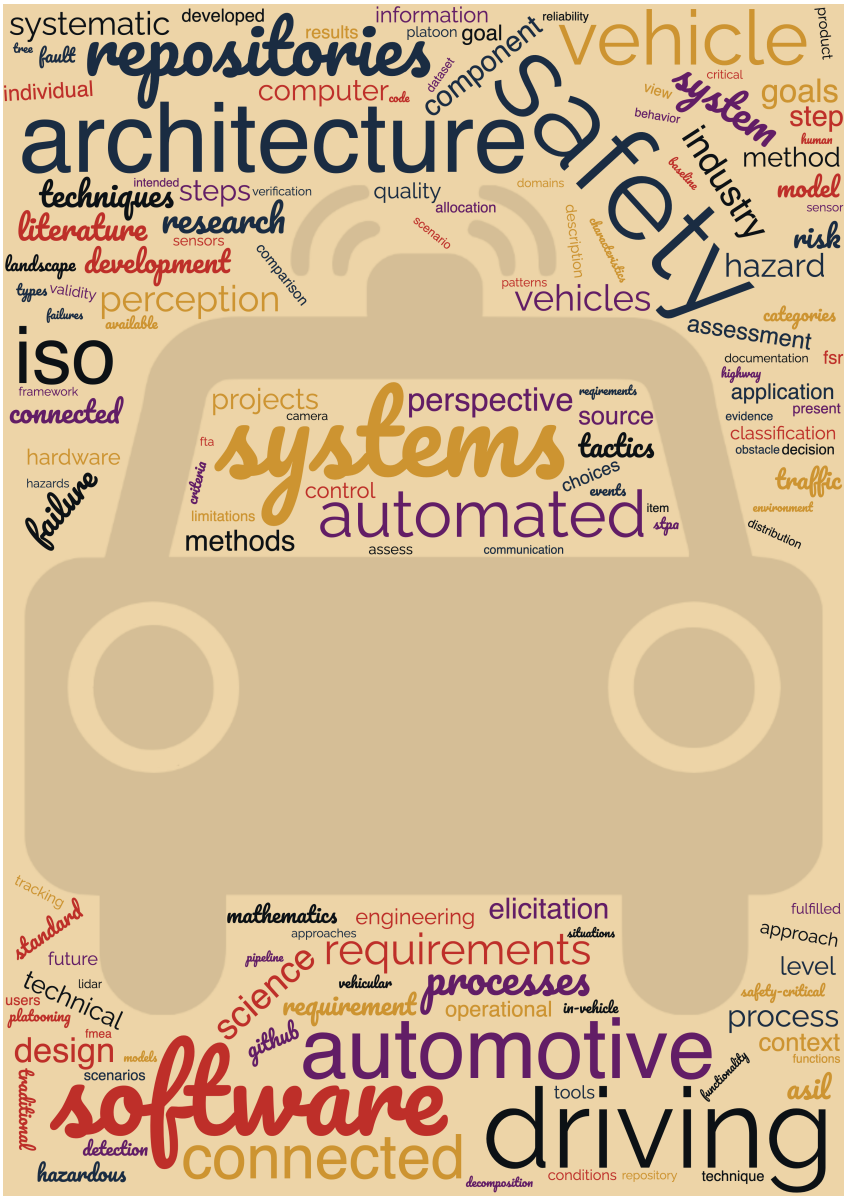


Figure 1.3: A word cloud formed from the content of this thesis (Source: wordclouds.com)

1

From a vehicle user and educator perspective, this is the first time they will be able to have broad access to the source code of software that goes inside future vehicles as well as free access to the tools that build them. From researchers' perspective, for the first time, they can compare the software and its development across companies with the detailed development data available via open-source development platforms. Before open-sourcing, practitioners have been in the knowledge silos of their own company/team with little knowledge of how the rest of the teams and companies develop their software. Open-sourcing has the potential to democratize this landscape. For companies, including car makers (otherwise known as original equipment manufacturers), their suppliers, and tool vendors, open-sourcing provides the potential to attract talent, gain traction for their product, as well as elicit community participation. For safety certification bodies, this open-sourcing trend acts as a new challenge and an opportunity to ensure software, especially safety-critical software, is developed appropriately.

With such broad implications across many dimensions, studying open-source automotive software is important for the industry as a whole and every stakeholder involved. This thesis provides a first glimpse of automotive software and its development in open source.

A word cloud formed from the content of this thesis that succinctly represents the main topics is presented in Figure 1.3.

1.3 Contributions and chapter organization

This thesis is primarily centered around two topics: *safety of automotive software (functional safety and safety of intended functionality)* and *automotive software in open source.* The contributions in this thesis are presented by answering seven research questions (RQ 1.1 to RQ 4), grouped into four categories. The first three categories (six research questions) address the *safety* side, while the last category (remaining research question) addresses *automotive software in open source.* The research questions and the resulting contributions are described below.

> ***RQ 1.1:*** *What processes are used for or applicable to safety requirement elicitation in the automotive domain?*

> ***RQ 1.2:*** *What techniques are used for safety requirement elicitation in the automotive domain?*

Automotive systems differ from other safety-critical systems with high competition, yearly release cycles, and the price-sensitive nature. Another key difference is that the operator (driver) is not trained to handle safety-critical issues. Safety-critical domains like aviation, space, and nuclear always rely on highly trained operators. Also, the industry

is moving towards automated driving, thereby eliminating the operator from the loop, making automotive vehicles autonomous systems working among humans and human-operated non-automated driving vehicles. RQ 1.1 and RQ 1.2 focus on the technical aspects of safety requirement elicitation for the automotive domain, including different processes, their steps, techniques to perform these steps, and use cases where each process can be applied. We answer RQ 1.1 and RQ 1.2 in Chapter 2. By answering RQ 1.1, we intend to identify, summarize, and compare the various end-to-end processes for safety requirement elicitation. In RQ 1.2, we dive deeper into the techniques (used for different steps in requirement elicitation processes) and compare and taxonomize them. The major contributions presented in Chapter 2 are:

- A summary, analysis, and synthesis of the body of knowledge in safety requirement elicitation for the automotive domain through a systematic literature review over 102 primary studies.
- Empirical validation of the need for a literature review via a systematic qualitative analysis.
- The first taxonomy of processes and techniques for safety requirement elicitation in the automotive domain.

One of the findings from answering RQ 1.1 is that existing safety requirement elicitation processes do not consider connected driving. Even if there is a set of requirements for connected driving, there is a lack of methods to assess these requirements in the software architecture of the connected driving systems. In Chapter 3 we address these research gaps by answering the following two research questions.

> ***RQ 2.1:*** *How to derive safety requirements for connected driving?*

> ***RQ 2.1:*** *How to assess safety requirements in the software architecture of connected driving vehicles?*

In Chapter 3, we present a method to assess the functional safety of existing automotive architecture for connected driving by combining methods from the safety engineering and software architecture domains.

In Chapter 3, we primarily focus on the design phase (concept development phase in ISO 26262) and validation of the resultant requirements in the software architecture in the final product. In Chapter 3, we also present an application of our method to the architecture of an academic prototype capable of connected driving. The connected driving

scenario used to demonstrate our method is *platooning*, in which a manually driven vehicle is autonomously followed by a train of vehicles.

In summary our contributions in Chapter 3 are the following:

- A new methodology for eliciting and assessing safety requirements for connected driving use cases.
- A proof of concept application of the methodology on an academic prototype from Eindhoven University of Technology.

Another finding presented in Chapter 2 is the lack of real-life case studies in the safety requirement elicitation and assessment of automated driving. In Chapter 4, we address this research gap by answering the following two research questions.

> ***RQ 3.1:*** *What safety requirements shall be fulfilled by a vehicle's perception system for autonomous driving in Dutch highway?*

> ***RQ 3.2:*** *How to assess the safety requirements in the design of a perception system?*

In Chapter 4, we present a case study on the safety assessment of perception system of an automated driving software stack from industry, Apollo [11], for its use in a segment of the Dutch highway A270.[14] Apollo is the most popular open-source automotive repository [12] with its development history on GitHub dating back to 2017. It is currently one of the most advanced automated driving frameworks [13] that is embraced by many of the world's top automakers and is used to offer automated driving services to the public.[15]

In Chapter 4, we focus on the perception system of Apollo. Perception refers to sensing surroundings for semantic understanding, such as identifying traffic signs and locating the vehicle's position and relative position of objects around [14]. This information is used for planning and executing the next driving decision. These perception systems are built as a combination of machine learning (ML) based and traditional software [13, 15]. Perception systems are arguably the most evolving and relevant part of any automated driving framework [12].

In Chapter 4, we elicit safety requirements for two classes of systems: (1) requirements that can be assessed in the traditional software and (2) requirements specific to ML systems. While there is a framework proposed in Chapter 3 for assessing traditional software safety

[14] https://www.openstreetmap.org/directions?engine=fossgis_osrm_car&route=51.4564%2C5.5408%3B51.4657%2C5.5865#map=15/51.4610/5.5636

[15] https://www.engadget.com/baidu-apollo-go-robotaxi-shenzhen-141727050.html

requirements, there is no similar framework for assessing safety requirements specific to ML systems. Therefore, we prepare a curated list of ML specific design choices relating to safety and use them for design assessment.

In summary, contributions in Chapter 4 are the following:

- We identify 58 safety requirements specific to a Dutch highway segment of A270 that can enable safe automated driving on highways.
- We present a repeatable method for safety requirement elicitation based on the current industry standards and guidelines from the automotive industry consortium [5, 6, 16, 17].
- We present the first curated list of 10 ML specific design choices for assessing the quality attribute safety.
- We assess Apollo's perception system's design for its use on a Dutch highway.

RQ 1.1 through RQ 3.2 and their answers in Chapters 2 to 4 explore the safety of automotive software. RQ 4 explores another aspect: automotive software development (in open source).

RQ 4: *What characterizes automotive software projects in open source?*

In Chapter 5, we answer RQ4 in the following two dimensions by analyzing ≈600 automotive and a similar count of non-automotive projects on GitHub created in a span of 12 years from 2010 to 2021:
(1) Categories & characteristics: We identify what types of automotive software projects are open-sourced and compare them to each other. We also compare the automotive projects to non-automotive projects. Further, we explore the characteristics of automotive projects (e.g., size and maturity of the field) and their stakeholders (e.g., key players and affiliations).
(2) Software development styles: We investigate different aspects of software development like collaboration (e.g., types of contributors, their contributions and interactions) and contribution style (e.g., independent vs. dependent).

The major contributions in Chapter 5 are

- A manually curated, first-of-its-kind dataset of actively developed automotive software and their classification along four popular dimensions, including safety-critical software and tools [18]. This dataset facilitates the replication of this study and future explorations into automotive software.
- A characterization of automotive software, including its temporal trends, popularity, programming languages, user distributions, and development activities.

1

This study shows that the automotive software landscape in GitHub is defined by 15,000+ users contributing to ≈600 actively-developed automotive software projects created in a span of 12 years from 2010 until 2021. These projects range from vehicle dynamics-related software; firmware and drivers for sensors like LiDAR and camera; algorithms for perception and motion control; to complete operating systems integrating the above. Developments in the field are spearheaded by industry and academia alike, with one in three actively developed automotive software repositories owned by an organization. We observe shifts along multiple dimensions, including preferred language from MATLAB to Python and prevalence of perception and decision-related software over traditional automotive software.

1.4 Reading guide and chapter origins

The content of this thesis is organized into two significant parts: safety of automotive software & automotive software in open source. Each chapter in this thesis originates from one or more papers, is self-contained, and can be read individually. The work in this thesis is a part of an automated and connected driving research program i-CAVE (Integrated Cooperative Automated VEhicles). The i-CAVE program intends to research and demonstrate connected driving capabilities from ground up.

The thesis starts with an overview of the first step in ensuring safety, safety requirement elicitation, in Chapter 2. Chapter 2 is based on the following publication.

- Sangeeth Kochanthara, Loek Cleophas, Yanja Dajsuren, Mark van den Brand. "Requirements Engineering for Safety of Automotive " *Submitted to a journal*

Chapters 3 and 4 address the research gaps in safety requirement elicitation identified in Chapter 2. Chapters 3 and 4 also address the safety assessment of automated and connected driving systems along with real-life case studies.
Chapter 3 is based on:

- Sangeeth Kochanthara, Niels Rood, Arash K. Saberi, Loek Cleophas, Yanja Dajsuren, Mark van den Brand. "A Functional Safety Assessment Method for Cooperative Automotive Architecture". In Journal of Systems and Software (**JSS'21**)

  In European Conference on Software Architecture (**ECSA'21**) - *Journal first track*

  Invited to Journal first track of the International Conference on Software Architecture (**ICSA'22**)

- Sangeeth Kochanthara, Niels Rood, Loek Cleophas, Yanja Dajsuren, Mark van den Brand. " Semi-automatic Architectural Suggestions for the Functional Safety of Cooperative Driving Systems". In International Conference on Software Architecture (**ICSA'20**) - New and Emerging Ideas track

Chapter 4 is based on:

- Sangeeth Kochanthara, Tajinder Singh, Alexandru Forrai, Loek Cleophas. "Safety of Perception Systems for Automated Driving: A Case Study on Apollo" *Under revision at a journal*

Chapter 5 presents a characterization of automotive software and its development in open source. Chapter 5 is based on:

- Sangeeth Kochanthara, Yanja Dajsuren, Loek Cleophas, Mark van den Brand. "Painting the Landscape of Automotive Software in GitHub". In International Conference on Mining Software Repositories (**MSR'22**) - *One of the five out of 47 accepted MSR 2022 papers invited for an extended journal version*

Finally, Chapter 6 concludes the thesis.

2

Automotive Safety Requirements: A Systematic Literature Review

Automotive is a 3 trillion dollar safety-critical industry. The ongoing paradigm shift from electro-mechanical systems to software-intensive systems, the move toward automated driving, high competition, yearly release cycles, and the price-sensitive nature set the automotive industry apart from other safety-critical industries. As in other safety-critical industries, ensuring safety in the automotive industry starts with safety requirements. Even though a plethora of safety requirements elicitation processes and techniques for the automotive domain have been proposed, to the best of our knowledge, no study characterizes them. This chapter characterizes the state-of-the-art in eliciting safety requirements via a systematic literature review. We select 102 primary studies from 2097 related articles. We identify and compare nine distinct processes for safety requirement elicitation and construct taxonomies of 38 distinct techniques used to conduct the different steps in each of these processes. This chapter can act as a guide and 'cheat sheet' for beginners and practitioners to choose processes and techniques for their projects. For researchers, this chapter provides an overview of the field, research gaps, and future research opportunities.

This chapter is based on:
S. Kochanthara, L. Cleophas, Y. Dajsuren, M. van den Brand. Requirements Engineering for Safety of Automotive. Submitted to a journal

2

With an estimated 1.4 billion cars on the road and a direct market capitalization of 3 trillion dollars,[1] the automotive industry forms a core part of modern civilization. The century-old automotive industry is undergoing arguably the most significant paradigm shift since its inception with (a) the move to driving automation and (b) the change of propulsion systems from fossil fuel to electric-based, along with associated changes in the powertrain[2]. These disruptions make the automotive industry a software-intensive and electronic industry rather than a traditional electro-mechanical one. This paper focuses on the safety of automotive software and the electronics that run the software. This is also referred to as *functional safety* [5] and *safety of intended functionality* [6]).

There is an inadequacy in safety of automotive software and electronics that run it, especially in comparison to electro-mechanical systems. For instance, software and related defects led to the recall of 7.5 million and 5.5 million vehicles in the United States in 2020 and 2021, respectively. These recalls dominated the top reasons for recalls both in the number of recalls and the number of affected vehicles.[3] Another example is the infamous Uber automated driving vehicle crash, which led to the death of a pedestrian. NTSB's[4] investigation found that software and inadequate safety culture in developing software and related systems were among the reasons for this crash.[5] Such inadequacies in ensuring safety can cause fatalities, economic losses, brand damage, destruction of traffic infrastructure, and indirect (economic) losses. A fundamental step for safer automotive software systems is incorporating safety into the automotive product life cycle right from the requirement elicitation stage.

There are many kinds of safety requirements based on the underlying cause of the requirement. For instance, safety requirements caused by deficiencies in specified driving behavior [7], safety requirements relating to failure or malfunction of components (also referred to as *functional safety* [5]), safety requirements resulting from functional insufficiencies of the intended functionality (also referred to as *safety of intended functionality* [6]). This paper focuses on the latter two types of safety requirements, in the context of systems, software, and hardware that runs the software. In the rest of this paper, safety requirements refer to these two types.

Our *goal* is to characterize the state-of-the-art in safety requirement elicitation processes and techniques in the automotive context via a Systematic Literature Review (SLR).

Secondary studies on safety requirement elicitation focused on: (a) the broader safety-

[1] https://www.bloomberg.com/opinion/articles/2021-12-29/cars-are-suddenly-worth-3-trillion-and-it-s-not-all-tesla

[2] Powertrain is the set of components, including engine or motor, that generates (and possibly stores) power and converts it to the rotation of the wheels.

[3] https://www.recallmasters.com/wp-content/uploads/2021/05/Infographic_2020_smaller.png

[4] National Transportation Safety Board (NTSB) is a U.S. government agency for civil transportation accident investigation.

[5] https://www.ntsb.gov/news/events/Pages/2019-HWY18MH010-BMG.aspx

critical systems domain [19]; (b) practices and challenges in the development of embedded systems [20]; (c) integration between requirement and safety engineering [21]; (d) managing safety in mobile robotic systems [22]; and (e) safety in the context of product lines [23]. However, to our knowledge, there are no secondary studies on safety requirement elicitation for automotive systems.

Automotive systems differ from other safety-critical systems. One difference is that the operator (driver) is not trained to handle safety-critical issues, while other safety-critical domains like aviation, space, and nuclear always rely on highly trained operators. Furthermore, the industry is moving towards automated driving, thereby eliminating the operator from the loop, making automotive vehicles autonomous systems working among humans and non-automated driving vehicles.

This paper presents the first SLR on safety requirement elicitation for automotive software and systems. Specifically, we focus on the technical aspects of safety requirement elicitation, including different processes, their steps, techniques to perform these steps, and use cases for each process.

We translate our goal into the following research questions:

RQ1: What processes are used for or applicable to safety requirement elicitation in the automotive domain?
By answering RQ1, we intend to identify, summarize, and compare the various safety requirement elicitation processes.

RQ2: What techniques are used for safety requirement elicitation in the automotive domain?
In RQ2, we dive deeper into the techniques (used for different steps in the processes) and compare and taxonomize them.

Our primary contributions are:

- A summary, analysis, and synthesis of the body of knowledge in safety requirement elicitation for the automotive domain through an SLR over 102 primary studies.
- Empirical validation of the need for such an SLR via a systematic qualitative analysis.
- Taxonomies of techniques for safety requirement elicitation in the automotive domain.

This chapter targets practitioners and researchers alike. For researchers, this study outlines the automotive safety requirements elicitation research field, research gaps, and future research opportunities. For practitioners, this paper provides a concise guide toward choosing one or more processes and techniques for safety requirement elicitation in their projects.

The rest of this paper is structured as follows. Section 2.1 presents this chapter's planning phase and design choices. Section 2.2 describes an overview of the research landscape via qualitative metrics. Sections 2.3 and 2.4 answer our research questions and discuss our findings. Section 2.5 elaborates implications of this chapter for research and practice, while Section 2.6 presents validity threats. Related work is outlined in Section 2.7. Finally, Section 2.8 concludes the paper.

2.1 Study design

The different steps in conducting an SLR can be organized into three phases: planning, conducting, and reporting [24]. This section primarily discusses the study's planning phase and our design choices in detail. Reporting includes drawing conclusions, considering threats, and disseminating results which are covered in later sections of the chapter.

The planning phase of this study is divided into the following 5 stages (based on various guidelines [24–29]):

- Evaluate the *need for this SLR* (Garner et al. [25]);
- Form a *search strategy* (Kitchenham et al. [24], Petticrew et al. [26], and Petersen et al. [27]);
- Create a *primary studies' selection* procedure (Kitchenham et al. [24]);
- Identify *quality assessment* criteria for the primary studies (Kitchenham et al. [24], Tiwari et al. [28], and Wieringa et al. [29]); and
- *Extract and synthesize data* and insights from primary studies (Kitchenham et al. [24]).

The rest of this section explains each of these stages in detail.

2.1.1 Need for SLR

To evaluate the need for this study, first, we conduct an initial search for secondary studies, followed by a qualitative empirical evaluation.

For the initial search, we use the search string: *"functional safety" AND (automotive* OR vehic*) AND ("systematic map" OR "systematic mapping" OR "systematic literature")*. The search string is created by combining the topic (*"functional safety"*), domain (*automotive* OR vehic**), and the kind of studies we are looking for (*"systematic map" OR "systematic mapping" OR "systematic literature"*). Note that we used the term *"functional safety"* instead of *"safety"* since the former resulted in a low signal-to-noise ratio. *Functional safety* refers to safety concerning (malfunction of) software and related systems. The term is universally adopted in the automotive domain [5, 6].

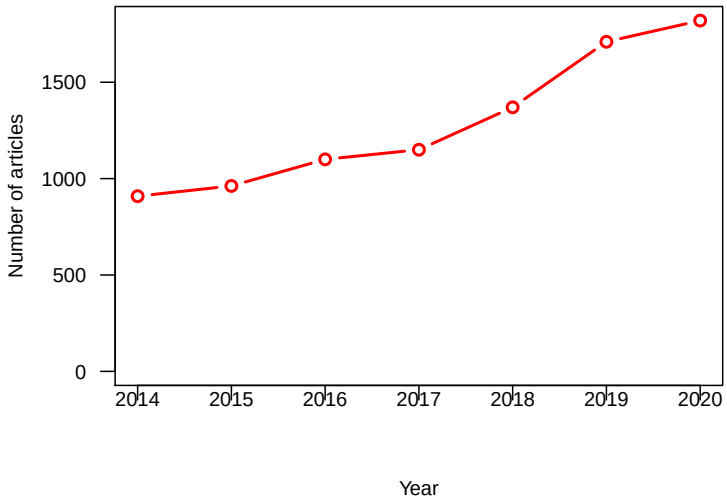


Figure 2.1: Number of articles in the result of Google scholar search of query: *"functional safety" AND (automotive* OR vehic* OR driv* OR automobile* OR AUTOSAR OR car)*. The query was performed on the 2nd of August 2021

Our Google Scholar search (on November 10th, 2020) did not identify any secondary study that focuses on safety requirement elicitation in the automotive domain. However, we found an SLR by Martins et al. [19] on approaches to elicit, model, specify, and validate safety requirements in the broader context of safety-critical systems. In this SLR, automotive formed 5.29% (8 out of 151) of all the primary studies [19]. Given the study's broader scope and rigor, we assumed that primary studies until 2013 are covered (since the primary studies search was conducted in 2014). We consider this study by Martins et al. [19] as a basis to evaluate the need for our SLR.

We evaluate the need in two steps: (1) an initial analysis of the trend in the number of relevant publications since 2014 as shown in Figure 2.1, which indicates a positive trend; and (2) a systematic approach for qualitative empirical evaluation, presented in the rest of this section.

We use the 3PDF framework [25] to validate the relevance of conducting this SLR empirically. The 3PDF framework is an empirical framework consisting of three sequential phases. A positive outcome of all three phases ascertains the need for an SLR. Note that the 3PDF framework [25] was originally designed to address the relevance of repeating an existing SLR. Meanwhile, we use it to identify the significance of this study using the SLR by Martins et al. [19] as a basis. Next, we explain the 3 phases of the 3PDF framework.

- *Phase 1: Assess currency*, aims to identify whether the research questions are already answered by available evidence or no longer considered relevant [25, 30]. This phase consists of the following three *yes/no* questions and is passed if and only if **all** of them are answered positively.

  1. *Does the published SLR still address a current question?*
     Yes. The study by Martins et al. [19] focuses on safety requirement elicitation for safety-critical systems with ≈5% of their primary studies from the automotive domain.
  2. *Has the SLR had good access or use?*
     Yes. The prior SLR [19] has a yearly average citation count of 15.5 (with a total citation of 62 when checked via Google Scholar on the 10th of November 2020). Citation count is a measure of access, use, and relevance [31, 32]. In the software engineering domain, a yearly average citation of 6.82 or above is judged as having had good access or use [31].
  3. *Has the SLR used valid methods and was it well-conducted?*
     Yes. Martins et al. [19] followed the well-established guidelines of Kitchenham [24] for conducting the SLR and the work has been published in a top-tier peer-reviewed journal from the domain.

  Since all the questions are answered *yes*, we proceed to the next phase.

- *Phase 2: Identify relevant new methods, studies, and other information.* This phase focuses on whether new information is not covered by the existing study, including study design, evidence synthesis, and new primary studies. The phase consists of two *yes/no* questions, and proceeding to the next phase needs **at least one** question to be answered positively.

  1. *Are there any new relevant methods (in conducting the SLR)?*
     Yes. In addition to the approaches used in the study design of the prior SLR [19], we add the following new methods: (a) full-text search and (b) both forward and backward snowballing.
  2. *Are there any new studies or new information?*
     Yes. The period of our initial search is between the end of Martins et al.'s study (2014) and 2020. This ensures that every primary study we considered is not included in their research. Furthermore, this period experienced landscape shifts in the automotive industry with automated and connected driving enabled by software-intensive sub-systems [12].

  Both the questions are answered with *yes*; therefore, we advance to the next phase.

- *Phase 3: Assess the effect of updating the review.* This phase aims to assess whether the information from the new primary studies influences the conclusion compared to the base SLR. A **Yes** answer to **any** of the two following questions empirically validates the need for a new study.

  1. *Will the adoption of new methods (for conducting the SLR) change the findings, conclusions, or credibility?*
     Maybe. The new methods that we adopted were full-text search and snowballing. Our initial set of primary studies is disjoint from the set of primary studies considered in the prior SLR [19]. Also, snowballing has resulted in identifying new techniques and wider adoption of some of the processes and techniques for safety requirement elicitation, which were not evident otherwise.
  2. *Will the inclusion of new studies, information or data change findings, conclusions, or credibility?*
     Yes. Only 5.29% of primary studies in Martins et al. [19] are from the automotive domain. We scope our work specifically to the automotive domain. This makes a direct comparison of our study with theirs inaccurate. Also, the study does not differentiate or taxonomize processes and techniques or present challenges specific to the automotive context.

Thus, the execution of the 3PDF framework empirically establishes the need for our SLR.

2.1.2 Search strategy

Our search strategy consists of identifying search keywords and multiple iterations to compose a search string followed by automatic search. We perform both forward and backward snowballing to widen the set of primary studies beyond the initial search.

The search string was constructed from two sets of strings, one for scoping: *(automotive, vehicle, vehicular, drive, driving)* and another related to the intervention: *(functional safety, hazard, accident, risk).* The final search string was formed by iterative refining (via piloting) to reduce the amount of noise while covering as much relevant literature as possible. The following search string was formed after several iterations:

The term *"functional safety"* is searched within keywords, title, and abstract *AND* the following terms were searched in the full text of publications.

(automotive OR vehicle OR vehicular OR drive OR driving) AND
(hazard OR risk OR accident)

To identify and compose the search string, we use the PICOC (Population, Intervention, Comparison, Outcome, Context) method [24, 26, 27] as detailed below.

- *Population*: peer-reviewed publications describing safety requirement elicitation processes and techniques as well as application of such approaches to the automotive domain [24, 26, 27].
- *Intervention*: processes or techniques for safety requirement elicitation [24, 26].
- *Comparison*: compare the different processes and techniques of safety requirement elicitation by means of identifying the different strategies used, and their context of application [24, 26, 27].
- *Outcome*: safety requirement elicitation process, different stages in the processes, techniques used to conduct each stage, and the following aspect of each process/technique: (a) context of use, abstraction level of application, and applicable components [24, 26, 27].
- *Context*: any phase in the automotive product life cycle that comprises safety requirement elicitation [24].

We use the same search databases as in the related studies [19, 21]. The databases included are IEEE Xplore, ACM Guide to Computing Literature, ScienceDirect, and SpringerLink. Except for SpringerLink, automatic search is directly performed in the corresponding database. Since SpringerLink does not have a feature for the intended search string, we used broader search criteria (searching in the whole body of publications rather than title, tags, and abstract), with which a bibliography is retrieved. A further refined search is performed within this bibliography using the reference manager Mendeley.

Once the initial set of primary studies is finalized based on full-text reading, we applied snowballing to gather additional studies. We applied the guidelines by Wohlin et al. [33] to conduct backward and forward snowballing. For backward snowballing, we checked the references of primary studies (using titles only) until no new studies were found. Likewise, for forward snowballing, we looked at the articles (titles only) citing our primary studies until no further studies were found. For forward snowballing, we used the 'cited by' feature of Google Scholar and went through the citations to every primary study. The number of studies considered in each of the steps mentioned above is presented in Figure 2.2.

2.1.3 Study selection

We create inclusion-exclusion criteria tailored to focus on the technical aspects of safety requirement elicitation, which is the focus of this study. We excluded studies on other dimensions of safety requirement elicitation. Some examples are: (a) social and human-related factors that play a role in eliciting requirements like meetings, reviews, communication

among different parties; (b) a combination of technical, social, and human-related factors; and (c) processes and techniques that merge safety requirement elicitation with requirement elicitation for other quality attributes like security.

Our inclusion and exclusion criteria are as follows:

Inclusion criteria:

I1 Any study that presents, compares, or discusses approaches (techniques, models, frameworks, methods, processes, or methodologies) to (help) elicit safety requirements, either used in or usable for the automotive domain.

I2 Studies relating to safety requirements in the context of safety analysis, hazard analysis, or safety-critical standards from the automotive domain.

The **exclusion criteria** cover secondary studies; articles that are not written in English; non-peer-reviewed articles (gray literature); short papers (< 4 pages); studies below a quality threshold of 50% according to the quality assessment criteria detailed in Section 2.1.4 below; and studies that do not explicitly specify or are not from the automotive domain.

To ensure reproducibility, we conducted an inter-rater agreement. Before the first author selected primary studies, the rest of the authors evaluated the selection criteria. To assess the quality of the selection process, the second and third authors independently used the selection criteria on two disjoint random samples of the initial set of studies. The first author independently evaluated these two sets of studies employing the same criteria, resulting in the inter-rater agreement measured to 0.8 and 1 with the second and third authors, respectively, according to Cohen's kappa statistics [34]. This shows the highest level of inter-rater agreement in both cases.[6] Each disagreement between two researchers was discussed and resolved with the intervention of a third researcher.

We followed a five-step study selection procedure as presented in Figure 2.2. We incrementally apply the inclusion-exclusion criteria in steps 1, 3, 4, and 5. This includes applying inclusion-exclusion criteria on the additional studies resulting from snowballing (only 17 studies were selected after the application of inclusion-exclusion criteria on the 48 studies resulting from snowballing).

2.1.4 Quality assessment

Quality assessment of primary studies is crucial (i) *"to investigate whether quality differences provide an explanation for differences in study results"*; (ii) as a *"means of weighting the importance of individual studies when results are being synthesised"*; and (iii) *"to guide recommendations for further research"* [24].

We derived the quality assessment criteria from the guidelines of Kitchenham et al. [24], Tiwari et al. [28], and Wieringa et al. [29], as summarized in the first column of Table 2.1.

[6] Since Cohen's Kappa statistic being at least 0.80 is considered best agreement [35, 36].

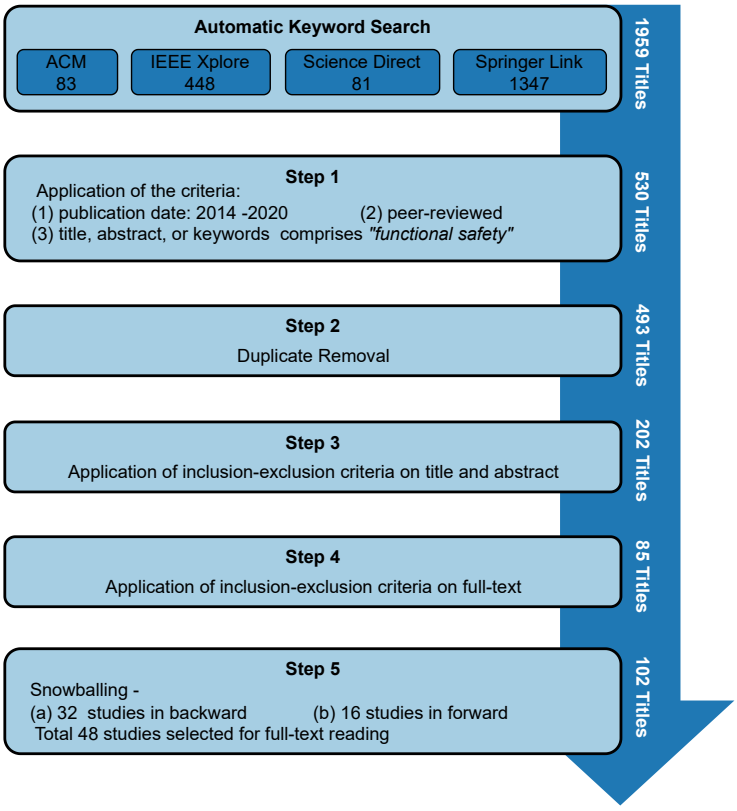


Figure 2.2: Procedure to select primary studies

We chose those questions from the guidelines that apply to our list of primary studies. For example, the question *"Was there any control group present with which the treatments can be compared"* [28] does not apply to studies that present the application of a requirement elicitation technique to an automotive component.

All the questions that form our assessment criteria have three possible answers *"Yes", "Partially,"* and *"No,"* with a score of 1, 0.5, and 0, respectively. The score of each primary study is the sum of scores for every pertinent question (see Table 2.1). We use this scoring method to gauge the primary studies' credibility, completeness, and relevance.

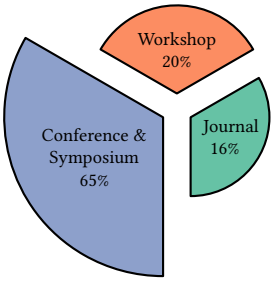
2.1.5 Data extraction and synthesis

We iteratively created a digital data extraction form. We created an initial set of attributes to be extracted from the primary studies and applied this to 10% of the initial set of primary studies. This form was iterated based on data synthesis at 30%, 50%, and 70% (of the initial set of primary studies) based on the categories and information emerging from data extraction. We applied a backward pass on the prior studies in cases where extra attributes were added to the data extraction form during the process. Finally, the following categories of data were extracted from primary studies:

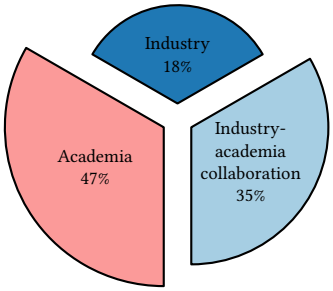
- *Administrative information:* Paper ID; Title; and Source.
- *Literature characteristics:* Authors; Year; Venue type (journal, symposium, conference, or workshop); Venue;
- Data pertaining to *RQ1: What processes are used for or applicable to safety requirement elicitation in the automotive domain?*
  - What process(es) are employed, compared, or discussed?
  - What is the motivation or reason to choose a specific process (or how a process compares with other processes)?
  - What are the steps followed in executing the process?
  - Which component or setting is the process applied to?
- Data related to *RQ2: What techniques are used for FSR elicitation in the automotive domain?*
  - What are the techniques employed, compared, or discussed?
  - Which high-level step (of a process) is accomplished using the specific techniques used or discussed?
  - What is the motivation or reason to choose a specific technique (comparison among techniques)?
  - Which component or setting is the technique applied to?

The data extraction form is a table with one column for each of the above categories and a row for each paper.

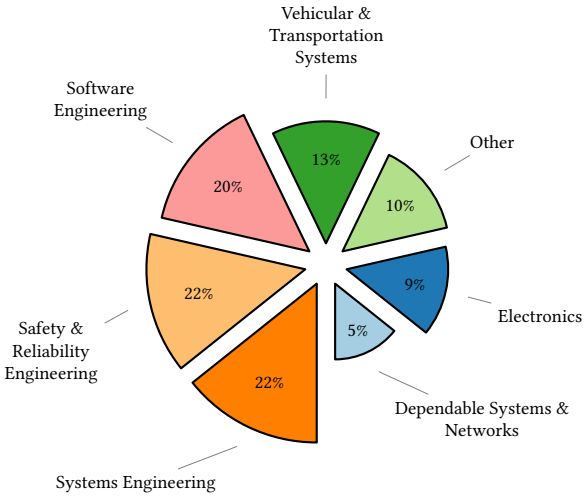
Data synthesis was performed after aggregating the information collected using digital forms. The data synthesis is achieved by cross-reading each column to answer the research questions.



(a) Primary studies classified on type of venues.



(b) Source of primary studies based on author affiliation



(c) Primary studies categorized on domain

Figure 2.3: An overview of the publication, domain, and contribution landscapes of the primary studies

2.2 Characteristics of primary studies

Our study selection process resulted in 102 primary studies [P1–P102]. This section presents an overview of the publication landscape, quality, and a preliminary analysis of the primary studies. We analyze the publication trend across venues. We also classify the studies based on their domain and on whether the study is a contribution from industry, academia, or both.

Table 2.1: Quality assessment summary of primary studies. The bold face part shows two aspect that only a few papers report

Percentage	Yes	Partially	No
Aim clearly presented	100%	0%	0%
Approach clearly explained	83%	17%	0%
Clarity of application context	31%	63%	6%
Threats to validity taken to consideration	**6%**	**1%**	93%
Presence of discussion on results	46%	31%	23%
Limitations/Scope discussed	**6%**	**9%**	85%
Related work discussed	64%	27%	9%

Quality assessment of primary studies shows that most existing studies do not report limitations and scope of their solution. We assessed the quality of each study according to the criteria summarized in Table 2.1. On application of our quality assessment criteria, we found that only a few papers (at least partially) report the following two aspects (highlighted with boldface in Table 2.1): (1) threats to validity, taken into account in 7% (7 out of 102) of primary studies; and (2) limitations, discussed in 15% (15 out of 102) primary studies. The first observation is not surprising since only a few studies (4 out of 102) report empirical analyses. However, with a few studies reporting scope or limitations makes the reuse and replication of a majority of studies difficult. Our advise for future papers is to explicitly state the limits and scope of proposed solution.

Quality assessment of primary studies

Most existing studies do not report limitation and scope, further limiting our ability to reuse. We advise future studies to report it.

Most studies in safety requirement elicitation are published at conferences. We classified studies based on the type of venue as shown in Figure 2.3a. Most of the primary studies (66 out of 102) are published at conferences and symposiums. Twenty studies are published in workshops and sixteen in journals.

The publication landscape of safety requirement elicitation has substantial contributions from industry. We classified the studies according to industry, academia, or industry-academia collaboration contributions. Two intuitive means to identify the source of contributions are (1) the affiliation of authors and (2) the source of case studies and data. The latter is not feasible in our context since many primary studies do not explicitly specify the source of the case study, data, or the examples they use. Therefore, we choose to classify the source of the study based on the author's affiliation into three categories: (1) industry, (2)

academia, and (3) industry-academia collaboration. If all the authors are affiliated with the industry, then the study is considered from industry and similarly for academia. If a study has author affiliations from both industry and academia, it is classified as industry-academia collaboration. The resulting classification is presented in Figure 2.3b. This categorization shows that most of the primary studies (54 out of 102) have some contributions from industry, making the safety requirement elicitation publication landscape one with a solid industrial contribution.

Publication landscape

A majority of studies are published in conferences and symposiums with 53% of all studies having contributions from industry.

Safety requirement elicitation is a multi-disciplinary field of research. We examined the domain of each primary study. We identified the domains by research areas listed on the web page of the publication venue of each primary study. When a venue represented one domain, publications were classified to that domain. There were also cases where a venue represented more than one domain. Here, we chose to report the domains together if most of the venues listed two or more areas together, like reliability engineering and safety engineering. Otherwise, we counted the study in both areas. For this analysis, we excluded venues like the International Conference for Convergence in Technology, which does not have specific research areas, or venues like the International Conference on Networks, Communication, and Computing, which specify a wide variety of unrelated domains. This does not affect our classification since less than 1% of the primary studies are from such venues. All the domains that cumulatively covered less than 5% of the publications were classified as 'other' in the final list of domains. The resultant set of domains is presented in Figure 2.3c. The set of domains depicted shows the multi-disciplinary nature of safety requirement elicitation research.

Multi-disciplinarity

Primary studies span across different disciplines with a domination of software, systems, reliability and safety engineering.

2.3 Safety requirement elicitation processes (RQ1)

In this section, we identify, summarize, and compare processes (also referred to as methodologies [P32,P59]) used or discussed in the primary studies for safety requirement elicitation in the automotive context. In our context, processes or methodologies refer to the high level steps that describe the elicitation of safety requirements [5,6,37] [P20,P40,P46,P59,P77,P86].

For example, the safety requirement elicitation process prescribed by the industry-standard ISO 26262 consists of four high-level steps as shown in Figure 2.4.

We organize the rest of this section into two parts. Section 2.3.1 compiles the safety requirement elicitation processes proposed, used, or discussed in the primary studies. Section 2.3.2 analyzes the application of the processes reported in the literature, their temporal trend, the associated research gaps, and the upcoming domain trends.

2.3.1 Findings

The primary studies mention 9 unique processes for safety requirement elicitation. We organize them into the following 7 categories (① - ⑦).

Processes

There are 9 unique safety requirement elicitation processes in the automotive domain.

① *ISO 26262*: The ISO 26262 standard [5] forms the basis for safety requirement elicitation in the automotive domain. One focus area of the standard is risks from systematic and random hardware failures in automotive electronic systems and the software that runs them (also termed functional safety). The standard is an adaptation of IEC 61508 [38] to the automotive domain, the latter being a generic standard that outlines safety guidelines for developing *any* electronic and programmable systems that carry out safety functions.

The safety requirement elicitation process outlined by the standard (ISO 26262: part-3 [5]), highlighted by the dashed red rectangle in Figure 2.4, comprises four steps: item definition, hazard analysis, risk assessment, and safety analysis. The item definition step specifies the system and its boundaries on which the rest of the steps will be performed. The hazard analysis step first identifies possible situations that can be a safety risk, followed by safety goals to prevent harm in those situations. The risk assessment step assesses the level of risk for each safety goal based on three factors:
(a) Exposure (frequency or probability of the safety goal violation);
(b) Controllability (in situations of safety goal violation by the driver); and
(c) Severity (of harm on violation of the safety goal). It then allocates a score from A through D, indicating the importance of covering a safety goal during further development steps, with D indicating the highest risk level and A the lowest. This score is called the Automotive Safety Integrity Level (ASIL) score. Note that these safety goals have a system-wide scope. Finally, the safety analysis step maps these safety goals to safety requirements for individual sub-systems or components, which can then be used in developing individual sub-systems.

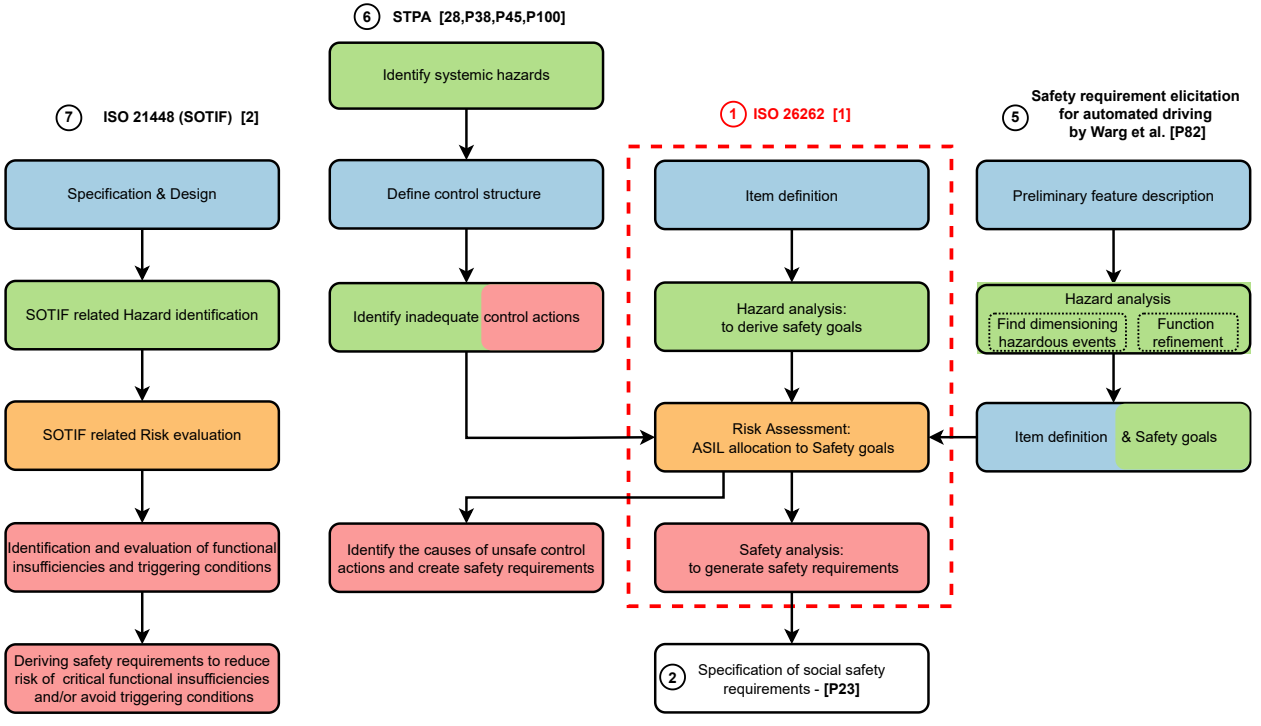


Figure 2.4: Different safety requirement elicitation processes from the automotive domain. The ISO 26262 process (highlighted in red) forms the basis for safety requirement elicitation in the domain. The circled numbers (①, ②, ⑤-⑦) point to the part of Section 2.3.1 in which the corresponding methodology is discussed. Similar steps across different processes are color-coded the same. The methodologies proposed by Schoenemann et al. [P59], Saberi et al. [P86], and Kochanthara et al. [P32], described in ③ and ④ in Section 2.3.1 use the entire ISO 26262 process without any modification (with the primary difference of partitioning the scenarios space before starting the ISO 26262 process). Therefore, we have not presented them in the figure.

The above process, prescribed by the ISO 26262 standard, forms the basis of safety requirement elicitation in the automotive domain. Note that we do not differentiate between the 2018 and 2011 versions of ISO 26262 or its regional variants (for example the Chinese GB/T 34590 safety standard, used by Zhou et al. [P85]) since the fundamental concepts and high-level steps do not differ among them. The other processes from the primary studies that are described in this section build on or complement ISO 26262 in various ways, fundamentally differing from ISO 26262's process either in its concept, model, or in the high-level steps.

Basis

The process outlined by ISO 26262 (part-3) forms the basis of all safety requirement elicitation processes in the automotive domain.

The rest of this section is organized as follows. Parts ② - ④ describe five processes that extend ISO 26262 requirement elicitation. After those we elaborate on three methodologies proposed as replacement for ISO 26262 in parts ⑤ - ⑥. Finally, we present a process (⑦) that complements the ISO 26262 elicitation process.

② *Social safety requirements*: Gharib et al. [P21] argue that ISO 26262 does not consider safety requirements specific to driver behavior, which they term as *"social safety requirements"*. An example of a social safety requirement is *"identify available information concerning the driver state (e.g., head pose and motion, hands and foot location and motions) at any point in time"* [P21]. Gharib et al. [P21] propose to add a step to elicit social safety requirements at the end of ISO 26262 requirement elicitation, as shown as a white box below the ISO 26262 process flow in Figure 2.4. They also present an example of the proposed extension on a maneuver assistance system to detect and respond to drivers' unintended maneuvers. However, they did not describe any systematic method to elicit social safety requirements.

Social safety requirements

Social safety requirements are safety requirements specific to driver behaviour, proposed by Gharib et al. [P21]. Details on how to elicit these requirements are not available.

③ *Safety requirements for connected driving*: Connected driving refers to multiple vehicular systems and traffic infrastructure communicating and working as a single system. This forms a system of systems and enables collective traffic optimizations. Such a system of systems perspective is missing in ISO 26262 [P32, P86], thus making ISO 26262 unsuited

for use in the connected driving context. Saberi et al. [P86] and Kochanthara et al. [P32] propose augmentations to the ISO 26262 process for this context.

The process by Saberi et al. [P86] uses the steps for safety requirement elicitation from ISO 26262 but is executed separately, from a system of a systems perspective. Their extension to ISO 26262 is inspired by hazard analysis for the system of systems from other domains [39]. Saberi et al. [P86] also partially outlines the usage of their method in the context of a connected driving use case of trucks, where a lead truck is driven by a driver and a sequence of other trucks autonomously follows the leader.

Kochanthara et al. [P32] suggest a two-step extension to the ISO 26262's process, considering the heterogeneity of connected systems. i.e., traffic infrastructure and multiple kinds of vehicles (for example, trucks and cars or different types of cars) forming a connected system. Their first step is partitioning the scenarios for connected driving operations into sets of scenarios specific to each different system (vehicle) and those common to or involving the entire connected system. Thus, a connected driving system consisting of n distinct types of vehicles will have $n+1$ sets of scenarios. Then, the ISO 26262 process is applied to each of the n distinct types of vehicles separately. The partitioning and applying ISO 26262 process on each of the n distinct types of vehicles, is the first step. In the second step, the scenarios specific to the connected system, i.e., common to the entire connected system or involving more than one vehicle, are considered. A connected system architecture is created using the individual systems' (distinct types of vehicles' and traffic infrastructure's) components. The connected system architecture should show the possible interactions across the individual systems' components. Now, the ISO 26262 method should be applied to the entire connected system using its architecture and the scenarios specific to the connected system. They also show the applicability of their approach in a similar use case as in Saberi et al. [P86].

Connected driving

Connected driving refers to multiple vehicular systems and traffic infrastructure communicating with each other and working as a single connected system. To elicit safety requirements for connected driving, Saberi et al. [P86] and Kochanthara et al. [P32] extends the ISO 26262 process to cover both individual-vehicle perspecitve(s) and a connected system perspective.

④ *Scenario-based safety analysis for automated driving*: Schoenemann et al. [P59] argue that the number of operating situations to be considered for safety requirement elicitation for automated driving might be unlimited, making the ISO 26262 process infeasible. Their proposal extends the ISO 26262 process for automated driving with the abstraction of the system's behavior to limit the number of parameters. The proposed process starts with decomposing the system's functional behavior into a higher abstraction of functional scenarios and then performing ISO 26262 safety requirement elicitation specifically for each functional scenario. Thus, the steps (other than decomposing the system's functional behavior into functional scenarios) are the same as ISO 26262.

⑤ *New requirement elicitation process for automated driving*: Warg et al. [P77] argue that ISO 26262 assumes a driver for fallback in case of a malfunction or failure of a component. At the same time, there is no driver to fall back to in the case of entirely automated driving. To address this gap, Warg et al. [P77] present an approach based on the ideology of adequately specifying the system and its function iteratively using their version of hazard analysis. This process ensures that the automated driving functions are adequately specified. It also ensures that all the relevant hazardous events related to all functions that enable automated driving are covered. Their process differs from the one by Schoenemann et al. [P59] (see ④ above) to automated driving in two aspects: (1) Schoenemann et al. [P59] uses the ISO 26262 process as is, while Warg et al. [P77] proposes a new iterative process to replace it; and (2) Schoenemann et al. [P59]'s process is aimed to scale the ISO 26262 process to an in-feasibly high number of operating situations via abstraction. At the same time, Warg et al.'s ideology is based on iteratively defining a function and minimal representative set of hazardous conditions.

Warg et al. proposed a new process as shown as the rightmost flow in Figure 2.4. It differs from the ISO 26262 process in the first two steps: preliminary feature description and new hazard analysis. The preliminary feature description step describes the proposed feature's anticipated benefits and initial scope. The hazard analysis step consists of two sub-steps: finding dimensioning hazardous events and function refinement. The dimensioning hazardous events are the sets of hazardous events that are sufficient to identify all critical safety goals. The function refinement sub-step aims to elicit requirements for the intended nominal functionality and define the scope of each function. Therefore, the hazard analysis step results in (ideally) a minimal set of hazardous events and refined functionality from the preliminary feature description.

Another significant difference between this process and ISO 26262 is that the definition of the system and its boundaries (item definition) is an intermediate step in Warg et al.'s [P77] process, yet it is the starting step in ISO 26262 (see Figure 2.4).

Automated driving

Automated driving refers to a vehicle taking over (part of the) driving and eliminating the need for a human for (some) driving tasks. Schoenemann et al.'s [P59] and Warg et al.'s [P77] proposes processes to address two limitations of the ISO 26262 process in an automated driving context: (a) the possibility of unlimited number of conditions [P59] and (b) the assumption of driver fallback which leads to inadequate specification of functionalities, respectively.

⑥ *Systems Theoretic Process Analysis (STPA)*: The primary difference between STPA and ISO 26262 is that STPA treats accidents as a control problem rather than a failure problem. STPA is proposed for safety requirement elicitation in the general context of safety-critical systems from a system theoretic point of view [37]. In literature, STPA has been used in the automotive domain in two contexts: (1) as a methodology for safety requirement elicitation that complies with the ISO 26262 standard [P33, P40, P94]; and (2) as a technique for the steps (similar to) hazard analysis and safety analysis [P89] [6], from the ISO 26262 and ISO 21448 (described in ⑦) processes. In this section, we discuss the former context.

STPA intends to prevent accidents by enforcing constraints on component behavior and interactions. The underlying model of STPA promises to cover more causes of accidents than component failures, like design errors and flawed requirements [37]. Therefore, two dominant reasons to advocate STPA over the ISO 26262 process are: (1) the wide variety of error types it covers (ISO 26262 only covers random hardware failures and systematic failures) [P33]; and (2) less dependence on expert knowledge [P33, P89]. The error types STPA covers include component failures, inadequate component interactions, software failures, human error, and system failure [P33]. Another advantage of STPA is that expert knowledge is not always required for requirement elicitation [P33, P89].

However, the original STPA process [37] lacks a risk assessment step. Mallya et al. [P40] showed that the STPA process could be augmented to include a risk assessment step from ISO 26262 as shown in flow ⑥ STPA) in Figure 2.4. The STPA safety requirement elicitation process itself consists of 4 steps, as shown in Figure 2.4. The first step (identify systemic hazards) identifies the possible accidents and hazards that can cause these accidents and is similar to the hazard analysis part of ISO 26262. The second step of defining a functional architecture (control structure in STPA's original terminology) includes defining the system's interaction with the environment and stakeholders in addition to defining the system and its boundaries as is done in the ISO 26262 process. The third step, identifying unsafe control actions using the functional architecture and hazards from the previous steps, is similar to a partial merge of the hazard analysis and safety requirement elicitation steps in ISO 26262. We mention partial merge here since identifying hazards (possible accidents) from hazard analysis step and identifying safety requirements from safety analysis step of

ISO 26262 is not a part of identifying unsafe control actions step in STPA. These two parts from hazard and safety analysis comes in first and last step of STPA respectively.

After third step in STPA, Mallya et al.'s augmentation [P40] adds the risk assessment from ISO 26262. The final step is similar to the last step from ISO 26262 (safety analysis). It identifies safety requirements based on the unsafe control actions (and their corresponding risk level in Mallya et al.'s augmentation [P40]).

System Theoretic Process Analysis (STPA)

STPA originates from system theory rather than control theory on which ISO 26262 is based. STPA promises inclusion of requirements beyond systematic and hardware failures (which is the scope of ISO 26262) including inadequate component interactions and human errors.

⑦ *ISO 21448*: The standard ISO 21448 [6] alternatively known as Safety of Intended Functionality (SOTIF), is introduced to complement ISO 26262 in contexts *"where proper situational awareness is essential to safety and where such situational awareness is derived from complex sensors and processing algorithms"* [6]. The SOTIF process focuses on hazards that do not exist because of failures, but instead because of insufficiency of specification, performance limitations, insufficient situational awareness, incorrect and inadequate Human-Machine Interface design, impact from active infrastructure and vehicle to vehicle communication, and external systems [6]. Note that attacks exploiting vehicle security vulnerabilities, and intentional actions that violate the system's intended use, like a substitute hand to fool an "hands-on wheel" detection safety measure, is out of the scope of SOTIF. Also, the standard is in its development stage, and a complete version is yet to be released to the public.

The SOTIF safety requirement elicitation process consists of five steps, as shown in Figure 2.4. The first step, specification and design, is analogous to the item definition step from ISO 26262. This step consists of defining the operating environment like road surface and climatic conditions, the system's interactions with users and traffic participants, and the system and its boundaries. The second step, SOTIF-related hazard identification, is analogous to the hazard analysis step from ISO 26262 but scoped to intended functionality rather than failure scenarios. The third step, SOTIF-related risk evaluation, is analogous to risk assessment from ISO 26262 and is based on three aspects: occurrence frequency of a given scenario, the severity of a triggering condition (same as in ISO 26262), and the effectiveness of measure taken for the safety of the intended functionality. The fourth step identifies insufficiencies of specification, performance limitations, and conditions that trigger the limitations that could initiate hazardous behavior. The final step is deriving safety requirements to avoid or prevent hazardous behavior.

ISO 21448 – Safety of Intended Functionality (SOTIF)

ISO 21448 complements the ISO 26262 process for automated driving and include requirements relating to insufficiencies of specification, performance limitations, insufficient situational awareness, incorrect and inadequate Human-Machine Interface design, impact from active infrastructure and vehicle to vehicle communication, and external systems.

2.3.2 Analysis

We analyze the processes presented in the prior section based on their temporal adoption trend and context of usage. Based on this information and the comparison presented in the preceding section, we present research gaps and upcoming domain trends.

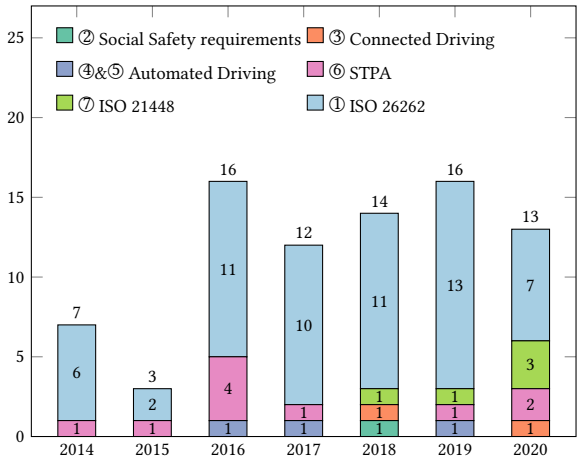


Figure 2.5: Number of primary studies that discuss safety requirement elicitation processes plotted across years

A process adoption's temporal trend is an indicator of its use, in the scope of peer-reviewed literature. A positive trend points to tooling and community support. Both of these are essential for the application of the process in large-scale systems. The adoption of processes across years does not indicate any apparent overall patterns as shown in Figure 2.5. However, the number of publications that mention the usage of ISO 26262's process peaked in 2019 with a substantial decrease in 2020. The number of publications that report usage of STPA first peaked in 2016, followed by a downfall and a small increase again.

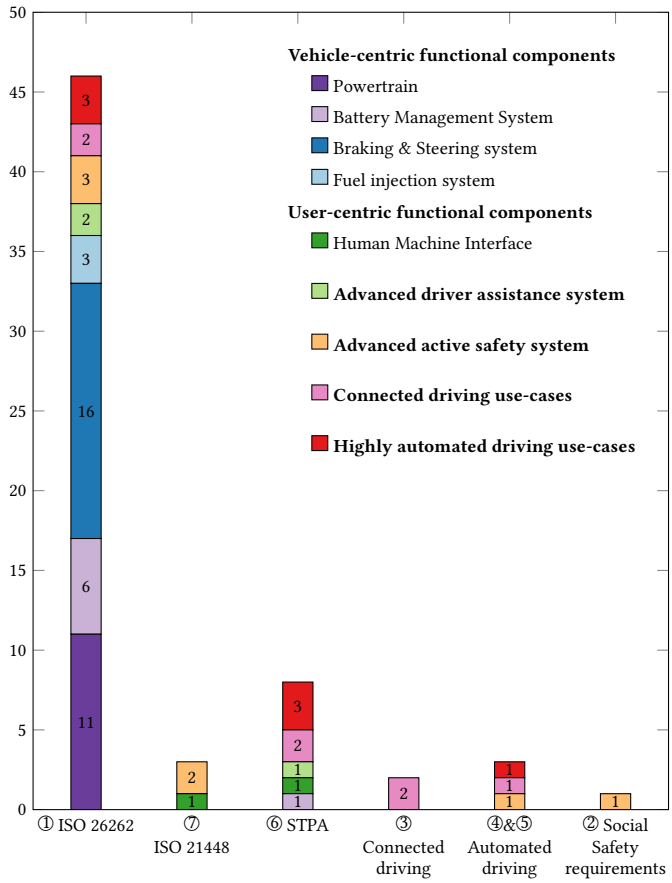


Figure 2.6: Number of primary studies presenting the usage of different safety requirement elicitation processes

Adopting the rest of the processes is reported primarily in the papers that describe them or in papers by the same authors. The only exception is one paper by Stolte et al. [P66] which reports usage of Warg et al.'s [P77] process (described in ⑤). In summary, the application of processes proposed by industry-specific standards dominates across years, while the other processes proposed to extend or replace the standards do not show wide adoption except for STPA. Note that the adoption estimates are based on the primary studies and the real industry adoption might be different. Since industrial usage might not always be apparent from papers published, it will have to be substantiated via future research, for instance, using interviews or surveys.

High adoption of industry standard processes

ISO 26262's safety requirement elicitation process dominates in adoption across years. Other processes proposed to extend or replace the ISO 26262's process do not show wide adoption with the exception of STPA.

The practicality and limitations of processes in different use cases will only be known by the application of the process in those use cases. Therefore, prior application contexts are important for researchers, practitioners, educators, and beginners to identify avenues for future research, the starting point for similar use cases, and to act as a guide for the application of processes. For this, we first group the components and use cases where the processes have been applied into the following six categories based on existing classification schemes in literature [12, 40].
(1) *Vehicle-centric functional components* are the components necessary for the functioning of the vehicle. Examples are powertrain, battery management system, braking and steering system (including their safety systems such as anti-lock braking system), and fuel injection system.
(2) *User-centric functional components*, which in our context is the Human-Machine Interface (HMI), for interacting with the vehicle.
(3) *Advanced driver assistance systems* like adaptive cruise control, traffic jam assist, and lane management system, which can assist a human driver in a driving task but cannot accomplish a complete driving task on their own.
(4) *Advanced active safety systems* like advanced emergency braking system, collision warning and avoidance systems, and maneuver assistance system that use multiple sensors and sensor fusion techniques to actively ensure the safety of traffic participants.
(5) *Highly automated driving use cases* where the vehicle is capable of handling a complete use-case without driver intervention like automated valet parking, automated shuttles that can operate without a human driver on specific routes, and completely automated driving that can drive ideally anywhere or at least in a geofenced area without the active intervention of a human driver.
(6) *Connected driving use cases* like platooning where multiple vehicles form a vehicle train with only the lead vehicle driven by a human and the rest autonomously following the vehicle in front.
The number of articles that report safety requirement elicitation in each of the above categories and the processes they use is plotted in Figure 2.6. The ISO 26262's process is used in primary studies for every category except for User-centric functional components (HMI); STPA and ISO 21448 processes and the processes proposed for automated driving have been reported to be used in more complex use cases (except in one study for battery management systems) and HMI. HMI is not seen traditionally as a safety-critical component;

however, with the advent of advanced driver assistance systems, and highly automated and connected driving, HMI is becoming safety-critical.

Essential versus Advanced functions

The ISO 26262's process has shown to be usable for safety requirement elicitation of essential automotive functions that do not require complex situational awareness.

Informed by the adoption, temporal trend, and comparison of processes, we point to the following three dimensions that need further exploration. One, components and use cases to which the current processes can be applied, but have not been presented in the literature. Two, new contexts and emerging technologies like neural processing units and Machine Learning (ML) based software, which might require new processes for safety requirement elicitation. Three, the introduction of new systems outside the vehicle like intelligent traffic infrastructure and use of the cloud for (assisting) automated driving. The scope and criticality of these external systems will have to be quantified to identify (1) whether they need a separate or integrated requirement elicitation and (2) whether the existing process can be used. Next, we elaborate on each of these aspects.

The literature lacks case studies on categories like advanced active safety systems like blind-spot detection and advanced driver assistance systems like adaptive cruise control, as evident from Figure 2.5. These features have matured from research and development to production and are standard features in today's high-end cars. Upcoming technologies currently in research and development, like highly automated and connected driving, also lack case studies compared to other categories. The adoption of processes other than the industry-standard processes is still low. The lower adoption makes the iterative improvements and scope of their applicability harder to identify and hinders further research. The majority of existing case studies on advanced systems for automated and connected driving are rudimentary examples. The practicality of new processes presented within the scope of these advanced systems is yet needed to be shown in practice. To summarize, new processes proposed in primary studies need more case studies in their respective context for conclusive verdicts on the feasibility of their application in real-world.

Case studies

There are few to no case studies that use requirement elicitation processes other than ISO 26262 or on advanced components and use cases.

All the advanced driver-assist and safety features as well as automated and connected driving are enabled by special-purpose hardware and software that utilizes ML. Yet neither of the two standards ISO 26262 and ISO 21448 mentions [41] any requirements or processes

regarding developing neural networks for usage in such systems. The basis and underlying model on which the requirement elicitation processes were built belonged to an era before ML based software were mainstream. In ML based systems, the system's behavior is dictated by the input data and not by human written logic. The newly proposed processes build on the basic safety requirement elicitation framework proposed by ISO 26262 and mainly focus on the increased complexity, non-existence of driver (controllability aspect), sufficient specification of (safety) requirements, and foreseeable misuses in the context of automated and connected driving, rather than on the above mentioned fundamental changes.

ML based sub-systems

Whether the current processes are sufficient to elicit safety requirements for special purpose hardware and ML based software is an open question.

Connected driving builds on collective traffic optimization by a set of vehicles communicating with each other and external entities (roadside infrastructure) using peer-to-peer networking or via the cloud. With the intelligent traffic infrastructure being a part of a driving task and the cloud acting as a data intermediary or a sensor, both of them become safety-critical components. However, the new methods proposed to elicit safety requirements for the connected driving context focus on the vehicles and do not consider the safety requirements for the infrastructure involved.

Smart infrastructure and intermediaries

How to elicit safety requirements for traffic infrastructure and communication intermediaries like the cloud in the context of connected driving is another open question.

The landscape shifts in the automotive industry with automated driving will also impact safety requirement elicitation. Like the smartphone replaced three separate devices for music, taking photos, and communication; automated driving bundles advanced tasks together. Therefore, many of the categories specified in Figure 2.6 might be a single one in the future. This changes prior assumptions on driver fallback and turns many non-safety critical systems into safety-critical. For example, maps, previously a non-safety critical part, used at human discretion, have an active role in automated driving and are sometimes considered as a sensor. Previously, the driver was assumed to be vigilant and ready all the time for fallback. Now, HMI can be used to decide on a driving task and for emergency takeover, making HMI a critical safety component. The impact of such changes on safety requirements and their elicitation is yet to be seen.

Blurring lines

No more separation of advanced driver assistance and safety systems; these all are bundled together in highly automated driving.

2.4 Safety requirement elicitation techniques (RQ2)

In this section we identify, review, taxonomize, and compare techniques (also referred to as methods [5, 37] [P20, P46, P78]) for safety requirement elicitation in the automotive domain. Techniques or methods refer to the alternative ways to conduct each high-level step [6, 21, 37] [P40, P46, P59, P77] in safety requirement elicitation processes as presented in Section 2.3. For example, two widely used techniques to conduct the safety analysis step in the ISO 26262 process (the last step in the red highlighted part of Figure 2.4) [5] are fault tree analysis [42] (FTA) and failure mode effect analysis [43] (FMEA).

We organize the rest of this section into two parts. Section 2.4.1 present a summary and taxonomies of techniques for safety requirement elicitation. Section 2.4.2 analyzes the techniques, their scope, the associated research gaps, and the upcoming domain trends.

2.4.1 Findings

There are 38 distinct safety requirement elicitation techniques discussed in the primary studies. Our taxonomy is inspired by stages in automotive product development and the reference product life cycle from industry standards [5, 6]. In automotive product development, the norm is to design and evaluate at the system level, then subsystem level, followed by the design and development of the individual hardware and software components. We follow the same structure for creating our taxonomy. Our taxonomy is organized based on the level of application (system, software, and hardware), the usage context (general, automated driving, and connected driving), and the scope of application of the techniques, as shown in Tables 2.2 to 2.4. This allows easy identification of the choice of techniques by both researchers and practitioners. We organize the entire taxonomy of techniques based on the steps in safety requirement elicitation processes (since the techniques are alternate ways to conduct these steps).

To group techniques based on the safety requirement elicitation steps, we must adhere to the definitions of what each step is intended to achieve in the safety requirement elicitation processes. However, in the literature, there is little consensus on this subject. For example, Vilela et al. [21] use the term *safety analysis* to denote any technique used in safety requirement elicitation in the larger context of safety-critical systems; while Kolln et al. [P33], and Abdulkhaleq et al. [P89] use both the terms hazard analysis and safety analysis to compare the same set of techniques. Since this study focuses on the

automotive domain, we follow the terminology from the industry standards ISO 26262 and ISO 21448. These standards have industry consensus and wide adoption (as is evident from the discussion in Section 2.3.2 above).

For consistency and brevity, we group techniques for similar steps of safety requirement elicitation processes (shown with the same color in Figure 2.4) together and refer to each group with the terms (for steps) from the ISO 26262 process. However, in our taxonomy, we refer back to the original processes (refer to the fourth column of Tables 2.2 to 2.4 that shows our taxonomy). Note that the social safety requirement elicitation step [P20], shown in the white box in Figure 2.4, is not similar to any of the four steps from the ISO 26262 process. We did not find any technique for this step in primary studies including the original study itself [P20]. Therefore, we do not consider this step for our taxonomy. Also, our taxonomy does not have the first step of the ISO 26262 process (item definition) nor similar steps in other processes. This step (formally or informally) defines the system, its boundaries, the environment it operates in, the traffic participants it interacts with, and the stakeholders. Such a definition uses languages for modeling, which is a topic in itself for another study and beyond the scope of this work. Therefore, we group techniques based on their usage for each of the other three steps (and corresponding similar steps in other processes shown in Figure 2.4).

We present the techniques in the following three parts (corresponding to the three steps in ISO 26262 process): ***Hazard analysis***, ***Risk assessment***, and ***Safety analysis***. Tables 2.2 to 2.4 show our taxonomy, one for each of the three steps. The techniques are presented in the third column of each table. The primary studies that mention these techniques along with the corresponding process (identified from each primary study) are presented in the fourth column. The first two columns present two dimensions of our taxonomy, and a third dimension is specified in the caption of each table. Note that we only considered explicitly mentioned techniques. For example, Wang et al. [P75] uses the method of brainstorming but does not mention it explicitly; thus, the study is not considered for the technique. In cases where a method is used for more than one step, it is presented in all steps.

Techniques

There are 38 distinct techniques used in primary studies for safety requirement elicitation. We group them into three categories: hazard analysis, risk assessment, and safety analysis. We taxonomize techniques in each category based on the application level, usage context, and scope, as shown in Tables 2.2 to 2.4.

Table 2.2: Hazard analysis techniques from primary studies. All the above techniques are reported in the corresponding primary studies with the scope of identifying/deriving hazardous events or situations and thus deriving safety goals.

Level of application	Usage context		Technique	Process [Primary studies]
System & Software	General	1	Guide-word based brainstorming	ISO 21448 & STPA [P98]
		2	Failure Mode and Effect Analysis (FMEA)	ISO 26262 [P84, P85]
		3	Undesired Combination State Templates	No process specified [P1]
		4	Hazard analysis using vehicle level simulator & item functional model	ISO 26262 & ISO 21448 [P62]
	Automated Driving	5	Situation/Scenario Analysis & Brainstorming	ISO 26262 [P96] ISO 21448 [P34]
		6	guide words based, HAZard and OPerability analysis (HAZOP)-inspired brainstorming	ISO 21448 [P34]
		7	Iterative hazard analysis and function refinement	Process for automated driving [P77]
		8	Shared and multi-level hazard analysis	ISO 26262 [P47]
	Connected Driving	9	Guide-word based brainstorming	Process for connected driving [P32], No process specified [P97]
Hardware specific	General	10	Brainstorming	ISO 26262 [P68]
		11	HAZOP	ISO 26262 [P15, P27]
		12	FMEA	ISO 26262 [P68, P76]
	Automated Driving	13	HAZOP extension to automated driving	ISO 26262 [P5]
		14	HAZOP based on sensor limitations to identify malfunctions	ISO 21448 [P42]

Hazard analysis

The objective of hazard analysis (and similar steps) is to identify (a) (minimal set of) hazardous events either caused by a system's malfunctioning behavior or related to the system's intended functionality, (b) unsafe control actions, (c) possible accidents, and (d) hazards that can cause the accidents. These can then be used to derive safety goals or to

refine the system's functionality (see Section 2.3.1 for more details). We taxonomize hazard analysis techniques as shown in Table 2.2 and discuss each of the techniques starting with the simplest one.

Brainstorming (5 and 10 in Table 2.2) is a technique to identify hazardous events and eventually safety goals for hardware, system, and software specific requirements [P34, P68, P96]. The idea is to first think of possible scenarios and situations that can lead to potential hazards; and then use these to identify safety goals to avoid harm in those hazardous situations.

Guide-word based brainstorming (1 and 9 in Table 2.2) is a more structured, systematic form of brainstorming. It uses guide-words to explore the hazardous situation space [P32, P34, P96–P98]. Guide-words are predefined words like *No* and *Late*, which, when combined with scenarios, can thus be used to identify potentially hazardous situations. This structured method is shown to be applied in general [P98] and in automated [P34, P96] and connected driving [P32, P97] contexts.

HAZard and OPerability analysis or *HAZOP* (11 in Table 2.2) is the technique from which the usage of guide-words stems. In its original form, HAZOP is a structured and systematic method with a specific documentation style. HAZOP also recommends a specific team composition (to conduct the analysis) with certain roles inside the team [44]. HAZOP is reportedly used in its traditional form at the hardware level [P15, P27].

Three *extensions to HAZOP* (6, 13, and 14 in Table 2.2) were proposed in the context of automated driving [P5, P34, P42]. Martin et al. [P42] showed that HAZOP, based on limitations of sensors like cameras and LiDARs, can be used to identify hazardous situations related to the safety of intended functionality (14 in Table 2.2). Another extension of HAZOP uses a skill graph as a functional model of the system with scenarios for automated driving to find hazardous events [P5] (13 in Table 2.2). Kramer et al. [P34] uses a HAZOP-inspired brainstorming approach using guide-words for identifying hazards in the context of automated driving (5 in Table 2.2). They also use it in the context of safety of intended functionality, focusing on environment triggers that can cause hazards.

Failure Mode and Effect Analysis or FMEA (2 and 12 in Table 2.2) is another traditional and systematic technique [43]. FMEA is a bottom-up method, starting from the failure of a component(s) and then identifying the potentially hazardous situations it can lead to. FMEA is reported to be used for hazard analysis at system, software, and hardware level [P68, P76, P84, P85].

Undesired combination state templates (3 in Table 2.2) is a technique proposed by Aceituna et al. [P1] for identifying hazards which can be caused by a combination of system components' (and environmental) states. They argue that traditional hazard analysis techniques like FMEA focus on hazards relating to the state of (failure of) one component or event. The proposed method identifies hazards relating to multiple (failure) events while reducing the number of state permutations needed in combinatorial approaches. The technique uses templates to identify the combination of events/states (rather than a single

event/state in FMEA), which can lead to a potential hazard. Thus, it forms a complementary method to the traditional hazard analysis techniques.

Iterative hazard analysis and function refinement (7 in Table 2.2) is a technique proposed by Warg et al. [P77] that can help reach completeness of safety goals for the completely automated driving functionality. They argue that in completely automated driving settings, the traditional methods used for hazard analysis are insufficient to ensure safety goals' completeness. They suggest that hazard analysis should have a broader scope to ensure that a vehicle's function fulfills its specifications for completely automated operation. Their technique uses an iterative procedure using trees of hazards and operational situations that can help reach a more complete and minimal set of safety goals than other techniques. They use hazard analysis as an aid rather than an afterthought, when defining the scope of vehicular functions. On the contrary, other techniques define the functions before hazard analysis.

Shared and multi-level hazard analysis (8 in Table 2.2) is a technique proposed based on the idea of shared responsibility in automated driving. The hazard analysis techniques discussed above take only the vehicle (and its driver) responsible for the potential hazards and associated safety goals. Monkhouse et al. [P47] suggest that automated driving comes with shared responsibility between multiple traffic participants to avoid hazardous situations. The study recommends performing hazard analysis considering the division of responsibilities, and handling hazard analysis at various levels, for instance, one level for each participant.

Hazard analysis using vehicle level simulator and item functional model (4 in Table 2.2) is proposed to increase the accuracy and reliability of hazard analysis techniques. Most methods mentioned above (except FMEA) do not use a detailed system model for hazard analysis. Sini et al. [P62] and Tao et al. [P68] use simulators to model system and its environment (item functional model) to aid hazard analysis [P62, P68].

Hazard analysis techniques

here are 12 distinct techniques for hazard analysis and similar steps discussed across 17 primary studies. Brainstorming forms the basis for a majority (1, 5, 6, 9, 10, 11, 13, and 14) in Table 2.2) of techniques. Some techniques (3, 4, 7, 8, 13, and 14 in Table 2.2) are proposed by the primary studies while the rest of them use existing techniques in the automotive context. Most of the techniques are reported in the context of the safety requirement elicitation processes of industry standards ISO 26262 and ISO 21448.

Table 2.3: Risk assessment techniques from primary studies. Except for the quantitative risk norm (fourth row in the third column), the rest of the methods are used in general context. The quantitative risk norm is proposed and used specifically to replace the ASIL levels in the context of automated and connected driving.

Level of application		Scope	Technique	Process [Primary studies]
System & Software	1	Identify/allocate Automotive Safety Integrity Level (ASIL) for systemwide safety goals	Usage of Controllability, Exposure, and Severity metrics from ISO 26262	ISO 26262 [P30, P32, P39, P41] [P62, P72, P76, P82, P85, P96] STPA [P6, P40, P94]
			Augmenting the ISO 26262 metrics with plant and fault modeling	ISO 26262 [P84]
			Identifying severity ratings using simulation	No process specified [P17]
			Quantitative Risk Norm to replace ASIL ratings for automated & connected driving	ISO 26262 [P9, P78]
	2	ASIL decomposition	High to low ASIL decomposition according to ISO 26262 guidelines	ISO 26262 [P51]
			Merging ASIL allocation method from ISO 26262 with ASIL decomposition	ISO 26262 [P37]
	3	Optimal ASIL allocation out of possible allocation options	Genetic Algorithm with cost heuristics	ISO 26262 [P3]
			Tabu Search with cost heuristics	ISO 26262 [P3, P102]
			Penguin Search	No process specified [P93]
	4	Reducing ASIL allocation search space	Cut-set based reduction	ISO 26262 [P23]
			Heuristic cost based reduction	ISO 26262 [P23]
			Ant colony optimization algorithm	No process specified [P24]
Hardware specific	5	Identify/allocate ASIL for functional component (hardware) specific safety goals	Usage of Controllability, Exposure, and Severity metrics from ISO 26262	ISO 26262 [P15, P27, P35, P36, P38, P60, P63, P68, P69, P76]
	6	ASIL decomposition	Dependent Failure Analysis	ISO 26262 [P83]
	7	Optimal ASIL allocation	Genetic Algorithm with cost heuristics	ISO 26262 [P3]

Risk assessment

Risk assessment intends to allocate scores for safety goals such that the score indicates the urgency and reliability level needed to address the safety goal. Table 2.3 shows a taxonomy of techniques for risk assessment derived from the primary studies. Unlike hazard assessment techniques, the risk assessment techniques have different scopes of application, as listed in the second column of Table 2.3. We elaborate on each group of techniques based on their scope of application.

Identification of ASILs (1 and 5 in Table 2.3) is the problem of identifying the risk level of a safety goal. ASIL is the risk scoring scheme from the ISO 26262 standard (see Section 2.3.1 under ① for more details). The standard also provides a metric that combines the three individual ratings to identify the ASIL level.

These metrics are used by a majority of the studies [P6, P15, P27, P30, P32, P35, P36, P38–P41, P60, P62, P63, P68, P69, P72, P76, P76, P82, P85, P94, P96] (first row of 1 and 5 in Table 2.3). These studies use the metrics with the three assumed individual ratings derived from operational scenarios.

All the studies use the metrics for their case studies except Khastgir et al. [P30]. They explore how to improve inter-rater reliability of risk assessment while using the metrics in ISO 26262. They look at the two following settings: one, a rerun of the same techniques by different teams using the same data; two, with no restrictions on techniques and data (i.e. not necessarily the same methods and data) by different teams but with the same analysis scope and objectives [45]. They propose a rule set for risk assessment to improve the reliability of risk assessment (the severity and controllability ratings) in the automotive context. They conclude that subjective interpretation and resulting unreliability and variation could be reduced using an exhaustive and explanatory rule-set [P30].

Augmenting the ISO 26262 metrics with plant[7] *and fault modeling* (second row of 1 in Table 2.3) is another method to reduce subjectivity of the ISO 26262 ASIL determination. The method stems from control theory and is proposed by Zhang et al. [P84]. They propose the use of mathematical modeling of the system and faults. Such modeling enables quantitative analysis via simulation. This can provide sound reasoning for exposure and controllability ratings, and evidence for the ratings can be provided by fault simulations.

Identify severity ratings using simulation (third row of 1 in Table 2.3), the usage of formal models and physics based simulations for risk assessment. This contrasts with the abovementioned studies, where severity level is computed based on various assumptions. Duracz et al. propose a method that allows computing severity levels for specific operational scenarios with accurate bounds on all the modeled parameters like the pre-collision and post-collision velocities, which contribute to a hazardous event [P17].

[7] Plant (in control theory) refers to a machine or system. Plant modeling refers to specifying the system as a relation between output and input signals. Plant and fault modeling refers to modeling both the system and probable faults.

Quantitative Risk Norm or QRN to replace ASIL ratings for automated and connected driving (fourth row of 1 in Table 2.3) is a risk assessment scoring system proposed by Warg et al. [P78]. They suggest that the automated driving systems' safe behavior results from a combination of tactical and operational decisions. Therefore, safety guarantees can be achieved by adjusting the proactive decision-making, in addition to addressing random and systematic failures. These two kinds of failures are independent of the traffic situations and are the only kinds of failures the ASIL ratings takes into account [P78]. QRN is proposed to substitute the fixed risk assessment criteria of ISO 26262. Warg et al. [P78] suggests to define what is regarded 'sufficiently safe' at design time. This definition is to used to identify discrete risk levels called consequence classes. Each consequence class receives a total norm frequency informing how often, at most, this kind of consequence is allowed to occur. The consequence classes along with their norm frequency is QRNs. QRNs can be used to classify incidents into a set of incident types. Now each safety goal will be associated with one incident type. Warg et al. [P78] demonstrate the applicability of QRN for automated driving while Bergenhem et al. [P9] show its applicability for connected driving.

ASIL decomposition (2 and 6 in Table 2.3) is the process of decomposing a higher ASIL rating of a functional component by implementing the functional component with redundancy. Here, each redundant component will have a lower ASIL rating than the original functional component, while combined, the functionality they achieve will still have a higher ASIL rating. The idea is that redundant components with individually higher probability of failure will have a lower overall failure rate. Readers can refer to Park et al. [P51] (first row in 2 Table 2.3) for an example implementation of the guideline.

Merging ASIL allocation method from ISO 26262 with ASIL decomposition (second row of 2 in Table 2.3) is proposed by Lidstrom et al. [P37]. They argue that the allocation and decomposition of ASILs should not be separate as specified in ISO 26262 and instead should be merged into one. They point out that separating the two processes applies only to systems with a specific kind of redundancy where two or more safety mechanisms check some state and take action sequentially. Such separation is not applicable in cases of redundancy where only one of the redundant components is operating and the other is on standby. They propose a method to combine the allocation and decomposition of ASILs. They apply it in the context of an actuation system for automated driving.

Dependent Failure Analysis or DFA (6 in Table 2.3) is a part of the ASIL decomposition where a higher ASIL rating is splited to lower ASILs among a set of components. DFA [46] is performed to ensure that a single root cause cannot lead to the failure of all the lower ASIL components: whether the ASIL rating, when split into multiple components, does not lead to dependent failures among the components. Young et al. [P83] propose a new scoring system for root causes of dependent failures. This scoring system can be used to compare failures' root causes, which can aid ASIL decomposition. They claim that their scoring system is more exhaustive and compelling than IEC 61508 – ISO 26262's predecessor.

Optimal ASIL allocation (3 and 7 in Table 2.3) problem arises when there are multiple ways to allocate ASIL ratings to individual components which together perform a function (forming a functional component). According to Azevedo et al., the ASIL allocation problem is a complex optimization problem with a vast search space of possible ASIL allocations to individual components [P3]. They present a genetic algorithm and tabu search algorithm with cost heuristics to find a strategy that minimizes the total cost of development and production while meeting the desired ASIL level with the least effort [P3]. Following Azevedo et al.'s work, Sorokos et al. [P102] also showed the application of tabu search to ASIL, allocation in the context of a braking system. Another work in the direction of optimal ASIL allocation is using the penguin search algorithm by Gheraibia et al. [P93]. They claim that the penguin search algorithm can produce optimal or near-optimal results within the least amount of time and resources for the computation. Detailing these search algorithms is beyond the scope of this chapter.

Reducing ASIL allocation search space (4 in Table 2.3) is important since, in any practical case, the search space for finding an optimal ASIL allocation that meets the safety and cost requirements has a huge solution space due to the combinatorial nature of the problem [P23]. Searching through this space may become impracticable in large and complex systems [P23]. Therefore Gheraibia et al. [P23] propose two solutions that can be sequentially performed to reduce the solution space: (1) Cut-set based reduction where cut sets[8] with different orders are formed as trees with their roots and nodes as the cut sets and leaves as basic events. The idea is to limit the possible ASIL range for a basic event (leaf node) and thus reduce the search space by reducing the possible ASIL allocations; (2) Heuristic cost-based reduction[9] which reduces solution space by grouping the ASIL allocations to equivalence classes and creating a priority list of the allocations in each equivalence class. Gheraibia et al. [P24] further builds on their earlier work [P23] and adds an ant colony optimization algorithm for further solution space reduction.

Note that techniques like FMEA and its augmentation [P11] are used to assess whether a component or system adheres to a specific ASIL level (*ASIL evaluation*). This is out of our scope since it does not belong to eliciting safety requirements but instead assesses the fulfillment of safety requirements.

[8] "A cut-set is a minimal combination of failures of components, which, if they occur in conjunction, lead to a hazard" and fault tree analysis is used to find cut sets. [P23]

[9] "Cost heuristics are functions that determine the cost of associating the ASILs to each component of the system" [P23]

Risk assessment techniques

There are thirteen distinct techniques for risk assessment and similar steps discussed across 34 primary studies. These are techniques to (1) identify/allocate risk scores to safety goals; (2) decompose risk scores to multiple components; (3) identify optimal allocation of risk scores; or (4) reduce risk score allocation space. Most studies discuss or use techniques belonging to the first category. The only process specified for risk assessment in all the primary studies is ISO 26262's process.

Safety analysis

Safety analysis is the process of deriving functional safety requirements from safety goals and allocating them to the individual components, system, software, or hardware architecture.[10] A taxonomy of techniques used for safety analysis in the primary studies is presented in Table 2.4. Now we summarize each method in the order of predominance of usage and simplicity.

Fault Tree Analysis or FTA (1 , 5 , 8 and 10 in Table 2.4) is a top-down, tree based, safety analysis technique that starts from a safety goal and leads to safety requirements and their allocation to architecture components [42]. The safety goal (or a potential hazard) is taken as the root node or top event of a tree made of logic gates as intermediate nodes. The safety requirements form the leaf nodes of the tree. The tree is constructed top-down from the root to the leaves. The fault tree can be qualitative (without any labels on edges connecting the nodes) or quantitative (with the edges labelled with failure probabilities). FTA has been applied for all levels and all usage contexts that we considered in this chapter [P10, P14, P15, P27, P32, P35, P36, P38, P45, P48, P58, P60, P82, P86].

FTA with fault classification (9 in Table 2.4) is proposed by Dajsuren et al. [P13] for identifying the relative contributions of different groups of faults (fault classes) to the safety goals. They use fault classification—a key-value structure indicating the frequency of different faults—rather than failure probabilities to label the fault tree starting with the leaf nodes. They use this method to identify the percentage of total potential failures caused by vehicle-to-vehicle communication faults in connected driving.

Dynamic Fault Trees or DFTs (2 in Table 2.4) are proposed by Ghadhab et al. [P92] to augment fault trees for faithful representation of vehicle system model. They suggest that the traditional fault trees are not sufficiently expressive for faithful representation of vehicle system models. DFTs extend fault trees with the following four specific gates: *sequence-enforcers*, for restricting sequence between children of a node; *priority-and*, for indicating priority between children of a node; *spare-gates*, for supporting reduced or zero

[10] In some processes, the intermediate safety goal step is skipped, and safety requirements are directly derived from hazardous events

Table 2.4: Safety analysis techniques from primary studies. All the above techniques are reported in the corresponding primary studies with the scope of identifying/deriving safety requirements from safety goals using architecture of the system or the specific component.

Level of application	Usage context		Technique	Process [Primary studies]
System & Software	General	1	Fault Tree Analysis (FTA)	No process specified [P10]
		2	Dynamic fault trees	No process specified [P92]
		3	Component integrated component fault trees	No process specified [P16]
		4	Model Based Safety Analysis (MBSA)	ISO 26262 [P48, P84] No process specified [P53]
	Automated driving	5	FTA	ISO 26262 [P58]
		6	Environment fault tree	ISO 21448 [P34]
		7	MBSA augmented with simulation	No process specified [P70]
	Connected driving	8	FTA	Process for connected driving [P32, P86]
		9	FTA with fault classification	ISO 26262 [P13]
Hardware specific	General	10	FTA	ISO 26262 [P14, P15, P27, P35, P36, P38, P45, P48, P60, P82]
		11	Common-cause fault analysis	ISO 26262 [P18, P27]
		12	Failure Mode and Effect Analysis (FMEA)	ISO 26262 [P27, P35, P48] No process specified [P53]
		13	Aging FMEA	ISO 26262 [P57]
		14	Failure Mode Effect and Diagnostic Analysis (FMEDA)	ISO 26262 [P12, P28, P45, P50, P95]
		15	FMEDA augmented with Simulation	ISO 26262 [P61]
		16	Dependent failure analysis	ISO 26262 [P50]

failure rate; and *functional dependencies* supporting modeling feedback loops and triggers. They show the applicability of DFTs in the case study of a vehicle guidance system.

Environment Fault Tree or EFT(6 in Table 2.4) proposed by Kramer et al. [P34] extends fault trees to specify environmental conditions using special gates. In EFT, environmental conditions are modeled as leaf nodes that trigger higher-level faults. EFTs classify the causes for deviation from correct behavior to (random) hardware faults, (systemic) design faults in hardware or software, (systemic) specification faults either due to incorrect assumptions or lacking a structural approach. This method is specified in the context of safety of intended functionality in automated driving.

Common Cause Fault analysis or CCF (11 in Table 2.4) is a safety analysis technique that uses fault trees to identify faults caused by the same set of causes or conditions. Such common cause faults can be fatal in case of redundancy, especially in cases where a higher risk level is addressed by using redundant components rated with lower risk levels. Here a common cause fault can lead to the failure of all redundant units at once, potentially leading to a sub-system or system-wide failure. Frigerio et al. [P18] suggest that such faults should be avoided across individual components contributing to redundancy. An application of CCF is presented by Huang et al. [P27] in the context of the steer-by-wire system's hardware.

Failure Mode and Effect Analysis or FMEA (12 in Table 2.4), in contrast to FTA, is a bottom-up safety analysis technique [43]. The analysis starts with the possible malfunction or failure of individual components. It works backward to identify the effects of the failure in the system and which safety goals they (failure of a component) violate. The potential failure modes are typically derived from experience with similar products and processes. In our primary studies, FMEA is reported in the contexts of the hardware part of steer-by-wire system [P27], brake-by-wire system [P35], and part of ignition system [P53]. Further optimizations for streamlining FMEA are proposed in [P48].

Aging FMEA (13 in Table 2.4) tailor fits FMEA to focus on aging effects for circuits in automotive. Aging FMEA proposed by Scharfenberg et al. [P57] analyze the electrical properties' change due to aging and identify aging-dependent critical hardware components that can lead to a potential hazard or safety goal violation. The method is an adaptation of FMEA assisted with simulation. They show the feasibility of the method using a fuel injection system case study.

Failure Mode Effects and Diagnostic Analysis or FMEDA (14 in Table 2.4) builds on FMEA [47] with adding three aspects to each failure mode that affects safety goals: (1) failure rate or the rate at which the component experiences faults; (2) whether there is a safety mechanism to detect the failure mode or probability to detect internal failures; and (3) the effectiveness of the safety mechanism at detecting faults. The end product of this analysis consists of the hardware parts related to failures and metrics that show the level of safety readiness. In primary studies, FMEDA is applied on hardware of anti-lock braking system [P45, P50], system-on-chip [P12], FPGA [P28], and powertrain electronics [P95].

FMEDA augmented with simulation (15 in Table 2.4) is presented by Sini et al. [P61] to help designers, especially in cases where the system's behavior is highly coupled with the vehicle's behavior. They simulate the system/component and generate possible failures or misbehaviors using fault injection and then propagate these misbehaviors to the vehicle level using a vehicle dynamics simulator. They use it for failure effect classification by taking the predicted effects on dynamics and drivability of the vehicle.

Component Integrated Component Fault Trees or C^2FTs (3 in Table 2.4) are proposed as a combination of FTA and FMEA [P16]. The resulting tree structure's root nodes represent safety goal violations or hazards of a system, leaf nodes represent basic failure modes, and the intermediate nodes present the relation between failure modes and hazards with Boolean gates. Domis et al. [P16] use C^2FT in the context of product lines.

Dependent Failure Analysis or DFA (16 in Table 2.4) focus on safety goal violation due to possible common cause(s) and cascading failures between elements [46]. DFA aims to identify the common causes that can violate required independence or freedom from interference between elements and, in turn, causes a safety goal violation. Nardi et al. [P50] mentions the usage of this method in the case study of an anti-lock braking system.

Model based safety analysis or MBSA (4 and 7 in Table 2.4) is an umbrella term that is used to denote any safety analysis that uses a (formal) system model created ideally using a model-based development process, extended with a fault model [48]. Using a system model in the safety analysis can minimize subjectivity and be more complete, consistent, and error-free than using an informal system model or no model. The underlying safety analysis technique(s) can be any of the above-discussed techniques. MBSA, in conjunction with FTA and FMEA, is reported in primary studies [P48, P53, P84]. In addition, a study by Tlig et al. [P70] extends MBSA with simulation for safety analysis of automated driving systems.

Safety analysis techniques

There are thirteen distinct techniques for safety analysis and similar steps discussed across 28 primary studies. These techniques build on two base techniques: (1) the top-down Fault-Tree Analysis or FTA; and (2) the bottom-up Failure Mode and Effect Analysis or FMEA. FMEA and its extensions are predominantly reported in hardware-specific contexts, while FTA and its extensions dominate usage in systems and software. While ISO 26262's process dominates the underlying process for which these techniques are used, a good number of primary studies (five out of 28) do not specify any process.

2.4.2 Analysis

We analyze the techniques presented in the prior section regarding their scope and context of usage. Based on this information and the comparison presented in the preceding section, we present research gaps and upcoming domain trends.

Each step in safety requirement elicitation can be conducted using multiple kinds of techniques. Especially for beginners, it is important to understand which techniques can be employed in a given context; for educators, which techniques to teach. Our study can aid in these directions. We find that no one technique fits all levels of an application or all use cases for any step in the safety requirement elicitation. For any real-life use case, we believe it is best to use a combination of techniques to identify safety requirements. Each technique has its strengths and drawbacks. For instance, FTA can easily be applied for the safety analysis at system, software, and hardware levels; however, it might not be able to find dependent failures. To the best of our knowledge, no cheat sheet lists the strengths and drawbacks of individual techniques. Our taxonomy is a mere first step in this direction. However, an in-depth comparison of the techniques for their use in the automotive context is out of the scope of this study and is an essential future research direction.

No silver bullets

No one technique fits all application contexts; it is best to use a combination of techniques to identify safety requirements.

The repeatability of safety requirement elicitation techniques is a key factor in ensuring safety from a safety engineering and requirements engineering perspective. For instance, one element in the safety certification in almost all domains is based on an assessment by an independent team by repeating or assessing the safety cases. Such activities require objective safety requirement elicitation. The primary studies show that most methods rely on expert knowledge, rendering them subjective. Surprisingly there is little effort to quantify the subjectivity. We found only one study on the subjectivity of a technique, which focused on a specific risk-assessment technique [P30]. Even though methods like model-based safety analysis are proposed to increase reliability and repeatability of the safety analysis, (1) their adoption rate is low (only one of the primary studies uses model-based safety analysis in their case study), and (2) there are fewer such methods for hazard analysis and risk assessment.

Repeatability

Most methods rely on expert knowledge rendering them subjective and thus hampering repeatability.

Informed by our taxonomy of techniques presented in the prior section, we foresee four aspects that need further exploration: (1) comparison among techniques, (2) completeness and coverage of safety requirements, (3) lack of techniques to support steps of newer processes, and (4) whether the current techniques are a match for the new application contexts arising along with the automated and connected driving. We elaborate on each of these aspects below.

Comparison among techniques is necessary to identify the most suited techniques for use by both practitioners and researchers. There is little empirical evidence on which method to choose among the available options. Current studies only compare the techniques FTA, FMEA, and STPA, where STPA is considered as a technique rather than a process [P33, P89]. Existing studies argue that STPA is better than FTA and FMEA. However, no studies specify the scope of the techniques which could allow practitioners to make an informed decision on which technique to choose for a specific use case. Also, most of the studies that compare the methods are on toy case studies, which might not represent methods' real-world efficacy.

Comparison

There is a lack of studies that systematically compare the requirement elicitation techniques.

Ensuring the completeness and coverage of safety requirements is essential for the rest of the development process. Failing to ensure this can lead to high costs, impact the timeline of product development, and potentially catastrophic consequences during operation [49, 50]. However, we did not find any studies that look in this direction for any of the specified techniques.

Completeness & coverage

Completeness and coverage of safety requirements, especially in the context of automated and connected driving, are seldom explored.

Any process for safety requirements is ineffective unless there are systematic techniques to support or perform the individual steps in the process. Even though the steps in newer processes like STPA are similar to the steps from more established processes like the one from ISO 26262, the way to conduct them and their intended outcomes are either different or have different scopes. Thus, the techniques to conduct the steps in traditional techniques do not apply to the newer processes, and we did not find any other systematic techniques in the primary studies to conduct the individual steps. For example, no systematic method

to perform the individual steps of STPA is used in any of the primary studies that apply STPA [P27,P40,P42,P87,P91,P98]. Rather these studies use informal guidance and previous examples to conduct STPA. The only exception is a technique for hazard analysis, namely iterative hazard analysis (7 in Table 2.2), which is specified to conduct a step similar to hazard analysis in a new process for automated driving (⑤ in Figure 2.4).

New processes versus old techniques

The newer processes, especially based on fundamentally different approaches, lack systematic techniques to support their steps.

One primary enabler for current innovations in automated and connected driving is the use of ML based sub-systems, especially for perception and planning [12]. Most of the techniques mentioned in this section were developed in and for an era prior to the development of these technologies. The special-purpose hardware like Neural Processing Units uses a different style of instruction execution than traditional processing units [51]. The applicability and adequacy of current techniques, which might be built on the assumptions for traditional general-purpose hardware, need further studies. It is also yet to be seen whether the current techniques can be applied in the context of safety requirements for developing neural networks, which are increasingly used in perception and planning subsystems of automated driving stacks.

ML based systems

The adequacy of current techniques to elicit safety requirements for special purpose hardware and ML based software is yet to be seen.

2.5 Implications

We presented a taxonomy and comparison between different processes. One use-case of this work is as a cheat sheet or guidebook for practitioners and educators. This work outlines what exists in the peer-reviewed literature in almost every technical dimension of automotive safety requirement elicitation.

This study has broad implications from research to practice. We present the implications in the rest of this section in the following five parts:

- *current state* of safety requirement elicitation;
- a trend of creating *islands of knowledge*;
- the *changing landscape* of the automotive domain;
- the lessons that can be learned and reused *beyond automotive software engineering*; and
- *education.*

Current state

We discuss two aspects of the current state of safety requirement elicitation research: *maturity* of the research field and *transparency & replicability* of the primary studies.

Maturity: The safety requirement elicitation for older technologies has matured, while the newer concepts need further research. Four ways to empirically measure the maturity of a research area are (1) author divergence [52], where a diverse set of authors indicate a mature research field; (2) prevalence of case studies [53] where the bulk of case studies point to maturity; (3) relation between academic work and what is applied in the field [52, 54] where more evidence of industry/practitioner participation form an indicator of maturity; and (4) convergence of best practices [54] where a majority of studies showing adaptation of a set of similar practices indicate maturity. Based on these four parameters, we classify the safety requirement elicitation into two contexts: for traditional components and the newer age concepts and components.

We define traditional components as the components in production for more than a decade. Examples are the steering system and most items belonging to vehicle-centric functional components, as shown in Figure 2.6. We define newer age concepts as those that have not entered or are currently entering the production stage and components that enable the implementation of the concepts. Example concepts are highly automated driving and example components are ML based perception systems that enable highly automated driving.

The safety requirement elicitation relating to the former context (traditional components) is more mature than the latter based on the above mentioned four metrics. The evidence are high author divergence, a wide range of case studies, and increased industry participation based on the (meta) data from 62 papers that explicitly mentions case study on specific components. Regarding a wider range of case studies, case study on traditional components form 44 papers [P2,P3,P7,P14,P15,P26,P27,P31,P35,P36,P38–P41,P44–P46,P49–P51, P54, P55, P57, P60–P64, P67–P69, P71, P72, P74, P76, P77, P79, P82, P84, P85, P87, P88, P98, P99] while 19 mentions case study on new age components [P5, P6, P9, P13, P21, P30, P32, P37, P42, P46, P59, P65, P66, P86, P90, P91, P94, P96, P101] with one study [P46] belonging to

both categories since it mentions more than one case study. Concerning author divergence, out of the 19 papers on new age components, up to four papers have common authors [P5, P65, P66] with at least two more similar clusters of papers with overlapping authors [P13, P32, P86, P90, P91]. In the 44 case studies on traditional components, most sets of authors for the papers are disjoint, with the size of the largest cluster of papers with the same authors being three [P61–P63]. With respect to industry participation, the papers with all authors affiliated with the industry form 5% of the newer age components while 16% for the traditional components.

For the fourth metric, the convergence of best practices, we have two angles: processes and techniques. In the context of processes, we can see convergence to ISO 26262 (see Figure 2.6). In the context of techniques, we can see convergence to brain-storming and related techniques for hazard analysis, usage of metrics from ISO 26262 for risk assessment, and FTA and FMEA-related techniques for safety analysis (see the last column of Tables 2.2 to 2.4).

Transparency & replicability: A majority of the primary studies do not specify details on techniques that the studies employ, hampering transparency and replicability. In the case of hazard analysis, many studies do not specify which technique they use for hazard analysis; instead, they directly present the results of hazard analysis. For risk assessment, the assumptions on coming up with a specific value for exposure, controllability, and severity are often not specified. For safety analysis, the intermediate results are often not presented, making it hard to understand and replicate the final result. For future studies, we recommend reporting the techniques, intermediate results from these techniques, associated assumptions, and their scope.

Islands of knowledge

From our 102 primary studies, we noticed a systematic trend of creating islands of knowledge inside companies where the detailed knowledge on safety requirement elicitation stays inside the companies and are not available via peer reviewed publications. We present two related aspects below.

Sharing specifications, intermediate results, and related data of research: Continuing our prior discussion on *transparency & replicability,* the majority of the studies, except for a few (e.g., [P32]), do not share the details on case studies. For example, an operational design domain (ODD) definition is essential for making an informed judgment on any result of hazard analysis of a system. However, most studies do not provide details but only discuss the final result. The assumptions (e.g., exclude snowing conditions) during the process and the intermediate steps (e.g., fault trees; hazardous events) are essential to judge the results produced in the papers and, most importantly, to build on for future studies. In its current format, this *"unknown details"* makes it difficult for newer researchers and other related disciplinarians to enter this field. The sharing of related data on research should be the

norm as in other software engineering fields like mining software repositories.

Sharing real-case studies: Given the existence of myriad vehicle types with various capabilities, it is safe to assume that considerable safety requirement elicitation has been performed for their development. Yet, to our knowledge, no real-world, industry-scale case studies have been published on this topic in peer-reviewed literature. So far, publications in collaboration with industries either use toy case studies (e.g., [P1]) or very high-level abstractions (e.g., [P86]) hampering any real-world reproducibility. The publication of real-world case studies can help researchers and the automotive community to identify current issues facing the industry and contribute to rectifying and proposing methods and techniques, rather than creating islands of knowledge inside companies. This can reduce the work to re-invent the knowledge and benefit the companies to get better talent and suggestions from academia. Additionally, with the higher reliance on software and related components for automated and connected driving, openness to independent safety verification/certification (in contrast to self-certification) should be made a norm.

Changing landscape

Is safety requirement elicitation catching up? The last decade marks arguably the most significant paradigm shift in the automotive industry since its inception. Four dimensions of this shift are the transition from internal combustion to electrification, automated and connected driving, ongoing shift to open source software development, and start-ups entering the field and finding success bringing disruptive ideas and new business models. This means profound changes in almost all dimensions of automotive software and the electronics that run them. Whether the safety requirements elicitation is catching up is still an open question. There is relatively more research on transition to electrification (especially on battery and power-train) as evident from Figure 2.6. The open-sourcing trend is still unfolding [12] and its safety requirement elicitation side might be too early to research. Automated and connected driving is achieved by combining special purpose hardware, ML based software, and traditional software to make them work seamlessly. We discuss three dimensions of this topic further: (a) functional safety and (b) safety of intended functionality in the context of ML based components, and (c) connected driving-specific issues. All the above aspects are particularly challenging to safety certification bodies across the globe.

ML based components & functional safety. Both software and hardware components specific to ML (and mainly neural networks) function fundamentally different from traditional components. For example, the software components are built from data rather than human logic. The hardware components are based on multi-threaded execution, parallelism, and many levels of optimizations compared to non-parallel execution units otherwise used. Given the ML based components are now becoming a part of safety-critical applications, we need (functional) safety requirement elicitation methods that take the peculiar nature of

these components into account, which is not the case currently. This might need thinking with a different perspective altogether than the traditional kinds.

When we look at the case studies (on functional safety), the vast majority use traditional processes and techniques. Even though newer methods and techniques for automated and connected driving are proposed, there is little evidence for their applicability and suitability in the real world. Also, the sufficiency of current processes and techniques to address ML based systems is another daunting question for industries and safety certification bodies. Multiple processes and techniques might be applicable to ML based systems. While our study presents a comparison among processes, no study on in-depth comparison of techniques, in the context of the automotive domain. To summarize, there are many potential directions for research in the ML based component's context, including case studies on newer processes and techniques and in-depth comparison among existing techniques for their suitability, sufficiency, and scope.

ML based components & safety of intended functionality. Safety of intended functionality, especially in the context of removing the human operator fallback, is out of the scope of traditional processes and techniques. Even though methods like STPA is developed to cover some aspects, its real-world adaption is low, and so far, the scope of STPA is at the system level. The standard, ISO 21448, is proposed to tackle the safety of intended functionality. It is still in its beginning stage and gives only conceptual directions than concrete guidelines. Safety of intended functionality in both hardware and software levels still needs processes and techniques that can be applied to special purpose hardware like neural processing units. The processes and techniques could be entirely new, extension of current ones, or studies show that existing techniques are applicable in such settings.

Connected driving specific issues. There can be three kinds of communications connected driving: vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-road users (V2R). The first two kinds are more established than the third kind. As we mentioned in Section 2.3, there is a lack of methods that integrates V2I along with the traditional automotive safety requirement elicitation. There are at least two challenges here, (1) safety requirement elicitation for communication intermediaries like cloud for connected driving[11] and (2) requirements spanning across multiple vendors (e.g., the manufacturers for smart traffic infrastructure and vehicles might be different). Further research is needed in these two directions.

While vulnerable road users form about a third of fatalities in road accidents in the USA[12], safety requirement elicitation for vehicle-to-road user communication is not mentioned in any of the primary studies. It is still an open question how to complement the information on road users (cyclists and pedestrians) beyond vehicle sensors (like camera

[11] Connected driving refers to a set of vehicles and traffic infrastructure communicate various parameters to optimize the collective traffic behavior. This communication is often achieved by using intermediaries such as cloud.

[12] `https://crashstats.nhtsa.dot.gov/Api/Public/Publication/813178`

and LiDAR) and remove their blind spots. Safety requirement elicitation regarding the same is also in its nascent stages and needs further research.

Multi-disciplinarity: beyond automotive software engineering

This chapter explored the technical dimension of safety requirement elicitation. However, there are other dimensions, e.g., combining safety requirement elicitation with aspects like security, tool support, and the human and social factors of safety requirement elicitation. These are some directions the future research can explore for a comprehensive understanding of safety requirement elicitation.

While this chapter considered a system and software view of safety requirement elicitation, the area is primarily multi-disciplinary, as shown in Figure 2.3c. Also, there are similar domains like advanced robotics (e.g., robots from Boston Dynamics[13]), which are multi-disciplinary, with similar challenges in perception systems and unsupervised deployment (in the future). It might be interesting to see how these domains compare and what can be learned from each discipline.

Education

The next generation of engineers who will have to build more of these new systems is being educated today. In the context of software engineering, most curricula do not focus on safety, let alone the safety of the newer types of systems. Since software systems are becoming more safety-critical, this trend needs to change, and safety has to be incorporated as a topic. We believe it is equally important to educate on the caveats and limitations of current techniques since it is crucial to understand the scope of existing processes and techniques for their correct usage in real-life. Also, in typical software engineering curricula, a systems view is seldom included but is crucial in the context of safety requirements. We strongly encourage including this in the curricula for educating future software engineers.

From a practitioner's perspective, it might be difficult to have a definitive idea on which processes and techniques to use in different contexts, especially for ML based systems. While this chapter can be used as a cheat-sheet, there is still room for an in-depth comparison of the techniques. Some directions are: (a) the scope and possible application contexts of techniques; and (b) objective criteria on when to and not to choose a technique.

2.6 Threats to validity

This section presents the threats inherent in our study despite our best attempts at mitigating them during design, implementation, and interpretation. Below we use the framework by Cook and Campbell [55] to discuss the threats to validity.

[13] `https://www.bostondynamics.com/`

Construct validity: For a systematic literature review to be useful, the articles analyzed should be representative, presenting an overview. To identify representative scientific articles, we employed multiple methods, including:

1. PICO method (see Section 2.1.2) to create the search string;
2. both forward and backward snowballing on the primary studies to improve the coverage of relevant research articles and to make the process as insensitive to the choice of the search string and databases as possible;
3. documentation of our search process, inclusion-exclusion criteria, and information extracted from the selected papers for transparency and replicability of our research; and
4. independently assessing the intermediate steps as authors, and resolving disagreements with discussions.

Still, we might have missed relevant articles, for instance, non-peer-reviewed articles.

Conclusion validity: In comprehending the scientific literature for presenting insights, we foresee two threats: one, characteristics of the available literature, and two, researcher bias in interpretation. For the first part, we have limited control over the availability of scientific literature. For instance, we observed that case studies on real-life vehicles could have added deeper insights for investigation. Unfortunately, such studies are limited. To mitigate researcher bias in interpretation, we followed the definitions from the literature and did not make assumptions or choices if the text was not clear. Also, we explicitly defined the criteria we followed to ensure objectivity.

Internal validity: As indicated above, our findings are susceptible to publication bias since we only analyze published scientific literature. Our choices ensure that our analysis rests on systematic scientific findings but implies that non-peer-reviewed articles (e.g., negative results) are systematically removed. It is possible that a study including other sources of information can present a different picture than the one depicted here. Also, our study inherits the same threats for quality assessment as in Kitchenham et al. [24], Tiwari et al. [28], and Wieringa et al. [29] as we adopted the same methods.

External validity: The representativeness of our study hinges on the domains we considered for our analysis and the representativeness of the studies within these domains. We search for relevant articles on a broader range: using multiple databases and applying techniques such as snowballing. But our investigation is also limited by our choices to study articles written in English, non-grey literature, and published in reputed venues. All of the above factors influence our findings' generalizability and define the scope within which our results apply.

2.7 Related work

We classify the secondary studies that cover safety requirements during product development into three categories: (*i*) studies applicable to multiple domains, (*ii*) studies on safety assurance, and (*iii*) studies that focus on the automotive domain. Since our scope is limited to safety, we do not discuss secondary studies on the intersection of safety and security. This section discusses relevant secondary studies and discludes primary studies as this chapter is a secondary study.

2.7.1 Cross domain studies

Pereira et al. conducted an SLR on requirements to be considered, and on requirement engineering practices and challenges in developing embedded systems [20]. They summarize open issues that can be potential research directions, which include: *"Requirements Specification for automotive system"; "Improve the development process for ensure functional safety requirements"; "Handling of non-functional requirements such as ···, safety,···"; "Specification of safety requirements";* and *"Analysis of hazard and threats, ···, and safety"*.

Martins et al. present an SLR on approaches to elicit, model, specify and validate safety requirements in the context of safety-critical systems and usage of these approaches in industrial settings [19]. This work forms a precursor to our study, as detailed in Section 2.1.2.

Vilela et al. performed an SLR to explore methods to improve the integration between RE and safety engineering [21]. They identified techniques and tools used for hazard analysis and safety analysis (to be) used by requirement engineering and safety engineering teams. They also generated a taxonomy of these techniques along with a separate taxonomy of the safety information used or generated by these techniques. This study is complementary to our focus.

Bozhinoski et al. [22] presents a systematic mapping study on managing safety in mobile robotic systems from a software engineering perspective. Their primary studies on safety management in the context of self-driving cars conclude that *"self-driving vehicles lack a standardized platform, processes, and tools for designing and analyzing safety approaches"* [22]. However, they did not consider safety standards from the automotive domain.

Another direction pursued in literature is functional safety in product line engineering. Baumgart et al. conducted a systematic mapping study to summarize topics covered and evaluation approaches used in the literature on the functional safety of product lines [23]. Gadelha Queiroz et al. performed an SLR for identifying approaches, methods, and methodologies from the intersection of the product line engineering and model-driven engineering for safety-critical systems [56]. We consider this direction of research out of scope for this work.

2.7.2 On safety assurance

Safety assurance is about providing evidence that safety is ensured. Safety requirements are a part of safety assurance, and in this SLR, we focus on eliciting functional safety requirements. Nair et al. conducted an SLR towards establishing evidence for system safety [57]. They focused on three aspects: the information that constitutes evidence; structuring of evidence; and evidence assessment. There are also model-based methods based on goal structured notation and conceptual models for establishing evidence for system safety [58–60].

Bolbot et al. examined different sources of complexity introduced to cyber-physical systems and explained different methods for safety assurance in those contexts [61]. This work does not specify methods used in the automotive domain nor uses any systematic method to identify the different techniques in the literature for safety assurance.

Ingibergsson et al. [62] conducted a systematic mapping study on (coding) practices in developing safety-critical software for field robots. Field robots are machinery for outdoor tasks like agriculture. They concluded that most approaches propose solutions to attain safety, focus on behavior modeling, and do not stress reliable software development e.g., involving formal verification. Half of the literature they considered uses non-standardized methods to develop software. They also note an increase in safety-related issues with the introduction of computer vision.

In this work, we concentrate on approaches for generating safety requirements rather than safety assurance. However, conclusions from these studies, for example, the nascence of technologies like computer vision which does not follow traditional development methods, equally, apply in the context of our work.

2.7.3 On safety in the automotive domain

Some secondary studies from the last decade discuss safety and are specific to the automotive domain. A few studies ([63, 64]) outline safety-related methods across the entire product life cycle, while others focus on a specific stage of product development [65], a specific technology [66], or a specific use-case [67]. In addition to the secondary studies, there is a proliferation of experience reports [68, 69]. We do not cover experience reports in our related work since they form a different class of studies than secondary studies.

Studies by Kannan et al. and Gosavi et al. consider the entire product life cycle [63, 64]. Kannan et al. [63] present a gap analysis between the objectives of the ISO 26262 standard and safety-related techniques to achieve these objectives. They conclude that there is a need for tooling for conducting HARA, combining sub-phases of product development, integration among the product development phases of design, requirement management, and validation and verification. They also find a lack of methods for ASIL decomposition for large-scale systems.

Gosavi et al. [64] summarize four primary studies from the product development phase at the system, hardware, and software levels covered by parts 4, 5, and 6 of ISO 26262. They noted the lack of a standard and inadequacy of ISO 26262 to address functional safety in the development of autonomous and semi-autonomous vehicles.

Gheraibia et al. [65] reviewed different approaches for the specific sub-phase of product development, ASIL allocation. ASIL allocation methods find optimal allocations of ASILs (QM, A, B, C, D, with QM being negligible risk and D being highest risk) to system architecture components such that safety requirements are guaranteed to be met at the lowest possible cost. They classify approaches for ASIL allocation into two categories—exact and optimization approaches—and present the pros and cons of each method.

Borg et al. present a short review of validation and verification (V&V) methods for the safety of the specific technology, machine learning, and deep learning methods, to be used in the automotive domain [66]. They classify support for V&V of machine learning and deep learning-based systems as formal methods, control theory, probabilistic methods, process guidelines, and simulated test cases. They conclude that the V&V methods for machine learning and deep learning lag compared to other areas. Also, those V&V methods suggested by ISO 26262 do not apply to developing components that use machine learning and deep learning methods. They stress the need for a new standard for them.

Axelsson [67] presented an SLR and a gap analysis on safety in the specific use-case of platooning. Platooning is a coordinated movement of vehicles as a train with minimal distance between vehicles in the pack. Specifically, they focus on safety analysis methods, hazards and failures, and solutions to improve safety for platooning. The authors also note a lack of studies from commercial settings and current research primarily from research prototypes or technology demonstrators.

Even though many of these studies overlap with our work, we did not find any systematic study on eliciting safety requirements for the automotive domain.

2.8 Conclusions

This work characterizes safety requirement elicitation in the automotive domain and presents a comprehensive overview of the current landscape through a systematic literature review. This chapter is a first step in the direction of summarizing, taxonomizing, and comparing processes and techniques for safety requirement elicitation for the automotive domain.

We identified nine safety requirement elicitation processes and 38 distinct techniques. Out of the nine processes, we observed that the process outlined in the ISO 26262 standard forms the basis for requirements elicitation in the automotive industry. Other processes have been proposed to complement, extend, or replace the process outlined in the standard. This chapter offers an overview, comparison, and taxonomy of such processes and corresponding techniques for safety requirement elicitation in the automotive domain. Based on

this information, temporal adoption trend, usage context, and scope, we presented research gaps and discussed current domain trends.

The current landscape shifts in the automotive industry, such as automated driving, connected driving, and the move to open sourcing, have profound implications for the safety of automotive systems. We empirically showed the immaturity of the safety requirement elicitation concerning newer concepts and components like automated and connected driving. For instance, the majority of processes and techniques for safety requirement elicitation were proposed more than two decades ago. At the same time, the upcoming perception and decision systems for automated driving rely on ML based components whose feasibility of usage in real-world scenarios (e.g., neural networks) was demonstrated only a decade ago. The applicability and scope of the processes and techniques to these new-age components is still an open question. Another example is the lack of studies on safety requirements regarding traffic infrastructure and communication with vulnerable road users, which are critical to accomplishing connected driving. Other dimensions, e.g., combining safety requirement elicitation with aspects like security, tool support, and the human and social factors of safety requirement elicitation, are yet to be explored.

We emphasize the importance of safety requirement elicitation of software and related systems and show the importance of a systems and multi-disciplinary perspective. This work opens up many future avenues for research and provide a concise and comprehensive guide to practitioners and educators.

3

Assessing Safety Requirements for Connected Driving

The scope of automotive functions has grown from a single vehicle as an entity to multiple vehicles working together as an entity, referred to as connected driving. The current automotive safety standard, ISO 26262, is designed for single vehicles. With the increasing number of connected driving capable vehicles on the road, it is now imperative to systematically assess the functional safety of architectures of these vehicles. Many methods are proposed to assess architectures with respect to different quality attributes in the software architecture domain, but to the best of our knowledge, functional safety assessment of automotive architectures is not explored in the literature. We present a method that leverages existing research in software architecture and safety engineering domains, to check whether the functional safety requirements for a connected driving scenario are fulfilled in the technical architecture of a vehicle. We apply our method on an academic prototype for a connected driving scenario, platooning, and discuss our insights.

This chapter is based on:

S. Kochanthara, N. Rood, L. Cleophas, Y. Dajsuren, M. van den Brand. Semi-automatic Architectural Suggestions for the Functional Safety of Cooperative Driving Systems, ICSA'20 (New and Emerging Ideas track) [70];

S. Kochanthara, N. Rood, A. Saberi, L. Cleophas, Y. Dajsuren, M. van den Brand. Semi-automatic Architectural Suggestions for the Functional Safety of Cooperative Driving Systems, JSS'21 [71]; and

S. Kochanthara, N. Rood, L. Cleophas, A. Saberi, Y. Dajsuren, M. van den Brand. Summary: A Functional Safety Assessment Method for Cooperative Automotive Architecture, ECSA'21 [72]

Traffic congestion was estimated to cost 305 billion dollars in 2017 to traffic participants in the United States of America.[1] With continuously increasing urban population [73], traffic congestion will continue to be an inevitable problem for the foreseeable future. Around 70% of all goods transported around the United States are moved by trucks, and the lion's share of the cost for operating trucks comprises fuel costs and driver salary [74]. One potential solution to reduce traffic congestion and operational costs is *connected driving*.

Connected driving refers to the collective optimization of the traffic participants' behavior by sharing information using wireless communication such as a peer-to-peer network or via other actors like the cloud [75]. connected driving can improve traffic efficiency, reduces cost, and increases comfort [76,77]. It is one of the 54 trends shaping the technology market, according to market research.[2] In the year 2020 alone, 10.46 million new vehicles, with some form of connected driving capabilities, are projected to hit the roads.[2][3] With millions of connected driving capable vehicles on roads, the safety of these vehicles needs urgent attention.[4]

A majority of the connected driving functionalities are achieved by determining a vehicle's behavior for optimal traffic behavior according to the information received from other traffic participants. Such optimal behaviors are achieved (partially or fully) using software-controlled steering, acceleration, and braking [78]. Therefore, any problem in the software can lead to catastrophic effects not only to the vehicle itself but also to other traffic participants. To avoid such events, connected driving systems are designed to operate in case of failure or fail safely.

The current guidelines to ensure the safety of automotive systems (and their architecture) are provided by ISO 26262:2018 - a product development standard for the automotive domain [5]. The ISO 26262 standard offers systematic methods from the safety engineering domain to identify safety requirements. Any automotive software architecture that fulfills these safety requirements is deemed safe-by-design.

ISO 26262 standard neither considers connected driving nor prescribes methods for architecture assessment. The standard is designed for single vehicles and does not include a connected perspective in which a set of vehicles is seen as a single entity [79, 80]. This can mean that a low-risk safety requirement from a single-vehicle perspective can have catastrophic effects on other cooperating vehicles [77]. To create a functionally safe architecture from a connected perspective, existing studies have extended the standard guidelines [70, 81] or presented an architecture framework [77]. Yet, checking the safety of software architecture of an existing vehicle for connected driving, remains an open question.

[1] https://www.smartcitiesdive.com/news/gridlock-woes-traffic-congestion-by-the-numbers/519959/

[2] https://go.abiresearch.com/lp-54-technology-trends-to-watch-in-2020

[3] https://www.forbes.com/sites/samabuelsamid/2019/10/28/volkswagen-includes-nxp-v2x-communications-in-8th-gen-golf/

[4] https://www.sciencedaily.com/releases/2019/05/190519191641.htm

ISO 26262 standard does not prescribe methods to assess the *functional safety* of automotive architecture. Many approaches to assess architectures with respect to quality attributes have emerged in the software architecture domain in the past three decades [82–89]. However, only some of these methods are designed for operational quality attributes like performance (in contrast to development quality attributes like maintainability) [82, 90]. To the best of our knowledge, none of these methods are designed to assess the operational quality attribute *functional safety* of automotive systems.

This chapter presents a method to assess the functional safety of existing automotive architecture for connected driving, by combining methods from the safety engineering and software architecture domains. Our method has two parts:
(*i*) derive Functional Safety Requirements (FSRs) for connected driving scenarios—an extension of our earlier work [70];
(*ii*) check whether the (technical) software architecture fulfills the derived functional safety requirements—a combination of techniques [84, 91, 92] adapted from the software architecture domain.

This chapter primarily focuses on the design phase (concept development phase in ISO 26262) and validation of the resultant requirements in the software architecture in the final product. We demonstrate our method on the architecture of an academic prototype capable of connected driving. The connected driving scenario used for demonstration is *platooning*, where a manually driven vehicle is autonomously followed by a train of vehicles.

The rest of the chapter is organized as follows. Section 3.1 presents the background relevant to the study. Section 3.2 describes the proposed method to derive FSRs and check for their fulfillment in vehicles' technical software architecture. Section 3.3 details the application of the proposed approach on an academic prototype for the connected driving use case, platooning, and interpreting the results from this case study. Section 3.4 discusses our implicit assumptions, applicability, and scope of our approach. Section 3.5 outlines related research. Section 3.6 presents threats to validity, followed by future research directions in Section 3.7 and the conclusion in Section 3.8.

3.1 Background

In this section, we discuss the three basic concepts upon which we build the contributions of this chapter. First, we outline the relevant concepts in automotive functional safety. Second, we discuss some basics on safety tactics and patterns. Last, we give a brief introduction to the two views of automotive architecture.

3.1.1 Functional safety

Functional safety is defined as *"an absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems"* [5] where E/E systems refer to electrical and/or electronic systems. In the automotive domain, functional safety is defined by two standards:

ISO 26262:2018 and ISO 21448 [6], serving complementary purposes. The former focuses on the hazards caused by the malfunctioning of components of a system, while the latter does on the hazards resulting from the functional insufficiency and misuse [5, 6]. ISO 26262 [5] is the current safety standard with its latest revision from 2018. In contrast, ISO 21448 [6], is currently available as ISO/PAS 21448 specifications with a formal release planned in 2021. The predecessor of these standards is the broader IEC 61508 standard [38], which outlines the functional safety guidelines for developing electrical/electronic/programmable electronic systems that are used to carry out safety functions [38].

We primarily focus on the concept phase (part 3) of the ISO 26262 standard, which outlines the derivation of FSRs and their allocation to functional architecture components. The concept phase is executed on an item where an item is defined as *"system or combination of systems, to which ISO 26262 is applied, that implements a function or part of a function at the vehicle level"* [5].

The derivation of functional safety requirements (FSRs) begins with creating hazardous events. Each hazardous event is a combination of a hazard, an operational mode, and an operational situation. An example of a hazardous event is a brake failure (hazard) in eco-driving mode (operational mode) while driving on a highway (operational situation). The operational modes and operational situations are derived from natural language descriptions of intended environments or situations where the system operates. This natural language description is referred to as scenario description or scenarios from hereon.

To ensure safety from hazardous events, safety goals are defined. These goals are broad, presenting high-level safety requirements. Each safety goal is allocated a score, termed Automotive Safety Integrity Level (ASIL), of A, B, C, or D, which specify the importance of achieving the goal (A for least important and D for most important) during further stages of product development. The ASILs are calculated based on exposure, controllability, and severity of each safety goal according to the ISO 26262 guidelines [5]. Each safety goal is decomposed into one or more FSRs [5]. Each FSR inherits the (maximum) ASIL from the safety goal(s) it is derived from.

In the literature, there is little consensus on safety requirements being functional or non-functional requirements. FSRs are classified as functional requirements in the safety engineering domain. However, FSRs are predominantly classified as quality requirements (non-functional requirements) in the software architecture domain [85].

3.1.2 Safety tactics and patterns

Architectural tactics encapsulate design decisions that can influence the behavior of a system with respect to a quality attribute [85]. Architectural tactics are abstract, do not impose a particular implementation structure, and can be seen as recommendations without a prescribed implementation. On the other hand, architectural patterns are well-defined structured entities with a prescribed implementation that realize tactics. This chapter

employs safety tactics and patterns [91, 92] which are architectural tactics and patterns to address safety.

3.1.3 Architecture views

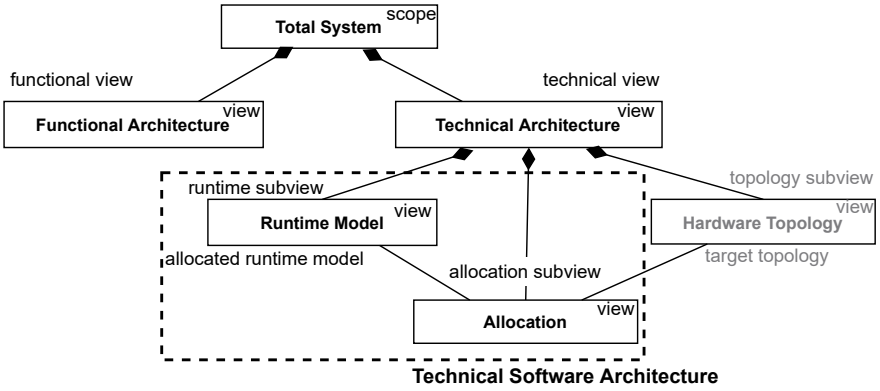


Figure 3.1: Functional and technical architecture views and their scope, adapted from Broy et al [93]. Functional and technical architecture are views of the same system at different architectural abstraction levels, with functional being the highest abstraction level. Runtime model describes system behavior while hardware topology describes the structure of hardware platform containing electronic control units, sensors, mechanical components, and the buses that interconnect them. Allocation associates elements of the runtime model with the elements of hardware topology. Runtime model and allocation together form technical software architecture.

We use the architecture of a system in two contexts: (*i*) to generate FSRs from hazardous events by mapping hazardous events to functional components of the system; (*ii*) to identify whether one or more safety tactics are used for the implementation of a functional component.

The first context needs a functional decomposition view of the system [5], known as functional architecture view [93–96]. In the automotive domain, the functional architecture view outlines functional composition, functional entities, their interfaces, interactions, inter-dependencies, behavior, and constraints in a vehicle [93]. This view is derived from the functional viewpoint, which considers the system from the angle of vehicular functions and their logical interactions from a black-box perspective [93]. Note that the scope of this view is at the system level.

The second context demands more details that are not available in the functional

architecture view but are available in the technical architecture view (also described as the implementation view) [93, 95–97]. The technical architecture view outlines specific software implementation, physical components (like electronic and electrical hardware), their relationships, the allocation of software parts to hardware components, the dependencies among software and hardware components, and constraints [93]. Clearly, there is strong conformity between the technical architecture view and the functional architecture view [93]. A pictorial depiction of these two architectural views is shown in Figure 3.1.

We chose the technical architecture view since it enables identifying whether one or more safety tactics are implemented, and this view is readily available, as it is mandatory in automotive projects [93]. In contrast, other views might lack necessary detail or may be outdated. In the rest of this chapter, we discuss the runtime model and allocation part of the technical architecture view, together termed as technical software architecture.

3.2 Research design

We propose a method that checks whether the technical software architecture of a vehicle fulfills the FSRs for connected driving scenarios. The method consists of two parts: (*i*) derive FSRs for connected driving scenarios (see Section 3.2.1 and Figure 3.2), and (*ii*) check whether the derived FSRs are fulfilled in the technical software architecture of a vehicle (see Section 3.2.2 and Figure 3.3).

FSRs for connected driving shall be implemented in individual vehicles. The ISO 26262 standard recommends mapping of FSRs (or breaking down FSRs) to individual system architecture components [5]. Further, such a mapping is crucial given the complexity and scale of the system. Referring to the existing solutions from the safety engineering discipline [98], the current methods do not map derived FSRs for connected driving scenarios to individual vehicle components [70]. Our solution bridges this gap by integrating a connected functional architecture (with its individual components belonging to the vehicular functional architecture) with the existing methods to derive FSRs. This step is presented in detail in Section 3.2.1.

Next, we check whether the derived FSRs are fulfilled in the technical software architecture of a vehicle. Our method of assessing the fulfillment of derived FSRs is a combination of techniques adapted primarily from the software architecture domain. With no existing architecture assessment techniques addressing the quality attribute of functional safety in the context of automotive systems, the proposed method takes inspiration from traditional architecture assessment techniques like ATAM [84, 85] and employs the safety tactic framework [91, 92, 99] to leverage existing architecture knowledge. This part of our method is presented in Section 3.2.2.

Alongside functional safety, cyber-security is another area that is increasingly addressed together with functional safety [100]. The scope of our approach is limited to functional safety and security is out of our scope. Moreover, FSRs are often fulfilled dedicatedly in

hardware or a combination of hardware and software. Even though the first part of our method associates FSRs to architecture components at the system level, the second part of our approach focuses on software. FSRs fulfilled in hardware architecture (hardware topology in Figure 3.1) is beyond the scope of our method.

3.2.1 Derive FSRs for connected driving

The deriving FSRs part of our method needs only a black box view of individual vehicle functions and interactions among these functions. Therefore, we use the functional architecture view for deriving FSRs for connected driving. Note that functional architecture is the overall system architecture (see Figure 3.1), which includes both hardware and software components.

We extend the traditional method outlined by the ISO 26262 standard [5] to derive FSRs for connected driving scenarios. The traditional approach (the concept phase of ISO 26262) is executed on an individual vehicle as the item. We propose a similar approach to be executed on the entire connected system in parallel. Figure 3.2 presents an overview of the proposed method. We first outline the traditional method, followed by our prior work on its extension [70] and our new contribution. For the rest of the chapter, we use the term *vehicular perspective* for an individual vehicle as a unit under consideration and *connected perspective* for a set of vehicles as a unit under consideration.

Traditionally, FSRs for a vehicular perspective are derived by mapping the safety goals for a vehicle on to the individual components of the vehicle's functional architecture. This process of mapping, also termed safety analysis, captures information on the malfunctioning of a component that can lead to violation of a safety goal. Safety analysis is performed using a systematic process like fault tree analysis (FTA) [42] or failure mode effect analysis (FMEA) [43]. To conduct safety analysis, we need two inputs: (*i*) the functional architecture that captures a system's decomposition into functional components and the interconnection between these components, and (*ii*) safety goals.

According to ISO 26262 guidelines [5], safety goals are derived from hazardous events. Hazardous events are found by decomposing the scenario description using the hazard analysis and risk assessment technique (HARA) [5]. This method to derive FSRs is depicted by part 3 and flows *a* and *d* of Figure 3.2, with *a* and *d* acting as inputs to safety analysis. This method of deriving FSRs from scenario descriptions has been standard practice in the automotive domain [5] for at least a decade [101].

During safety goal derivation using HARA, each safety goal is assigned an ASIL level. The ASIL level is allocated based on the severity of the damage possible by the hazardous event, and the probability of exposure and controllability of the vehicle during the event, according to the metric provided by ISO 26262 [5]. Each FSR inherits the highest ASIL of the safety goal(s) it is derived from. An FSR with ASIL 'D' indicates that the most stringent safety measures must be applied to meet the FSR. In contrast, ASIL 'A' indicates a lower risk and lower level of safety measures.

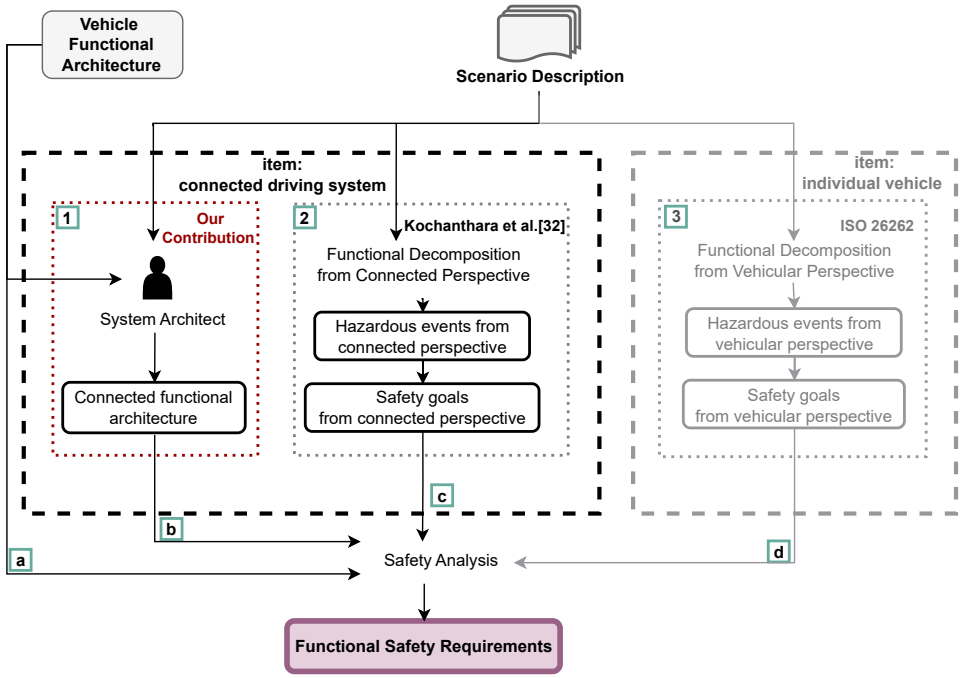


Figure 3.2: Method to derive FSRs for connected driving scenarios. The gray part on the right is the traditional method from ISO 26262 [5], and the black part on the left is our addition to the traditional approach. System architect represents external entities involved in creating the connected architecture.

In a connected system, a safety goal for one vehicle can lead to an FSR in another vehicle. For example, consider a simple connected driving scenario of one vehicle (*follower*) autonomously following another manually-driven vehicle (*leader*) using vehicle-to-vehicle communication for coordination. A safety goal in this setting is: *"the follower shall autonomously accelerate in accordance with the acceleration of the leader."* Even though the safety goal seems to belong to the autonomously accelerating component of the *follower*, it also maps to the functional architecture component(s) of the *leader*. This safety goal leads to the following FSR on the acceleration sensing component of the *leader*: *"failure in the acceleration sensing component of leader shall not communicate incorrect acceleration information to the automatic steering component of the follower"*. Failing to meet this requirement (and its associated safety goal) can potentially lead to a crash. Such safety goals, however, will only be visible in the connected perspective.

In the proposed method, we have one item per individual vehicle type, and an item for the entire connected system of which the vehicles are part. A connected system can have more than one type of vehicle (for example, two vehicles with different functional architectures forming a connected system) and other entities like a cloud, enabling connected driving capabilities. In the case of more than one type of vehicle (with different functional architectures), each kind of vehicle will form an item. For each item, except for a connected system, the traditional ISO 26262 analysis described above is applicable. We believe that two items, as shown in Figure 3.2 will generalize to other scenarios that require more than two items. Such cases only add replication of traditional ISO 26262 analysis (see shaded part in figure 3.2) for each additional item (i.e., each unique functional architecture). In any case, there will only be one connected functional architecture and thus only a connected item. For the rest of this section, we consider two items: an individual vehicle (representative of all vehicle functional architectures) and the connected system.

We propose that FSRs for a connected system are derived from: (*i*) safety goals from the vehicular perspective (as in the traditional method), and (*ii*) safety goals from the connected perspective. Along these lines, our prior work [70] extended the traditional process to derive safety goals for the vehicular perspective to the connected perspective (annotated as part 2 in Figure 3.2) to cover FSRs from both perspectives. This process partitions the scenario description into vehicle-specific and cooperation-specific parts. Next, we apply the traditional safety goal identification steps to the two parts. FSRs from the vehicular perspective are then derived, as discussed above.

We observed that the connected functional architecture should be built using individual vehicle functional components. This will preserve the mapping between functional architecture of connected system and its implementation view (in the technical architecture of the vehicles). A connected functional architecture is required for safety analysis techniques like FTA [42] to derive FSRs, by mapping safety goals to components of functional architecture. We propose that the connected functional architecture be built from (*i*) the functional architecture of individual vehicles that constitute the connected system and (*ii*) the connected scenario description of the interaction between individual vehicles. With these requirements, system architects can create a functional architecture of the connected system such that the individual components of the architecture are mapped onto the components of the functional architecture of vehicles. This process is labeled as part 1 in Figure 3.2; the complete process of deriving FSRs from the connected perspective is shown by the labels 1, 2, *b*, and *c*.

In summary, the presented method maps each individual connected driving scenario to a set of FSRs, where each FSR is associated with at least an individual vehicle function, which in turn is associated with a functional component. Note that a one-to-one mapping is suggested for the efficiency of method and is not mandatory. Mapping an FSR to multiple functional components is unwise for two reasons: (1) the responsibility is not clear, therefore implementation may go wrong; and (2) testing may not be feasible at that level and only

integration testing can assess the achievement of that FSR. In the rest of the chapter, we assume that each FSR can be mapped to a functional architecture component.

3.2.2 Check fulfillment of FSRs

Our method to check for the fulfillment of FSRs in the technical software architecture of individual vehicles is organized in two phases. Phase one ensures that it is possible to realize all the FSRs by identifying whether there are conflicting FSRs. Phase two describes a systematic method to check for the fulfillment of FSRs in the technical architecture. Figure 3.3 depicts an outline of the process.

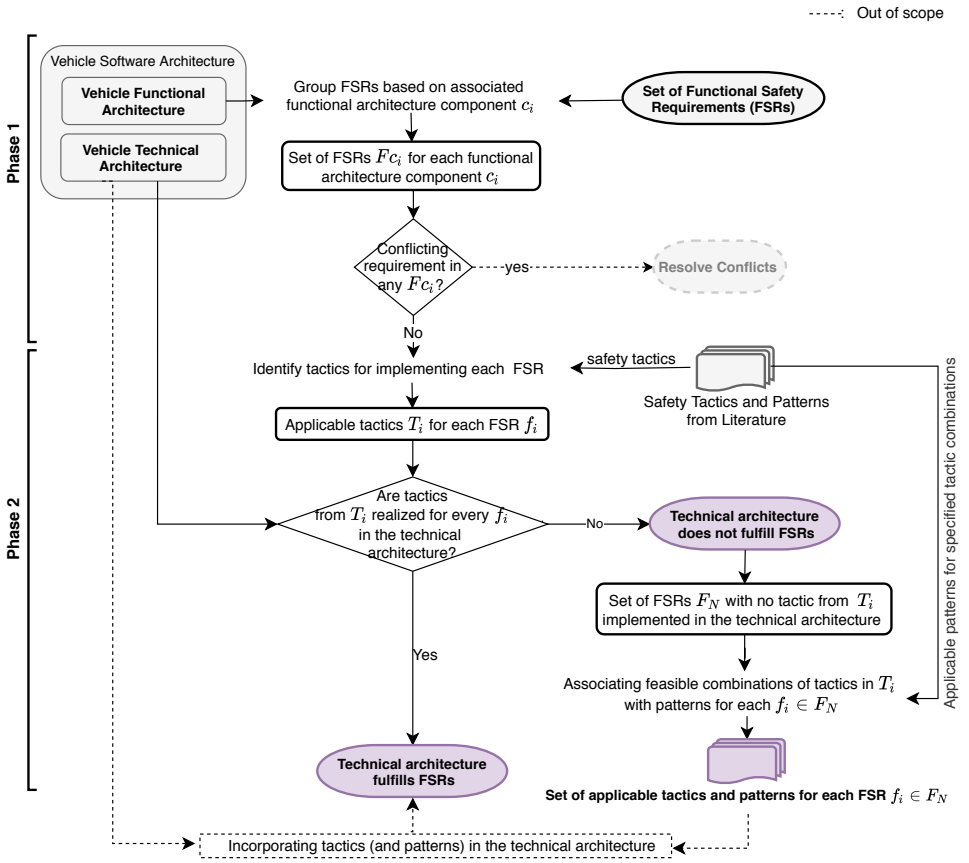


Figure 3.3: Method to check the fulfillment of FSRs in technical architecture

Our method uses both functional and technical views. The functional view is used for a sanity check among FSRs for conflicts. The technical view, in contrast, is used for checking the implementation of each vehicular function (and its associated safety mechanisms) against the corresponding FSRs.

In phase one, we check for conflicting FSRs. Two FSRs are conflicting if both of them cannot be fulfilled at the same time. A hypothetical example of conflicting FSRs is:
FSR_01: A failure in the actuation sensor shall be indicated by a fault message from the sensor.
FSR_02: A failure in the actuation sensor shall cease any further messages from the sensor.
FSR_01 and *FSR_02* are conflicting requirements: sending a message for *FSR_01* and not sending any message for *FSR_02* for the same event (failure in the actuation sensor), which cannot be realized simultaneously.

Comparing every pair of FSRs for conflicts will lead to a quadratic number of comparisons (if n is the number of FSRs, the number of comparisons is $n(n-1)/2 \approx O(n^2)$). We compared FSRs that belong to the same functional architecture component for conflicts. This can reduce the number of comparisons up to a factor of d, where d is the number of functional components (i.e., the number of comparisons can be reduced up to $n(n-d)/d \approx \Omega(n^2/d)$). Such a reduction is possible since safety analysis techniques for deriving FSRs ensure that each FSR belongs to only one functional component [5, 42, 102]. Further, FSRs belonging to a component can have conflicts among themselves but not with the FSRs belonging to other components. For example, in our case study in Section 3.3, we derived 31 FSRs across 8 functional components. Comparing every pair of FSRs would result in 465 comparisons; however, grouping FSRs based on functional components reduced it to 60. This process is annotated as Phase 1 in Figure 3.3.

The presence of conflicting requirements points to flaw(s) in any of the following: (*i*) the functional architecture, (*ii*) functional decomposition of the scenario, or (*iii*) the scenario itself. This is based on the assumption that the rest of the steps are carried out without mistakes. These conflicts need resolution before proceeding. While resolving such conflicts is beyond the scope of this work, checking for these conflicts provides a sanity check that it is possible to meet all FSRs in a given technical architecture.

An FSR may be fulfilled by a safety tactic or a combination of safety tactics. To identify whether an FSR is fulfilled, we propose checking the vehicle technical software architecture for the implementation of safety tactics [91,92] that can meet the FSR. This is achieved in two steps: (*i*) identify a set of safety tactics (hereafter referred to as *applicable safety tactics*) such that the implementation of each tactic, in itself or in combination with some other tactics in the set, can fulfill the FSR; and (*ii*) check whether any feasible combination of tactics from the applicable safety tactics that are present in the vehicle technical architecture meets the FSR. Note that, for an FSR f_i and its corresponding functional component c_i, the applicable safety tactics for f_i need to be compared with only the safety tactics implementations used in the technical architecture counter part of c_i and its associated safety mechanisms since f_i is only associated with c_i.

Applicable safety tactics for an FSR can be identified based on the FSR description (by navigation through a tactic hierarchy) [85, 91, 92] or by matching the FSR description to the descriptions of each tactic [92]. Consider the following example FSR: "*failure in the acceleration sensing component of leader shall not communicate wrong acceleration information to the automatic steering component of the follower.*" According to the first method—safety tactic hierarchy [91]—an applicable safety tactic for failure containment using redundancy is *diverse redundancy* [91]. The same tactic can be identified by matching the FSR description to the tactic description [92]. For example, the *diverse redundancy* tactic's description—*"introduction of a redundant system which allows detection or masking of failures in the specification or implementation as well as random hardware failures"* [92]—matches the FSR description.

By the end of this two step process of identifying applicable tactics and checking the technical architecture for these tactics, we will have a list of FSRs that do not have any feasible combination of tactics implemented. If the list is empty, then the vehicular technical architecture fulfills all the FSRs for the given connected driving scenario. Otherwise, the list shows the FSRs that have not been fulfilled.

As a by-product, for each unfulfilled FSR, we will also have a set of applicable tactics such that some feasible combinations from this set can fulfill the FSR. These combinations point to a set of safety patterns since safety patterns are associated with the safety tactics they implement [92]. These applicable safety patterns (and applicable tactics) provide the system architects with a set of possible design decisions to realize the unfulfilled FSRs. Detailed analysis on the applicability of these safety patterns and trade-off analysis among them is beyond the scope of our work.

Note that the architecture tactics are not associated with any safety integrity level. Therefore, whether a tactic can address a given ASIL level is a research topic on its own and is beyond the scope of our work. Our objective for (the second phase of) our method is to identify relevant tactics to see whether they are implemented in the technical software architecture.

3.3 Research method

This section presents an application of the proposed method on a connected driving scenario: *platooning*. First, we describe the platooning scenario and the functional architecture of an individual vehicle, the two inputs to our proposed method. Next, we present the results of applying our method to platooning and its interpretation. All artifacts generated are available online [103].

A platoon is a vehicle train in which a manually driven vehicle (referred to as *leader*) is autonomously closely followed by at least one vehicle (referred to as *follower*). In a platoon, vehicles coordinate with each other using vehicle-to-vehicle (V2V) communication. Platooning has shown the potential to (*i*) reduce average fuel consumption [76]; (*ii*)

improve safety—for example, by preventing rear end collisions by enabling platoon-wide braking [77]; and (*iii*) increase traffic throughput by increasing average speed and reducing traffic jams. In this case study, the scope of platooning is limited to highways and highway interchanges.

We applied the proposed method on a connected driving software architecture developed for the i-CAVE project[5] that is deployed on *Renault Twizy*[6] – a small electric vehicle. The vehicle is fitted with extra sensors and actuators including a complete software stack (hereafter referred to as i-CAVE demonstrator). The software stack of the i-CAVE demonstrator is deployed on a combination of a real-time computer—an Advantech ARK-3520P[7]—that runs the Simulink RealTime operating system and an Nvidia's Drive PX2 platform.[8] In-depth details of the prototype (including mechanical design, the details on individual sensors and extra outfits on the vehicle) are beyond the scope of this chapter. Interested readers can refer to Hoogeboom's work for details [104].

A simplified functional architecture of i-CAVE demonstrator is shown in Figure 3.4a. For simplicity, we present only those functional components that are fundamental to achieve platooning. The arrows indicate data flow from sensor abstraction to actuator while the system as a whole is a closed control loop. Note that we focus on uni-directional information flow in our case study. However, a complete platoon setting will include bi-directional information flow.

Some of the functional architecture components are grouped to classes based on their functionality (as shown in Figure 3.4a). For example, sensor abstraction is a class of components that contain two types of functional components namely actuation sensors and environment perceptions sensors. The functional components inside each class act as independent entities and do not have data flow between them. The functional components of the architecture are described below:

a) *Sensor abstraction* consists of hardware sensors and their encapsulation via its software interfaces. Two classes of sensors are functionally distinguished: (*i*) *actuation sensors* that monitor vehicle state and dynamic attributes like speed and inertial measurements; (*ii*) *environment perception sensors*, like RADAR and GPS, that monitor the vehicle's external environment and localize the vehicle on the map.

b) *Sensor fusion* combines data from different kinds of sensors to generate information about the vehicle and its surroundings. The sensor fusion of i-CAVE demonstrator has three functional components: (*i*) *host tracking* that combines location and inertial measurement data to determine the absolute position of the vehicle, (*ii*) *vehicle state estimator* that combines acceleration information with data from actuation sensors to estimate the dynamic state

[5] https://www.nwo.nl/en/cases/i-cave-five-years-research-cooperative-and-autonomous-driving

[6] https://www.renault.co.uk/electric-vehicles/twizy.html

[7] https://www.advantech.com/en/products/1-2jkd2d/ark-3520p/mod_6666bf1e-af4f-47b6-8006-1a0a89eb3c93

[8] https://developer.nvidia.com/drive/

of the vehicle, and (*iii*) *target tracking* component that combines data from environment perception sensors like radar to detect objects and other vehicles in the surroundings of the vehicle.
c) *V2V communication* communicates actuation-related signals for platooning between a vehicle and its surrounding vehicles.
d) *Vehicle control* generates control signals for autonomous actuation of the vehicle using the information about the state of the vehicle, its surroundings, and information about the vehicle in front (received via V2V communication). When manually driven, this component receives actuation commands from a human driver.
e) *Actuator* is hardware and corresponding software interface for accelerating, steering, and braking of the vehicle, also known as drive-by-wire interface.
Note that the components to fulfill non-functional requirements (outside the platooning functionality), like safety management components, are not shown since they are not part of basic functional architecture needed to achieve platooning.

3.3.1 Derive FSRs for platooning

Following are the steps in the first part of our method, depicted in Figure 3.2.

Functional decomposition: We decompose the platooning scenario description (also referred to as *SD*) into five sub-scenarios.

SC-1 A vehicle can join a platoon as a follower after the last follower.

SC-2 A follower can leave a platoon.

SC-3 A platoon can split into two platoons.

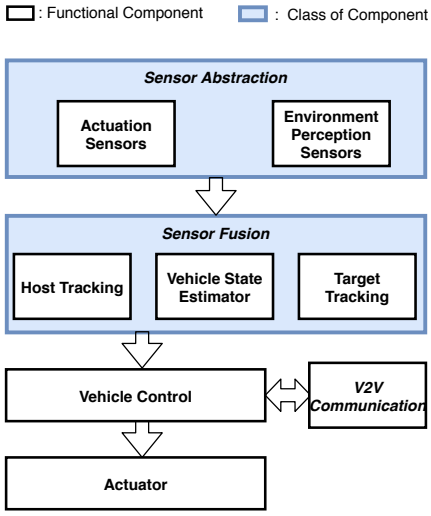
SC-4 Two adjacent platoons can merge into a single platoon.

SC-5 When the leader leaves a platoon, the first follower becomes the new leader.

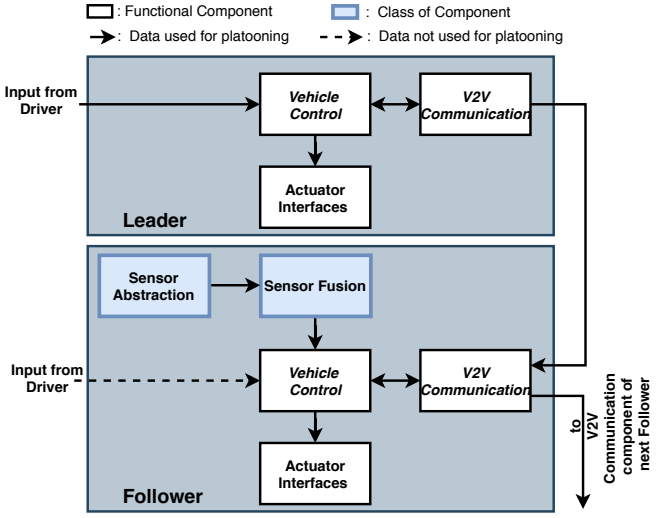
A platoon is formed when one vehicle joins another vehicle to form a two-vehicle platoon. Eventually, a platoon is disbanded when a vehicle leaves a two-vehicle platoon. The join and leave actions in a platoon are performed manually by the driver of the vehicle.

The platooning scenario description is partitioned into 9 functions from the vehicle perspective and 6 functions from the connected perspective. These functions are listed in Table 3.1.

Hazardous events: Next, we identify hazards relating to these functions. We use the seven most common guide words from the automotive domain (*no, more, less, as well as, part of, reverse,* and *other than*) [38] to identify 57 hazards. For example, the platoon function—*"keep sufficiently safe inter-vehicular distance"*—with the guide word *less* creates the hazard—*"keeping less than sufficiently safe inter-vehicular distance"*—that can potentially



(a) Vehicle functional architecture



(b) Connected functional architecture for platooning

Figure 3.4: A simplified functional architecture of i-CAVE demonstrator and the platooning architecture (connected architecture) derived from it.

lead to crash inside a platoon. The list of hazards is available online [103] and the count of hazards derived from each function is shown in Table 3.1.

These 57 hazards (26 from connected perspective and 31 from vehicular perspective) when combined with operational modes (7 from connected perspective and 6 from vehicle perspective) and operational situations (2 per perspective) resulted in 340 hazardous events, 140 from vehicle perspective and 200 from connected perspective. Note that not every combination of hazards, operational modes, and operational situations is feasible and the infeasible combinations are not considered further. An example of a hazardous event from connected perspective is: *"keeping less than sufficiently safe inter-vehicular distance (hazard) during merge with another platoon (operational mode) on highway (operational situation)"*.

Safety goals: For each hazardous event, we created a safety goal to prevent it. We merged similar goals in each perspective to have 14 and 11 safety goals from vehicle and connected perspective respectively. For example, the safety goal *"sufficiently safe inter-vehicular distance shall be kept regardless of the operational mode or operational situation of the platoon"* is formed by combining the goals derived from 56 hazardous events.

For ASIL allocation to safety goals, we assumed that all vehicles inside a platoon, except for the leader, cannot rely on a human driver for fallback in case of any failure. For vehicles joining or leaving a platoon, during the process of joining and leaving, we assume a human driver for fallback in case of failures. We have given the lowest score for *controllability* in the scenarios pertaining to follower vehicles. Since the leader is human-driven, the *controllability* of the leader vehicle is assumed to be the highest. The highest levels are assigned to the *severity* if a vehicle or platoon failure causes a crash since we assumed the speed range for highways. We assumed different *exposure* levels based on scenarios (joining platoon, leaving platoon, splitting of a platoon, merging of two platoons, and change of leader in a platoon) and operational situation (highway or highway- interchange) with the highest exposure levels in operational scenario highway. Therefore, most of the safety goals are assigned ASIL D. The detailed list of *exposure, controllability,* and severity levels assigned and resulting ASIL for each safety goal is available online [103].

Connected functional architecture: Figure 3.4b shows a simplified connected functional architecture for platooning with functional components for platooning as well as the working of vehicles within a platoon at the functional level. The connected functional architecture is created by four system architects, who are mechanical engineers involved in the development of i-CAVE demonstrator with at least a master's degree and a minimum of two years of experience in automotive architecture development. The connected functional architecture contains the same functional components as the vehicular functional architecture (see Figure 3.4a and Figure 3.4b), but only the components that are used to accomplish the connected functions and their interconnections are used. For example, a design choice of the system architects was to communicate the information from the vehicle control functional unit of the leader to the follower and not to communicate the sensor information between the leader and the follower. Thus, in the leader, the *sensor*

Table 3.1: Hazards for functions identified from platooning description

Connected functions	Hazard (count)
Keep optimized inter-vehicular distance within a platoon	4
Make place for a vehicle to join	6
Merge with another platoon	4
Split into two platoons	4
Change leader	4
Keep proper distance to the surrounding traffic	4

Vehicle functions	Hazard (count)
Autonomously follow the vehicle in front (follower)	3
Keep a proper distance to the surrounding traffic as part of a platoon (follower)	5
Leading the platoon (leader)	4
Take leader role of platoon	3
Switch from leader to follower role	3
Join platoon (follower)	2
Leave platoon	2
Timely react to the actions of surrounding vehicles in a platoon	5
Follow traffic indications, signs and rules	4

abstraction and *sensor fusion* class of functional components are not used for connected driving functions. Also, these functional components are not used for leader's own driving functions since the leader is manually driven. Therefore, in the connected functional architecture, in the leader block, these components are not shown for leader (see the leader block at the top of Figure 3.4b).

Safety analysis: Finally, FSRs are derived by mapping safety goals to the functional architectures using fault tree analysis (FTA) [42]. The FTA generated 16 FSRs from the vehicle perspective and 15 FSRs from the connected perspective. i.e., 31 in total. The count of FSRs for each functional component is presented in Figure 3.5 along with some example FSRs in the second column of Table 3.2.

Interpretation of results

In our case study, the traditional safety analysis (vehicular perspective) according to ISO 26262, resulted in 16 safety goals leading to 16 FSRs. While the proposed extension of safety analysis resulted in 9 more safety goals and 15 more FSRs, resulting in a total of 25

safety goals and 31 FSRs. The maximum number of FSRs from the vehicular perspective is associated with the *vehicle control* component (6 FSRs), while in the context of FSRs from the connected perspective, it is the *V2V communication* component (5 FSRs). Another interesting note is that most of the FSRs (17 out of 31; 12 from the vehicular perspective and 5 from the connected perspective) is assigned with ASIL D while only a relatively low number of safety goals (7 out of 25; 6 from the vehicular perspective and 1 from the connected perspective) as assigned with ASIL D. This difference in ASILs between safety goals and FSRs because of most functional safety requirements being related to multiple functional safety goals, and FSRs inherit the highest ASIL of their related safety goals.

Our count of FSRs (31 FSRs from 25 safety goals in total) is low compared to industry scenarios in which a similar count of safety goals are linked to more than 100 FSRs. We believe that the reduced number of FSRs is related to the simplicity of our vehicle functional architecture. To give perspective, a reference architecture presented in [105] has 39 functional components while our simplified architecture has 8.

It is possible to have overlap of FSRs derived from both the perspectives. That is, the same FSR can be derived as a result of connected and vehicular perspectives. Our case study, however, did not result in any overlapping FSRs.

3.3.2 Check fulfillment of FSRs

Following are the steps in second part of our method, depicted in Figure 3.3.

Check for conflicts in the derived FSRs: We grouped FSRs based on their associated functional architecture component. For example 9 FSRs belong to the functional architecture component *vehicle control* and 3 of them is shown in Table 3.2 (see details in the third–fifth row, first and second column). The overall count of FSRs grouped on associated component is shown in Figure 3.5. Within each group, we compared the descriptions of each pair of FSRs to identify potential conflicts. We did not find any conflict in the 8 groups. The complete list of FSRs grouped by functional architecture component and compared pairwise is available online [103].

Identify safety tactics for implementing each FSR: For each FSR we identified a list of applicable safety tactics. We chose the following 13 safety tactics on which the 15 most widely used safety patterns build [91, 92]: *simplicity, substitution, sanity check, condition monitoring, comparison, diverse redundancy, replication redundancy, repair, degradation, voting, override, barrier* and *heartbeat* [91, 92]. More details on individual tactics are presented in Appendix A.

Table 3.2: FSRs that are found to be fulfilled in the technical architecture of i-CAVE demonstrator. FSRs in blue cells are derived from connected (platooning) perspective and other FSRs are derived from vehicle perspective.

	FSR	Applied Tactics	Implementation in Technical Architecture
Environment perception Sensors	Failure of *environment perception sensors* shall not result in the generation of incorrect information on distance to the surrounding vehicles and objects.	*Sanity Check, Barrier, Heartbeat, Condition Monitoring*	Cyclic Redundancy Check (CRC) for messages *(sanity check)* and validity time per message *(heartbeat)* is implemented in the *Environment perception Sensors* component while a watch dog is implemented in the safety management *(condition monitoring)*. Software interface for each sensor is implemented independent of each other to protect from unintended influence between interfaces *(barrier)*.
Actuation Sensors	External interference shall not invalidate/corrupt data from *actuation sensors* .	*Sanity Check*	CRC and a message counter is implemented
Vehicle Control	A failure in *vehicle control* shall not cause generation of incorrect actuation signals	*Barrier, Condition Monitoring*	Two independent driving modes are implemented in *vehicle control* component. One mode generate control signals (when in follower role) relying on *V2V communication* and the other without relying on *V2V communication (barrier)*. A monitor for checking correct working of (and switching between) the two modes is implemented in safety management *(condition monitoring)*.
	A failure in *vehicle control* shall neither inhibit nor modify the input from driver to further pass on.	*Simplicity*	The driver input is bypassed directly to *Actuators*.
	A failure in *vehicle control* shall not cause a switch to manual drive mode while in platooning mode	*Sanity Check, Override, Condition Monitoring*	A state machine based method for mode selection and monitoring is implemented as a part of safety management.
V2V Communication	Failure in *V2V communication* shall not transmit incorrect information to or receive incorrect information from a vehicle joining or leaving a platoon.	*Heartbeat*	Heartbeat messages to continuously monitor reliability of communication channel are implemented in *V2V Communication*.

Each safety tactic has an aim and a description of its scope [92]. For example, the aim of safety tactic *simplicity* is to *"avoid failure by keeping a system as simple as possible"* and its description is *"Simplicity reduces system complexity. It includes structuring methods or cutting unnecessary functionality and organizing system elements or reducing them to their core safety functionality to eliminate hazards."* [92].

Applicable tactics for each FSR: To identify whether an implementation of a safety tactic can realize an FSR, the aim and description of the safety tactic is matched with the description of the FSR. Examples of FSRs, safety tactics that match them as well as their implementation are presented in Table 3.2. Table 3.2 also shows that the first FSR listed does not match the simplicity tactic (not present in column 3) resulting from the inherent complexity of *environment perception sensors*. A complete list of the 31 FSRs and matched safety tactics is available online [103].

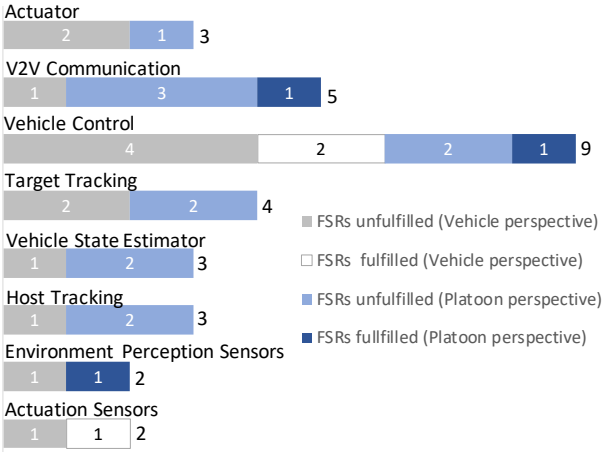


Figure 3.5: The 31 FSRs, grouped by associated functional component. The total FSRs for each group is shown at the end of each stacked bar

Check for safety tactics implementations in technical architecture: Finally, to identify whether the vehicle architecture meets an FSR, we analyzed the implementation of the associated functional architecture component in the technical architecture of the i-CAVE demonstrator. The technical architecture of the i-CAVE demonstrator is implemented in MATLAB/Simulink. We inspected the MATLAB code as well as the Simulink state flow diagram to identify functional architecture components as well as any associated safety management system. We mapped the implementations of these functional architecture components to the safety tactics identified for each FSR to evaluate whether each FSR is fulfilled by the technical architecture. Table 3.2 shows the FSRs that are found to be fulfilled

in the technical architecture, the tactics applied from the set of applicable tactics, and how the specific combination of applied tactics fulfills the corresponding FSR. Also, an example of FSR that is found to be unfulfilled is: *"A failure in* Actuator *(software interface) should not cause propagation of incorrect control signals to hardware actuators".* A complete list of unfulfilled FSRs is available online [103].

For each functional component, Figure 3.5 shows the count of FSRs that are realized and not realized from the vehicle as well as the platooning perspective, respectively. Recall from Section 3.3.1 that we derived 16 and 15 FSRs from the vehicle and platooning perspective, a majority of them relate to vehicle control. Out of the 16 FSRs for the vehicle perspective, 3 FSR are fulfilled by the vehicular technical architecture and the remaining 13 FSRs are unfulfilled. Likewise, for the connected perspective, 3 FSRs are fulfilled and the remaining 12 FSRs are unfulfilled. We showed our results to the four system architects of the i-CAVE project. They confirmed that fulfilled FSRs are implemented and unfulfilled FSRs are not implemented in i-CAVE demonstrator. For the 25 FSRs unfulfilled by i-CAVE demonstrator, we provide a list of applicable safety patterns that can act as a starting point for the next design iteration of the technical architecture.

Interpretation of results

Our study shows that the technical architecture meets only 6 out of 31 FSRs (with all six fulfilled FSRs presented in Table 3.2). In our case study, we checked whether FSRs are fulfilled in the i-CAVE demonstrator using 13 safety tactics. The set of tactics was chosen based on their use in the 15 most widely used safety patterns [92]. It is possible that we might have classified some fulfilled FSRs to be unfulfilled since we considered only 13 tactics. However, the architects of the i-CAVE project agreed to our findings. This indicates that our classification was correct.

Our assessment using these safety tactics showed that, out of the 15 FSRs from the connected perspective, 12 were unfulfilled. Notably, we found almost as many unfulfilled FSRs from the vehicle perspective as from the connected perspective. An explanation for this observation relates to the capabilities of the vehicle behind the i-CAVE demonstrator. The i-CAVE demonstrator uses a Renault Twizy[9] which is a bare-bones two seater electric vehicle. To give perspective, the Renault Twizy is small enough ($2338mm \times 1381mm$) to be used in bicycle lanes, is lightweight (gross weight of 690 kilograms) and has a driving range of up to 51 kilometers. In contrast, Tesla's entry level vehicle, Model 3,[10] is almost double in dimension, three times in gross weight, and more than ten times in driving range. As a result, the i-CAVE demonstrator has limited features and components. Also, the demonstrator is a work-in-progress being developed iteratively by a multi-domain team. Since some parts of the technical architecture were not implemented during our case

[9] https://www.renault.co.uk/electric-vehicles/twizy/specifications.html
[10] https://www.tesla.com/model3

study, our results merely point to the missing implementations as unfulfilled FSRs in the vehicle perspective. Future iterations of the demonstrator[11] can use this list of unfulfilled FSRs from both the vehicle and the connected perspective to improve the i-CAVE technical architecture.

In summary, our case study found unfulfilled FSRs from the connected perspective showing the viability and applicability of the proposed method. The results of our case study show better coverage of safety goals by providing additional FSRs as compared to the ISO 26262 process [5]. The current safety engineering methods to derive FSRs are outlined by ISO 26262 standard [5, 80] which lacks the connected perspective. It provided valuable insights in the context of i-CAVE project. In our case study, we found 15 FSRs from the connected perspective, making up 48% of all FSRs. Yet it is still a mere illustration of our method. Clearly, replications are required to verify the generalizability and scalability of our method. Nonetheless, our results corroborate the existing body of knowledge [78,80,81] in showing that the current safety standard misses FSRs from the connected perspective.

The results of our case study show better coverage of safety goals by providing additional FSRs as compared to the ISO 26262 process [5]. The current safety engineering methods to derive FSRs are outlined by ISO 26262 standard [5,80] which lacks the connected perspective. In our case study, we found 15 FSRs from the connected perspective, making up 48% of all FSRs. Our results corroborate the existing body of knowledge [78, 80, 81] in showing that the current safety standard misses FSRs from the connected perspective.

3.4 Discussion

We present a deeper exploration into the proposed method in terms of implicit assumptions. We describe how our solution is likely to apply to connected driving scenarios in real-life. Below we discuss the implicit assumptions in our method, the scalability, generalizability, and the scope of our method.

3.4.1 Assumptions

The proposed method borrows some assumptions applicable to single-vehicle and applies them to connected driving. These assumptions are derived from the safety engineering domain as well as the software architecture domain. For example, it is assumed that proper functional separation of the system is always possible. This results in every FSR being mapped to exactly one functional component. Such a functional separation is a standard practice in the safety engineering domain and has been followed for at least five decades [42]. This separation is also underlined by the product development standard in the automotive domain—ISO26262 [5, 101] and automotive architecture frameworks [93]. Nonetheless, the applicability of this assumption in connected driving is not established.

[11] https://www.nwo.nl/en/cases/i-cave-five-years-research-cooperative-and-autonomous-driving

Similarly, the second part of our method relies on two assumptions: (*i*) it is possible to map functional components to implementations in the technical software architecture; and (*ii*) every FSR can be fulfilled by a combination of safety tactics. Our first assumption comes from the architecture frameworks in the automotive domain [93]. The assumption about safety tactics stems from the architecture domain, which considers safety tactics as design primitives, and architectures are formed by the combination of design primitives [85]. Mature architecture assessment methods like ATAM also rely on this assumption, albeit in the context of tactics for the quality attributes they focus on [84, 85]. Nonetheless, the applicability of this assumption in the automotive domain is not established.

3.4.2 Applicability

Our case study presents a simplified version of connected driving use cases. In real-life, the proposed method should work on a bigger scale and apply to connected driving systems with various entities. The potential factors limiting the scalability and generalizability of a method include the complexity of the system, heterogeneity of participating systems (e.g., different types of vehicles (car and truck) and/or vehicles from different manufacturers), and inclusion of entities other than participating vehicles, like the cloud, to enable connected driving functionalities.

Scalability: Our method is modular, which means that it is likely to scale to complex systems. The method uses two levels of abstraction at the architecture level. The functional architecture view separates functionalities such that each component performs one unique function and collectively performs a connected driving function. This ensures that the safety requirements for connected driving functions can be allocated to individual vehicular components without entering into their implementation details. In the second part of the method, all the FSRs pertaining to one component are assessed against their implementation details. Segregation of safety requirements pertaining to each component and handling each component separately, ensures the applicability of our approach to complex systems.

Heterogeneity: The functional architecture view acts as a black box separating functional components and interaction among functional components from their implementation, making our approach agnostic of vehicle type and brand. As a result, we believe that our approach can assess the safety of connected driving systems that involves different kinds of vehicles (for example, platoon containing both trucks and cars) as well as different automotive brands (for example, platoon containing cars from BMW and GM) as long as the functional architectures of the participating entities are provided.

Entities other than vehicles: Entities enabling connected driving functionalities can be beyond participating vehicles. One such example is cloud communication. The connected architecture and corresponding item definition in the first phase of our approach are specifically introduced to ensure that all the entities involved in enabling connected functionalities are systematically considered in the safety analysis. For example, in the

use cases that include cloud communication, the cloud will be a part of the connected architecture.

Fail-operational and fail-safe designs: Our work is designed for connected systems irrespective of their operational design domain and whether they are designed to be fail-operational or fail-safe [106, 107]. The proposed method is generic for both fail-operational as well as fail-safe systems. Our method ensures that both connected and vehicular perspectives are covered while deriving FSRs.

3.5 Related work

The proposed method has two parts: (i) deriving FSRs for connected driving and (ii) check whether each FSR is fulfilled in the technical software architecture of the vehicle. These two aspects are addressed separately in the literature, and the related research primarily stems from two domains: software architecture and safety engineering.

Software architecture

A variety of architecture assessment techniques has emerged from the software architecture research community in the past three decades. These architecture assessment techniques assess the "goodness" [85] of an architecture(s) with respect to some property (or a set of properties). Such properties are termed as quality attributes. Quality attributes can be divided into two broad categories: operational (e.g., reliability, performance) and development (e.g., maintainability, re-usability) [90]. This chapter focuses on *functional safety* as an operational quality attribute.

The software architecture assessment techniques proposed for operational quality attributes [82, 83] mainly use mathematical modeling & analysis and scenarios of system operation (also known as scenario-based techniques) to uncover whether the architecture achieves the intended quality attributes sufficiently [108]. The assessment techniques that use mathematical modeling mainly focus on reliability and performance as quality attributes [109]. Some studies have shown that these techniques are not scalable and hence not suitable for complex systems of systems like connected driving systems that are built by multiple inter-disciplinary teams [109].

The prominent scenario-based architecture assessment methods for operational quality attributes are the Architecture Trade-off Analysis Method (ATAM) [84, 85], Scenario-Based software Architecture Re-engineering (SBAR) [86], Software architecture Comparison Analysis Method (SCAM) [87], Domain Specific software Architecture comparison Model (DoSAM) [88], and Pattern-Based Architecture Reviews (PBAR) [89].

PBAR is designed for light weight evaluation, primarily performed on small projects with some case studies on projects with at most 10 developers [89, 110]. It is not suitable for evaluation of complex safety-critical systems that we assess in this chapter [89]. SCAM and DoSAM are designed for comparing different architectures rather than assessing an

individual architecture [87, 88]. These methods grades each of the architectures under comparison on a normalized scale, typically from 0 to 100, and use this to characterize the fitness of a candidate architecture in contrast to others. SBAR is an iterative method for re-engineering of architectures for functionality based re-design [86] including for architectures that might not properly separate functional concerns. We assume that the automotive architectures for our analysis are designed based on separation of functional concerns since this is a standard practice in the automotive domain, enforced by safety engineering [5, 101]. Moreover, SBAR suggests scenario-based techniques for development quality attributes and simulation based assessment for operational quality attributes [83]. In contrast we consider scenario-based methods for the operational quality attribute *functional safety*.

ATAM is the most mature and widely used architecture assessment method in practice [85]. ATAM, in its current form, is primarily used to analyze trade off among different quality attributes and to identify stress points and sensitivity points in the architecture under assessment. ATAM facilitates usage of existing knowledge in the form of tactics, which we take inspiration from and reuse in our proposed method.

ATAM considers six quality attributes, however, functional safety is not one of them [85]. Note that case studies of ATAM's application to safety critical domains like avionic systems [111] do not stress safety as a primary quality attribute either. Even though ATAM provides some methods for scenario elicitation, it does not provide a systematic method for scenario decomposition to generate requirements for individual architecture components. This is crucial in the context of systems of systems since a scenario may lead to a multitude of requirements affecting different systems which are to interact with each other to perform the intended action(s).

In summary, within the field of software architecture, none of the software architecture assessment methods that we found are applicable for analyzing the functional safety of connected automotive systems.

Safety engineering

Now, we present related research on the application of safety engineering concepts in automotive software and system evaluation. We primarily present related research on (*i*) identifying FSRs in automotive settings and (*ii*) methods to check or ensure that a technical architecture realizes FSRs.

Identifying FSRs: Studies on deriving FSRs largely focused on the perspective of individual vehicle as a system while just a few explored the perspective of a set of vehicles as a system. Studies to derive FSRs from the vehicle perspective present different mechanisms to generate safety goals and map these to the functional architecture using safety analysis methods. For instance, Beckers et al. [112] presents a model-based method to define FSRs given safety goals.

Studies on connected driving systems try to replicate the mechanisms from an individual vehicle perspective. For example, Oscarsson et al. [113] uses system theory for the safety analysis from the perspective of set of vehicles as a system. Another study proposed an alternative safety analysis technique, using possible accidents as a starting point to identify FSRs [114]. Our study closely follows the study by Saberi et al. [81] in deriving FSRs from connected driving scenarios.

Checking or ensuring architecture realizes FSRs: Studies on a single vehicle perspective use many different approaches to ensure that systems satisfy FSRs. One approach uses an architecture description language for safety verification [115]. Martin et al. [116] uses architecture patterns to incorporate FSRs in the design phase. Sljivo et al. [117] presents a method for fulfillment of FSRs at design time using design patterns and contracts. Other approaches use formal methods to verify that systems satisfy FSRs, although the solutions do not scale [118–120].

Ensuring fulfillment of FSRs as part of a connected system is challenging [77]. A majority of works on connected systems proposes a reference architecture from a system of systems viewpoint [77]. Some other solutions look at specific architecture components in specific connected scenarios, thereby missing high-level insights [78].

To the best of our knowledge these studies, with their scope of complete system architecture, focus on fulfilling FSRs during the design phase. This chapter, in contrast, focuses on checking for the fulfillment of FSRs on existing architectures or when designing architectures.

3.6 Threats to validity

Our proposed method and the findings from the case study are susceptible to threats relating to human participation and choice of techniques. Below, we present potential threats and our attempts at mitigating them.

Cognitive bias: Several steps of our proposed method and the related case study rely on the expert opinions of architects. This step may have resulted in cognitive bias [121] in relation to human judgment. To mitigate this threat, for every step that required human judgment, we consulted at least three experts (in addition to the first two authors), who performed the steps independently. For example, (*i*) the connected functional architecture was created from the vehicle architecture and scenario descriptions, in consultation with four expert system architects, independently (*ii*) two of the authors independently checked for conflicts among FSRs; and (*iii*) The validity of the safety goals depends on the decomposition of a scenario description to functions. The decomposition of the scenario description to functions was validated by the third author, who is an expert in functional decomposition with over five years of industry experience in the functional safety domain and a participant in the development of the automotive industry's functional safety standards ISO 26262 and ISO 21448.

Technical bias: To generate FSRs, we chose fault tree analysis [42] as the safety analysis technique. The choice of other techniques, like failure mode effect analysis [43], may influence the outcome. We need empirical studies to check whether the choice of safety analysis technique introduces differences in findings.

Choice of safety tactics: In our case study, we checked whether the FSRs are fulfilled in the i-CAVE demonstrator using 13 safety tactics. The safety tactics are chosen based on their use in the 15 most widely used safety patterns [92, 99]. This list of safety tactics is not complete and defines the scope of our case study.

3.7 Future work

Our work is an initial step in the direction of functional safety assessment for connected driving. This section presents potential future directions.

Cyber-security alongside functional safety: Cyber-security is a prominent directions to explore in connected driving alongside functional safety. The connected nature of connected driving increases the potential attack surfaces and can compromise the system's functional safety. Integral approaches that consider safety and security together are a potential future research direction.

Hardware topology: Functional safety is often achieved via hardware architecture or a combination of hardware and software. The second part of our method focused only at the software level. Extending the second part of the approach to address functional safety requirements that are fulfilled specifically in hardware topology and the combination of hardware and software is the logical next step of the proposed approach.

ASILs for safety tactics: Currently, safety tactics are not associated with ASIL levels. This means that, from the current taxonomy of safety tactics, we can only conclude whether a tactic addresses an FSR rather than whether it addresses the FSR at the specific level of ASIL. Augmenting safety tactics with ASILs is a potential future research direction. This will allow prioritizing FSRs based on the risk associated with them. This may also be a step towards a trade-off analysis where each FSR can be traded off with other requirements based on the risk associated.

Alternative architecture abstraction levels: Currently, the second part of our approach, checking fulfillment of FSRs, uses the technical architecture view. It can be argued that a higher level of architecture abstraction than the technical architecture view can be used instead. We plan to evaluate this in the future.

Finally, our method adapts existing solutions for addressing functional safety in the context of connected driving scenarios. Alternative methods to check for unfulfilled FSRs will be an interesting direction to explore.

3.8 Conclusion

This chapter investigated whether the architecture of a single vehicle meets the functional safety requirements for connected driving. We proposed a method to ensure that an automotive architecture is functionally safe to operate in given scenarios. The proposed method derives functional safety requirements for a connected driving scenario and checks whether they are fulfilled in the technical architecture of a vehicle. The method is a combination of methods adapted from the safety engineering and software architecture domains. We show the usability of our method for a connected driving scenario, platooning, on an academic prototype, resulted in uncovering functional safety requirements that were not fulfilled by the software architecture. Our method is motivated by and reinforces the notion that functional safety should not be an afterthought in the design of automotive architectures rather be used for defining the architecture of the automotive system.

4

Safety of Perception Systems for Automated Driving: A Case Study on Apollo

The automotive industry is now known for its software-intensive and safety-critical nature. The industry is on a path to the holy grail of completely automating driving, starting from relatively simple operational areas like highways. One of the most challenging, evolving, and essential parts of automated driving is the software that enables understanding surroundings and the vehicle's own as well as surrounding objects' relative position, otherwise known as the perception system. Current generation perception systems are formed by a combination of traditional software and machine learning-related software. With automated driving systems transitioning from research to production, it is imperative to assess their safety.

We assess the safety of Apollo, the most popular open-source automotive software, at the design level for its use on a Dutch highway. We identified 58 safety requirements, 38 of which are found to be fulfilled at the design level. We observe that all requirements relating to traditional software are fulfilled, while most requirements specific to machine learning systems are not. This study unveils issues that need immediate attention; and directions for future research to make automated driving safe.

This chapter is based on

S. Kochanthara, T. Singh, A. Forrai, L. Cleophas. Safety of Perception Systems for Automated Driving: A Case Study on Apollo, Under revision at a journal

The automotive industry has transitioned from an electro-mechanical to a software-intensive industry. Current and future vehicles are characterized by immense use of software to enable automated (a.k.a. self-) driving. Today, industry players like Waymo and Baidu have shown the capability of completely automating driving (i.e., without the need of a human driver for emergency takeover) in relatively simple situations like specific geographic locations and restricted weather and illumination conditions.[1] Likewise, many software companies, including Apple, Sony, and Uber, are reportedly developing their automated driving frameworks.[2]

This chapter focuses on perception systems in automated driving frameworks. Perception refers to sensing surroundings for semantic understanding, such as identifying traffic signs and locating the vehicle's own position and the relative position of objects around [14]. This information is used for planning and executing the next driving decision. Perception systems are arguably the most evolving and relevant part of any automated driving framework [12].

Software engineering research on perception systems has explored multiple aspects, including their development [122], complexity [123], and use of machine learning (ML) models in perception systems of existing automated driving frameworks [13,15]. With many automated driving frameworks transitioning from research to production, one challenge that the automotive industry, regulatory bodies, and legal authorities experience today is the safety of automotive software, which is imperative for its public acceptance [124]. Relating to safety, recent software engineering literature primarily focuses on validation & verification; with most of the studies on testing [125–127] and related aspects [128]. Bridging a gap in the literature, this chapter presents a case study assessing the safety of perception systems at the design level.

We study Apollo 7.0's [11] perception system software for its use in a segment of the Dutch highway A270.[3] Existing studies show that Apollo is the most popular open-source automotive repository [12], with its development history in GitHub dating back to 2017. It is currently one of the most advanced automated driving frameworks [13], embraced by many world's top automakers, and is used to offer automated driving services to the public in parts of the world.[1]

This study makes two contributions. One, eliciting safety requirements, and two, design assessment of the elicited safety requirements. We focus on three aspects of safety requirement elicitation (a) system or sub-system failures [5]; (b) data corruption [5]; (c) insufficient situational awareness arising from limitation of sub-systems on specific conditions (e.g., due to weather) [6]. There are more dimensions to safety requirement elicitation like deficiencies in specified driving behavior [7]; and incorrect and inadequate

[1] https://www.engadget.com/baidu-apollo-go-robotaxi-shenzhen-141727050.html
[2] https://bwnews.pr/3KRZwYS
[3] https://www.openstreetmap.org/directions?engine=fossgis_osrm_car&route=51.4564%2C5.5408%3B51.4657%2C5.5865#map=15/51.4610/5.5636

human-machine interface design leading to inappropriate or incorrect user situational awareness (e.g., confusion, overload, or inattentiveness to users) [6, 8]; which are not considered in this study.

For safety requirement elicitation on failures and data corruption we use the industry standards and traffic authority guidelines [5, 16, 17, 129] based on its high adoption [12], compliance requirements [129], and proven applicability in automotive domain[1]. To the best of our knowledge, no existing case study in the scientific literature that elicits these three kinds of safety requirements for a real-life highway and a mature software stack from the industry.

The resulting requirements can be divided into two categories: (1) requirements that can be assessed in the traditional software and (2) requirements specific to ML systems. An example of traditional software requirement is *"a failure of the camera sensor in the camera-based perception system shall not lead to an incorrect estimation of the state of vehicles or other obstacles."* An example of an ML system requirement is *"the performance deterioration of a camera-based perception system due to low light in the night shall not lead to an incorrect estimation of the state of vehicles or other obstacles."* [4]

For traditional software safety requirements, we use existing frameworks to assess Apollo's design [70, 71, 85]. Since there is no similar framework for assessing safety requirements specific to ML systems, we systematically prepare a curated list of ML specific design choices relating to safety and use them for design assessment. Our assessment uses publicly available data like documentation, architecture, code, datasets and related artifacts, and scientific papers linked to the documentation. An overview of the entire elicitation and assessment process is depicted in Figure 4.1.

In summary, our study contributes the following:

- We present a case study of a mature, automated driving software stack from industry for its real-life highway use, the first in the scientific literature. For transparency and replicability, in addition to the safety requirements, we provide results from all intermediate steps [130].
- We identify 58 safety requirements, specific to a Dutch highway segment of A270, that can enable safe automated driving on highways.
- We present a curated list of 10 ML specific design choices for assessing the quality attribute safety at design level.

[4]Note that these are hypothetical examples similar to the ones from this case study, which does not demand knowledge of the detailed architecture of the perception system for comprehension. Original requirements might require some knowledge of the perception system architecture to comprehend. These actual requirements and how they are systematically derived are presented in Section 4.2.

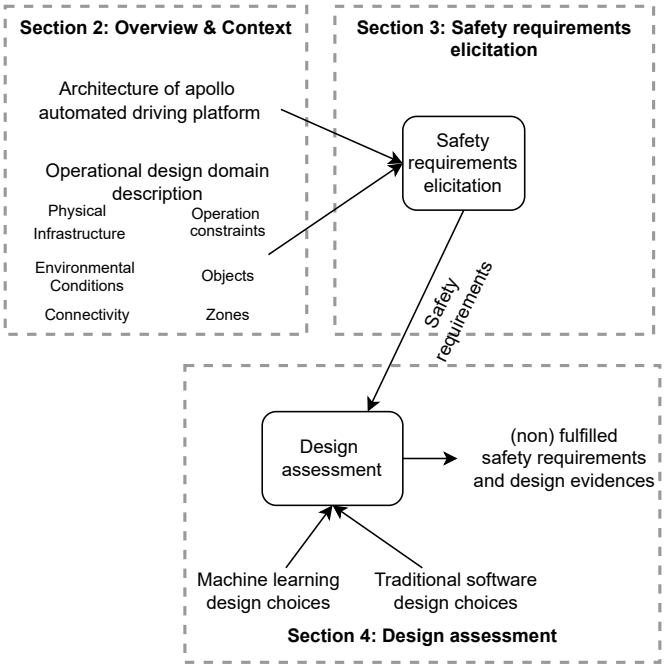


Figure 4.1: Overview of the design assessment process

- Our study shows that there exists design evidence for the fulfillment of 38 out of 58 safety requirements. A detailed description of how and where to find them is available as a part of the replication package [130].

The rest of the chapter is organized as follows. Section 4.1 presents an overview of safety assessment along with a brief introduction to the architecture of Apollo automated driving framework and a description of our operational area—a Dutch highway segment. Section 4.2 and Section 4.3 describe how we elicit and assess safety requirements, respectively, and our findings. Section 4.4 discusses our findings and their implications for research and practice. Threats to validity and related work are presented in Section 4.5 and Section 4.6, respectively. We present concluding remarks in Section 4.7.

4.1 Overview and context

This section presents an overview of the safety assessment process and outlines the assessment context. The entire process can be divided into three parts, as shown in Figure 4.1.

The first part is systematically identifying the necessary information needed to conduct safety requirements elicitation. This includes Apollo's detailed architecture [11] and a systematic description of the intended operational area [16, 17]. There did not exist a detailed architecture or a operational area description. A detailed description of both and how we created them is presented in Sections 4.1.1 and 4.1.2.

The second part is deriving safety requirements for Apollo's perception system. In this work, we focus on safety requirements relating to three aspects: (1) failure of a component; (2) data corruption; (3) limitations to the intended functionality (leading to insufficient situational awareness). For the limitations in the functionality of ML components, we concentrate on different weather and illumination conditions of the operational area that can lead to the limitations. Section 4.2 presents the method we used for eliciting safety requirements and the resulting requirements.

The third part is assessing the perception system, where we focus on the design decisions and how they (do not) fulfill the safety requirements. The perception system relies on multiple ML models along with traditional software. We assess the requirements related to failure and data corruption in the architecture of the traditional software. ML models have different design choices since they are fundamentally different from traditional software. For ML models, the logic is automatically deducted from the training data, while logic is manually programmed for traditional software. Requirements on limitations to the intended functionality (specific to ML components) are assessed in the sub-systems that rely on ML components. We explain the method and the results in Section 4.3.

4.1.1 Apollo: an open autonomous driving platform

Apollo is an open-source, automated driving platform from the Chinese search engine company, Baidu. Our choice of Apollo is motivated by its popularity [12], prominence of usage [12], continuous development since 2017, prior usage in research articles (e.g., [13]), and industry ownership. We use the current version, Apollo 7.0, for this study. The information used in this study is derived from publicly available documents, including Apollo's documentation on Github [11].

To create a detailed architecture, we started with the documentation available in GitHub. While an abstract outline is available in the documentation, the detailed architecture as shown in Figure 4.2 did not exist. To create this architecture, we combined information from the source articles pointed by the documentation as well as prior research article [13] that discusses the platform. Each individual module is identified from the documentation and folder structure of the repository. The architecture of the individual modules is identified based on the code, referenced (scientific) articles, and associated documentation. For example, the module for localization and its overall role in perception is found using the overall documentation[5] and the organization of the repository. Then the next level of

[5] `https://github.com/ApolloAuto/apollo#readme`

details is identified from the module documentation[6] and code[7]. The next level of details is derived from the source article [131] (pointed at by the module documentation) which dives deeper into the architecture and implementation of the different localization techniques. The rest of this section presents an overview of Apollo's architecture with a focus on its perception system.

The components in Apollo's architecture can be grouped into four categories: perception, decision & control, interface to vehicle platform, and safety systems, as indicated with the dotted rectangles in Figure 4.2. The dotted arrows in Figure 4.2 (inside the perception system) indicate information processing pipelines. The pipelines and components consisting of ML systems are shown in light green.

The *perception system* is responsible for understanding the surroundings, identifying obstacles, and giving all information needed for components in decision & control. In Figure 4.2, the dotted arrows inside the perception system indicate information processing pipelines. The pipelines and components consisting of ML systems are shown in light green.

The *decision & control* part is formed by the following 4 sub-parts: (a) *prediction*, which predicts the trajectory of moving objects surrounding the automated driving vehicle; (b) *routing*, which identifies a path from source to destination to be followed by the vehicle; (c) *planning*, which plans the next maneuver of the vehicle based on the inputs from prediction, routing, and perception; and (d) *control*, which takes its inputs from the planning and various sensors to identify the current pose (a combination of position and orientation including yaw, roll, and pitch[8]) of the vehicle and generate messages to the vehicle platform for executing automated driving through the trajectory obtained from planning.

The *vehicle interface* is responsible for two kinds of functions: (a) conveying commands like steering angle and throttle to execute the desired maneuver of the vehicle (or simulation system) on top of which the Apollo framework operates; and (b) dealing with other parts like lights and turn signals.

The *safety system* is responsible for monitoring (primarily) the perception and decision & control parts to identify potential faults and failures and maintain the automated driving system in a safe state. For instance, in the event of partial failure of the perception system, the safety system is responsible for making the vehicle reach a safe stop.

We exclude some components from the architecture that are not required for the perception system's safety requirement elicitation—e.g., human-machine interface. In the rest of this chapter, we focus on the perception system.

The perception system in Apollo primarily uses three kinds of sensors to sense the environment: camera, Light Detection And Ranging (LiDAR), and radar. The information

[6] https://github.com/ApolloAuto/apollo/tree/master/modules/localization#readme
[7] https://github.com/ApolloAuto/apollo/tree/master/modules/localization/proto
[8] Yaw, roll, and pitch three different kinds of motions of a vehicle based on three different axes. For a pictorial view and details see https://www.liskeforensics.com/blog/title/vehicle-pitch-roll-and-yaw/id/249/

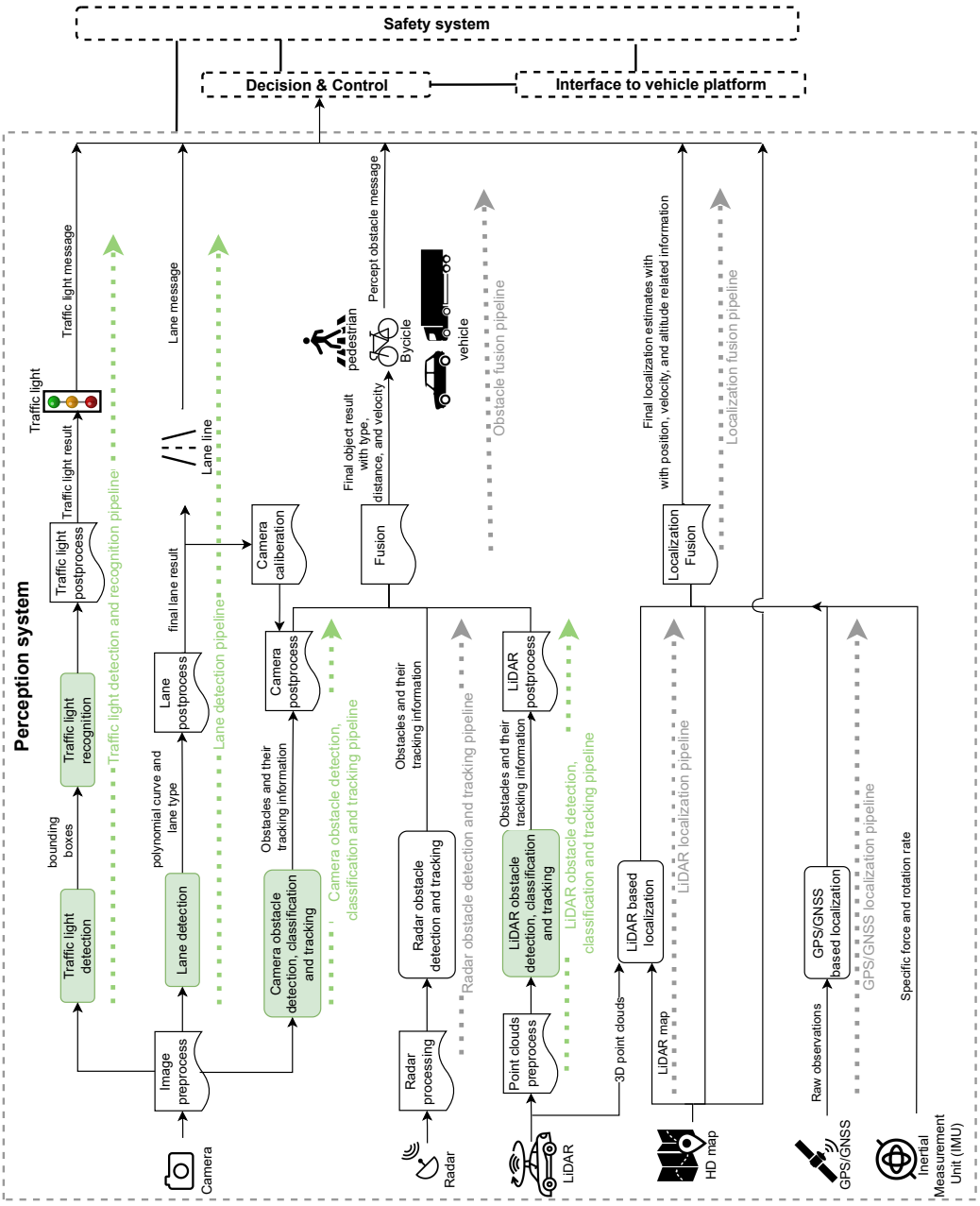


Figure 4.2: Relevant parts of Apollo's architecture workflow.

from these sensors is augmented with details from a High Definition (HD) map. The data from each of these sensors is processed individually for obstacle classification (camera and LiDAR) and obstacle detection and tracking (camera, LiDAR, and radar) as shown in Figure 4.2. The camera is also used for traffic light detection, traffic light color recognition, and lane detection and tracking. The information from the individual object perception and detection sub-systems is further fused to have an overall view of all the objects surrounding the vehicle and allow their tracking. For the self-localization and pose (a combination of the position and orientation of the vehicle) estimation, data from GPS/GNSS, Inertial Measurement Unit (IMU), LiDAR, and HD map are used. Localization is performed individually using data from LiDAR and GPS/GNSS. Further, this data is combined with HD map and IMU data to identify the automated driving vehicle's position, velocity, and altitude-related information. Note that the following details of the architecture are not shown in Figure 4.2 for simplicity. (1) The information from the HD map is used in (a) traffic light detection and recognition pipeline; (b) LiDAR obstacle detection, classification, and tracking pipeline; and (c) radar obstacle detection and tracking pipeline. (2) The output of the localization fusion pipeline is used in (a) radar obstacle detection and tracking pipeline and (b) traffic light detection and recognition pipeline.

This entire suite of sensors and software around them are organized as 5 (kinds of) sensors (camera, radar, LiDAR, GPS/GNSS, and IMU), HD map, and their 9 pipelines that use the data from the sensors (as highlighted in dotted arrows in Figure 4.2). Each of these pipelines forms a module or a cluster of modules in Apollo 7.0.

4.1.2 Operational design domain description

This study is on complete automated driving with no human supervision[9] in a Dutch highway segment. For this level of automation, the current automotive safety standards [5, 6], industry consortiums [16], and regulatory bodies [17] recommend that the operational area of the automated vehicle should be taken into account to identify safety requirements. Moreover, specifying an operational area reduces the complexity and overall set of scenarios for developing and deploying autonomous driving vehicles rather than considering every possible scenario. Such a scoping has been shown to make it feasible to deploy automated driving vehicles without human supervision with current technological limitations [132].

The operational area for this case study is a 3.4-kilometer segment of highway A270 in the Netherlands.[10] We systematically define our operational area based on the best practices outlined by industry consortiums and traffic regulatory bodies [16, 17]. The data

[9] Otherwise known as Level 4 of automation. For a brief overview, refer to `https://www.sae.org/blog/sae-j3016-update`

[10] `https://www.openstreetmap.org/directions?engine=fossgis_osrm_car&route=51.4564%2C5.5408%3B51.4657%2C5.5865#map=15/51.4610/5.5636`

for specification of the operational area are extracted from maps[11], Google Street View[12], and guides from Dutch authorities [129, 133]. Our operational area definition consists of the following six aspects:

1. *Physical infrastructure*, i.e. characteristics of the traffic infrastructure including road types, surfaces, markings, and geometry;
2. *Operational constraints* which include speed limits and traffic conditions;
3. *Objects* that can be present on the road, including signage and types of road users;
4. *Environmental conditions* that include weather, weather-induced road conditions, particulate matter on the road due to weather, and illumination;
5. *Connectivity* including possible (wireless) networking options and data provided via these networks;
6. *Zones* that include different traffic related zone classification.

A detailed specification of the operational area, individual variables considered in each of the above six categories, and their range of values is provided in the replication package [130].

4.2 Safety requirements elicitation

4.2.1 Method

In the automotive domain, methods for safety requirement (also referred to as functional safety requirement) elicitation of software systems is described in two domain-specific safety standards: ISO 26262 [5] and ISO 21448 [6]. ISO 26262 covers the safety requirements relating to the malfunction of components, while ISO 21448 describes the limitations in achieving the intended functionality of ML based components. We use the two standards to derive safety requirements for situations when (a) components of the perception system become non-operational; (b) components are operating as intended, but the output is lost or corrupted before reaching the destination; and (c) limitations arise in achieving the intended functionality of ML based components due to unsuitable weather and illumination conditions.

For the first two cases, safety requirement elicitation methods are described in ISO 26262 standard [5]; and for the last case, in ISO 21448 [6]. A framework combining the two standards for eliciting the safety requirements can be described in three steps: (1) hazard analysis, (2) risk assessment, and (3) safety analysis.

[11] https://www.openstreetmap.org/
[12] https://www.google.com/streetview/

(1) Hazard analysis focuses on identifying potentially hazardous situations to the traffic participants or infrastructure. The hazard analysis step results in system-wide safety goals to prevent harm in those situations. We use the hazard and operability analysis (HAZOP) [44] technique which uses systematic brainstorming to identify such situations. HAZOP identifies all possible situations based on the environment, functions of the automated driving vehicle, and the possible behavior of other traffic participants. Then, the technique associates a situation with possible harm to generate hazardous events using guide words. For example, the harm (otherwise known as a hazardous event) *"does not avoid collision with a decelerating vehicle in front, in the driving lane"* is formed by combining: the guide word *no* with the situation *"avoid collision with a decelerating vehicle in front, in the driving lane"*. We used the guide-words: *no, more, less, as well as, part of, reverse, other than, early, late, before,* and *after*, which are widely used in literature [70, 71]. The situation is a combination of function (*"avoid collision"*), the behavior of traffic participant (*"decelerating vehicle in front"*), and the operational area (*"in the driving lane"*). The latter two parts forming the situation, i.e., (a) all possible situations and (b) behaviors of traffic participants, are directly from the operational area description (detailed in Section 4.1.2). Next, each hazardous event is converted into a system-wide safety goal to prevent, avoid, or reduce its impact.

(2) Risk assessment estimates the risk associated with a safety goal. Since every situation does not lead to the same level of harm, we need to prioritize situations based on their potential for harm. The safety standard's [5] framework proposes four risk levels (*'A'* through *'D'* in increasing order of importance) for a safety goal, also referred to as Automotive Safety Integrity Levels (ASILs). These ASIL levels are identified based on qualitative levels of three parameters: (a) *exposure*, relating to the frequency of occurrence of a hazardous situation [5]; (b) *controllability*, relating to the level of control a vehicle has of the situation [5]; (c) *severity*, relating to the severity of the potential harm in a situation [5]. The safety goals without a reasonable risk are classified as a different risk level, QM (or quality management), and are removed from further consideration. The assumptions we have taken to arrive at a specific risk score are described in Section 4.2.1.

(3) Safety analysis translates the system-wide goals to the requirements on individual components. Broadly, there are two types of approaches for safety analysis: (a) deductive or top-down analysis, where a top-level event (such as a system-wide safety goal) is divided into requirements for lower-level components of the perception system; (b) inductive or bottom-up analysis where the analysis starts from the bottom-level events to identify its possible impact [19].

We use fault tree analysis [42], a deductive analysis technique, to translate system-wide safety goals to the safety goals specific to components (pipelines, sensors, HD map, or safety system). The choice of fault tree analysis is based on its prominence and use in literature in similar contexts [19]. We further subdivide each safety goal (specific to a

component) into requirements for (a) failure of a component and (b) corruption or loss of messages during communication among components. We need two pieces of information to perform fault tree analysis: (1) system-wide safety goals and (2) detailed architecture of the system. The system-wide safety goals resulted from the first step–hazard analysis; and we created the detailed architecture of Apollo as described in Section 4.1.1.

We also consider weather and illumination conditions that can limit individual components in achieving their safety goals for the components that use ML based systems. We use inductive analysis, as specified in the ISO 21448 standard [6], for identifying external conditions (or triggering conditions) that can violate safety goals due to the limitations of ML systems in delivering the intended functionality and translating them into requirements. To conduct this inductive analysis, in addition to safety goals and detailed architecture, we need a third kind of information–the possible weather and illumination conditions applicable to safety goals. This information is derived from the operational area description (see Section 4.1.2).

The result is a list of safety requirements where each requirement is mapped to a (set of) component(s). Note that the ISO 21448 standard also provides a post-design risk evaluation for the safety requirements specific to ML based systems' limitations. Doing so is beyond the scope of this work since this evaluation pertains to the validation and verification (of the measures to make risks due to the limitations of ML based systems tolerable) and not design assessment.

The above steps are performed by the first two authors. The results are compared until an agreement is reached. The process was supervised by the third co-author who is a researcher from industry with more than ten years of experience in the automotive industry and more than 15 years of experience in safety-related and safety-critical systems development and certification according to IEC 61508 [38] and ISO 26262 [5]. Note that the scope of this study is using existing methods to elicit safety requirements. The state-of-the-art in automotive safety requirement elicitation involves considerable manual efforts. Automating and reducing the amount of manual effort is its own research topic and is out of the scope for this study. However, to ensure soundness while using the current method, two authors performed the entire set of steps. For the first step (hazard analysis), which is the one step with the possibility of individual interpretation, we performed an inter-researcher agreement [35]. The result was a kappa score of 1.0 [35] showing an ideal agreement. The ideal agreement might be the result of a clear and extensive definition of operational area and vehicular functions. Going a step further, the entire process was supervised by researchers from both academia and industry (the third and fourth authors), with the researcher from industry who has more than ten years of experience in the automotive industry and more than 15 years of experience in safety-related and safety-critical systems development and certification according to IEC 61508 [38] and ISO 26262 [5]. For future validation and repeatability, we have provided the entire set of intermediate results and sources of information in the replication package [130].

4.2.2 Results

Hazard analysis. We identified a tractable number of scenarios for hazard analysis by combining automated vehicle operations with driving situations. For this study, the operations of the automated vehicle can be divided into two categories: (1) avoid collision with other road users and obstacles, and (2) follow traffic rules in the operational area. Likewise, we partitioned the driving situations into the following four categories: (1) driving in the lane, (2) changing lanes, (3) location-specific behavior for an intersection, and (4) location-specific behavior for a merging point. The scenarios for hazard analysis are then a cross-product of the operations and driving situations.

We used guide words to identify hazardous events for the list of scenarios identified for hazard analysis. An example of a hazardous event is *the automated driving vehicle does not avoid collision with a slower-moving vehicle in its driving lane.* This way, we identified 69 potential hazardous events. Finally, we translated each potential hazardous event into a system-wide safety goal. One such safety goal is *the automated driving vehicle shall avoid collision with obstacles or vehicles in the driving lane.* The data relating to deriving scenarios, hazardous events, and finally, system-wide safety goals is available as a part of the replication package [130].

Risk assessment. Since not all safety goals are equal, we assign a risk score (or ASIL) to each safety goal. We identified risk scores in terms of controllability, severity, and exposure as mentioned before and defined in ISO 26262 standard [5]. To assign a risk score to each safety goal, we make two assumptions: (1) no 'controllability' by a human driver in case of hazard since we focus on fully automated driving; and (2) high severity levels for highway driving speeds [134]. For example, the above-mentioned safety goal was assigned the risk level ASIL D (highest).

Similar safety goals are aggregated to form one safety goal, inheriting the highest ASIL level of the safety goals combined. We identified 18 distinct safety goals, aggregated from the 69 safety goals identified above. The aggregation of safety goals was performed by combining different future or current operations of the vehicle. For example, the safety goals : *"avoid collision in the scenario: decelerating vehicle in front in operational mode: driving in the lane";* and *"avoid collision in the scenario: decelerating vehicle in front in operational mode: changing lanes";* and other similar safety goals are aggregated to – *"avoid collision with an object (obstacle or vehicle) in driving lane in all operational modes".* Further, we excluded three safety goals (since they have ASIL level QM[13]; as discussed in Section 4.2.1) and explored the remaining 15 safety goals in the rest of this study. More details on risk assessment are available in the replication package [18].

[13] ASIL QM–for Quality Management–is the lowest risk level [5]. According to the industry standard ISO 26262, a safety goal with ASIL QM does not require further consideration.

Safety analysis. Using fault tree analysis, we translate the fifteen system-wide safety goals into the safety goals relating to the nine pipelines, five sensor types in Apollo (camera, LiDAR, radar, GPS/GNSS, IMU), and HD map (see Figure 4.2 for details). We map each safety goal to the entire pipeline and the safety system to ensure that the design choices in Apollo that might satisfy our safety goals will be covered in the design assessment part described in Section 4.3. One such pipeline-specific safety goal is: *LiDAR obstacle detection, classification, and tracking shall estimate the correct state of vehicles and other obstacles.*

Next, we converted each safety goal into requirements relating to failure, data corruption, and ML based systems' limitations. For example, two requirements derived from the above-mentioned pipeline-specific safety goal are: *"if any component in LiDAR obstacle detection, classification, and tracking pipeline becomes non-operational, then this failure shall not lead to an incorrect estimation of the state of vehicles or other obstacles"*; and *"if the output of any component in the LiDAR obstacle detection, classification, and tracking pipeline is corrupted or lost, then this corruption or loss shall not lead to an incorrect estimation of the state of vehicles or other obstacles".*

We identified 30 safety requirements relating to failure and data corruption of the different pipelines, as shown in Table 4.2. Details of individual requirements are presented in the replication package [130].

To derive requirements on limitations of ML components, we first identified the pipelines that use ML based systems. Similar to prior work [13], we noticed that out of 9 pipelines, only 4 use ML based solutions (based on analysis of documentation and code). These pipelines are (1) traffic light detection and recognition, (2) lane detection, (3) camera obstacle detection, classification, and tracking, and (4) LiDAR obstacle detection, classification, and tracking (as also shown in Figure 4.2). These four pipelines use two sensors: camera and LiDAR. Therefore, we use safety goals specific to the four pipelines to identify the limitations of ML solutions relating to the weather and illumination conditions in the operational area. Mainly, we map the description of the known limitations of our operational area to the violations of safety goals specific to the four pipelines. The limitations of ML solutions, as identified from the literature on camera and LiDAR, are as follows:

1. *Camera* related limitations: low-illumination conditions [135, 136], illumination conditions rarely captured in training data sets such as dusk and dawn [6], and weather conditions, in particular fog [137], rain [138, 139], snow [140], and strong sunlight [141].

2. *LiDAR* pipelines are not affected by low illumination conditions. However, they are affected by strong sunlight [142] and conditions leading to light (laser) scattering effects which include fog [143], rainy conditions [144], and snow [145].

The requirements for each of the above conditions are identified from the respective safety goals and allocated to the corresponding pipelines. An example requirement allocated to

LiDAR obstacle detection, identification, and tracking pipeline is *"if the performance of LiDAR obstacle detection, classification, and tracking pipeline is deteriorated due to moderate inclement levels of fog, then this deterioration in performance shall not lead to an incorrect estimation of the state of vehicles or other obstacles"*.

The result consists of the 28 requirements as shown in Table 4.2. The details of individual requirements are available in our replication package [130]. A summary of results of the entire safety requirement elicitation is presented in Table 4.3. Note that all our requirement phrasings are based on the industry-specific standard guidelines and literature specific to the automotive domain [5, 6, 146]

4.3 Design assessment

There are many ways to assess the safety requirements for perception system software, including formal verification. To assess the safety requirements for perception system software at the design level, one widely used method is to assess the software using its underlying architecture. However, this solution alone does not work for a perception system consisting of ML components. In addition to architecture, it also requires datasets and ML models to describe them adequately. Excluding these artifacts is not an option since the design decisions for these artifacts can directly impact quality attributes such as safety [147–150]. Therefore, for the safety assessment of perception system software, we study its (1) software architecture and (2) design choices specific to ML based systems.

Table 4.1: ML design decisions related to safety

Design choice [secondary study]	**Objective of the design choice [source papers pointed by secondary studies]**	**Where is it assessed?**
Input data is complete, balanced and well distributed [147]	The dataset used for training (and testing) the ML model shall demonstrate how it covers (corner cases of) the intended application area and qualitative or quantitative representativeness of different categories. [151–154]	Dataset and its related artefacts
Design specification [148]	The (safety specific) properties for which the ML model is designed for shall be specified, for example via formal specification or breaking down ML components into smaller algorithms to work in hierarchical structures [155–157]	ML model related artefacts including scientific paper or documentation

Design choice [secondary study]	Objective of the design choice [source papers pointed by secondary studies]	Where is it assessed?
In-distribution error detectors [148]	Use of mechanisms like (a) run-time prediction error detectors or monitors, (b) prediction for high confidence samples and withholding result otherwise, (c) employment of classification with reject function, and (d) failure prediction through a secondary model, to maintain the safety of the system in case of model failure for instance due to weak representation learning [158–160]	ML model related artefacts including scientific paper or documentation, ML (software) architecture, code, and documentation
Out-of-distribution error detectors [148]	Employment of techniques to detect outliers or out of distribution samples (inputs outside training distribution, for instance, input that were not in the training set or those that the ML model did not learn during the training process), like using an ensemble of leaving-out classifiers [161–163]	ML model and related artefacts including scientific paper or documentation, ML (software) architecture, code, and documentation
Domain generalization [148]	The ML model shall demonstrate its robustness to deviations in the input data distribution in contrast to the training set, for example, through techniques like adversarial domain adaptation and multi-task learning [164–166]	ML model and related artifacts including scientific paper or documentation
Robustness to corruption and perturbations [148]	The ML model shall demonstrate its robustness to natural corruptions (e.g., due to camera lens flairs and snow in contrast to an ideal situation) and perturbations (e.g., elastic deformation due to different viewing angles, occlusions) for instance, by using data augmentation and style transfer [167–169]	ML model and its training related artifacts including scientific paper or documentation
Uncertainty estimation [148]	There shall be uncertainty estimation for ML model, including confidence on its prediction and uncertainty for unknown samples. This is crucial in detecting domain shift and out of distribution samples at run-time using techniques like deep ensemble and Monte Carlo dropout. [170, 171]	ML model and related artefacts including scientific paper or documentation, ML (software) architecture, code, and documentation

Design choice [secondary study]	Objective of the design choice [source papers pointed by secondary studies]	Where is it assessed?
Uncertainty monitoring [150]	There shall be mechanisms for quantifying and monitoring uncertainty. Such mechanisms shall identify degradation of models or silent failures (erroneous outputs despite nominal, fault-less operation). This can also enable detection of domain (distribution) shift and out of distribution samples [172–174]	ML model and related artefacts including scientific paper or documentation, ML (software) architecture, code, and documentation
N-versioning [150]	Rather than using a single ML model, using ensembles of ML models, for example, using an interpretable or rule-based model as back-up, leading to reduced risk of over-fitting, better approximate prediction uncertainty, and facilitate interpretability. [172–176]	Software architecture of the component that include the ML model, ML model and related artefacts including scientific paper or documentation, ML (software) architecture, code, and documentation
Metric monitoring and alerts to detect failure [150]	There shall be mechanisms for monitoring to detect silent failures (errenous outputs despite nominal, fault-less operation) of the ML system [176–178]	Software architecture of the component that include the ML model, ML model and related artefacts including scientific paper or documentation, ML (software) architecture, code, and documentation

4.3.1 Method

The software that forms the perception system of Apollo can be classified into two categories: (1) traditional software, where humans decide on the logic; and (2) ML software, where one of the factors the logic is derived from, is the data. Since the two types of

software are developed differently, their design choices and considerations for safety differ in some aspects. We assess the safety requirements of the perception system by identifying the design decisions using its architecture and complementing it with additional artifacts specific to ML software.

Software architecture design choices:
To identify the design choices of software, we look at its architecture. We look at the smallest units of architecture design choices, called tactics [85]. Tactics are abstract design decisions without an implementation structure that can influence the behavior of a system [85]. An example of a tactic is *diverse redundancy* which is the introduction of redundant systems for detecting or masking failures [92]. Tactics that address the quality attribute safety are called safety tactics.

We use all 13 safety tactics (*heartbeat, simplicity, substitution, sanity check, comparison, replication redundancy, diverse redundancy, condition monitoring, repair, voting, degradation, override* and *barrier* [92]) that have been codified and presented as a framework in prior studies [91, 92] (See Appendix A for more details on individual tactics). Prior studies have shown the use of these safety tactics in the automotive domain for safety assessment [70, 71]. For more details on the framework, we point our readers to the studies by Wu et al. [91] and Preschern et al. [92].

ML design choices:
To the best of our knowledge, no framework exists in the literature that codifies ML design choices that address the quality attribute safety. Therefore, we refer to prior secondary studies that have aggregated the (best) ML design choices. These ML design choices are for different life-cycle stages and have demonstrably direct impact on quality attributes [147–150]. We aggregate the known (best) ML design choices from prior works and curate a list of ML design and related choices to assess the quality attribute safety.

To identify ML design choices, we follow a two-step process. First, we create an aggregated list of design choices specific to ML software. Then, we select design choices specific to our use-case, i.e., relating to safety and applicable in the context of ML software relating to camera, LiDAR, or object/lane/color recognition and tracking.

List of design choices. To create an aggregated list of practices or decisions specific to ML based components, we rely on secondary studies, which are aggregations of primary studies. To identify secondary studies, we search Google Scholar using the following keyword: *"safety" AND "software architecture" AND ("machine learning" OR "artificial intelligence" OR "neural networks") AND ("review" OR "survey").* In this search term, we added *"software architecture"* to improve the signal-to-noise ratio.

Once we identified a list of secondary studies summarizing ML practices (e.g., [147–150]), we shortlisted studies that are (a) the most recent for the most comprehensive list of

practices and (b) have at least one design choice specific to ML systems and particularly safety, and at least one design choice specific to the limitations relating to weather or illumination conditions. We identified 4 secondary studies [147–150] which cumulatively discuss 67 design choices or practices specific to ML based systems.

We applied another level of inclusion criteria to identify ML design choices relevant to safety assessment. These include:

- Applicable to automated driving systems (Apollo's perception system).
- Identifiable from architecture, model, code, dataset-related artifacts, or documentation. For example, practices like neuron coverage testing, fuzz testing, and formal verification cannot be identified from the abovementioned artifacts.
- Applicable to ML software's design stage (like design choices related to ML model or dataset).
- Related to the quality attribute safety.
- Usable for countering limitations caused by weather or illumination conditions.

Note that these inclusion criteria are defined iteratively.

We found 10 ML specific design decisions that are listed in Table 4.1. These design decisions correspond to the choices made relating to the dataset (like ensuring *"input data is complete, balanced and well distributed"*), the neural network model, and other design choices (like *"monitor data quality issues"*) relating to the safety of ML systems.

Assessment

To identify whether the design decisions in the perception system of Apollo fulfill the safety requirements, we use a previously demonstrated method in the automotive domain [70, 71, 85]. This method has two parts. First, identification of the applicable design choices for each safety requirement such that the implementation of a design choice itself or in combination with other design choices, can fulfill the safety requirement. Here we have two categories of design choices (safety tactics and design choices specific to the limitations of ML) and three categories of requirements (failure, data corruption, and limitations to ML). We make a cross-product of the requirements to the selected design choices. The requirements concerning failure and data corruption are crossed with the safety tactics used in the prior studies [70, 71]. The rest of the requirements (regarding limitation to ML systems) are crossed with the design choices specific to ML systems selected in the previous step (refer to Section 4.3.1). Each combination in the cross product is checked for validity, and invalid choices are discarded.

For example, consider the traditional software requirement *"if any component in LiDAR obstacle detection, classification, and tracking pipeline becomes non-operational, then this*

failure shall not lead to an incorrect estimation of the state of vehicles or other obstacles". When crossed with the thirteen tactics, this requirement has thirteen possible choices. Of these thirteen, two invalid choices are *substitution* and *repair*. The *substitution* tactic [14] is invalid in this context because LiDAR pipelines in the automotive industry are still in their initial stages and have not yet reached wide adoption; we do not have any alternate, well-proven option to choose from. The *repair* tactic [15] is invalid since manual intervention is not an option for our use-case of fully automated driving and automatic restore is not applicable in this context. An example of a valid choice is *sanity check* [16] since it is possible to continuously monitor the state and output of the pipeline for implausible outputs or states.

After this step, we have a list of design decisions associated with each safety requirement. Each design decision in the list, either in itself or in combination with other design decisions (from the list), can fulfill the safety requirement.

Next, we look for evidence of whether these safety requirements are fulfilled in the artifacts relating to the perception system software of Apollo. We rely on publicly available artifacts: architecture, code, documentation, dataset descriptions, and scientific papers pointed to by Apollo documentation for our assessment. For efficiency, safety tactics related to failure and data-corruption-related requirements are first checked in the architecture description and documentation. If a safety requirement is not satisfied, we look at the code of specific components for coverage of the requirement. To identify the parts of the code that deal with data corruption or failure, we look for error messages and logging statements in the code of specific components associated with the safety requirement.

For requirements relating to the limitations of ML systems, we first look at the dataset, documentation, and the base paper for design decisions. If a requirement is not satisfied, we analyze the specific parts of the code relating to the requirement. Table 4.1 (third column) presents pointers to which subjects (e.g., documentation, source code) are used to identify the usage of each of the design decisions.

For example, consider the safety requirement *"if the performance of LiDAR obstacle detection, classification, and tracking pipeline is deteriorated due to moderate inclement levels of fog, then this deterioration in performance shall not lead to an incorrect estimation of the state of vehicles or other obstacles"*. One way to satisfy this requirement is *n-versioning* (see 9th design choice in Table 4.1 for details). To identify whether *n-versioning* is used starts with identifying ML models used for the pipeline and then looking at the properties of

[14] The aim of substitution tactics is to *"avoid failures though usage of more reliable components"* [92]. This tactic can be further described as *"components or methods are replaced by other components or methods one has higher confidence in. For hardware and software, this can mean usage of existing components which are well-proven in the safety domain" [92].*

[15] The aim of substitution tactics is to *"bring a failed system back to a state of full functionality"* [92]. This tactic can be further described as *"the full system functionality is manually or automatically restored if a system failure occurs" [92]*

[16] The aim of sanity check tactic is *"detection of implausible system outputs or states"* [92].

these models and how they are trained. The model and its properties can be identified from the code[17], the documentation[18], and associated research articles [179, 180]. According to the articles [179, 180] these models do not use *n-versioning*.

Another way to satisfy this requirement is to have the training and testing data being complete, balanced, and well distributed (see first design decision in Table 4.1) concerning moderate inclement levels of fog. In this context, Apollo has used only the neural network architecture from a few research articles [179, 180] and trained them with custom data. The training data and its source are not available; thus, we cannot make any conclusion about the data. In other cases like *camera obstacle detection, classification, and tracking pipeline*, datasets typically consist of meta-data (and its summary), including time of day and place of capture. The meta-data and summary can be used to identify the distribution of illumination and weather scenarios (the focus of this study) required to assess requirements relating to ML systems. For example, the dataset used for training ML models in the camera obstacle detection, classification, and tracking pipeline is captured from Phoenix, San Francisco, and Mountain View. Therefore the dataset does not contain weather situations like snow and sleet while our operational design domain can. Thus, the dataset does not represent our context's weather and illumination conditions.

We followed a conservative approach of marking requirements to be fulfilled at the design level only if we found conclusive design evidence in the specific components relating to a requirement.

Note that design evidence is not a guarantee that a safety requirement is fulfilled in the final product (similar to passing the testing phase does not show the absence of bugs). Instead, it indicates that the requirement is considered at the design level. Without such consideration, the requirement will not be fulfilled when the design is implemented. In our context, we are looking at the final product architecture and, thus, what exactly is implemented in the final product. Therefore, if the results do not point to any design consideration for a requirement, it shows with high confidence that the final product does not satisfy the specific safety requirement.

Similar to safety requirement elicitation (detailed in Section 4.2), The state-of-practice in design (safety) assessment is still manual effort heavy. Therefore, two authors perform the above processes under the supervision of researchers from academia and industry, as detailed in Section 4.2.1.

For selecting the ML specific design choices, the inclusion criteria and the search terms were defined iteratively. For the selection of the ten ML-design choices and their feasibility for each requirement, an inter-researcher agreement was calculated using Cohen's kappa coefficient [35]. We got a score of 1.0 and 0.84, respectively, indicating an ideal agreement in the first case and a very good agreement in the second case. The ideal agreement in the first

[17] https://github.com/ApolloAuto/apollo/tree/master/modules/perception/lidar
[18] https://github.com/ApolloAuto/apollo/tree/master/modules/perception/lidar#readme

case might be due to the clarity and systematic nature of secondary studies [147, 148, 150]. The second score (0.84) shows (relative) difficulty mapping design choices to ML specific requirements.

Table 4.2: Components and count of associated requirements. Every requirement is assessed in the safety system (refer to Figure 4.2 for details) in addition to the component itself. Further, the requirements relating to sensors (IMU, LiDAR, Radar, Camera, GPS/GNSS) and HD map (last 6 rows in this table) are assessed in all modules that use their output.

Component	Req.s related to failure or data corruption (fulfilled)	Req.s specific to ML (fulfilled)
Traffic light detection and recognition pipeline	2 (2)	8 (8)
Lane detection pipeline	2 (2)	7 (0)
Camera obstacle detection, classification, and tracking pipeline	2 (2)	7 (0)
Radar obstacle detection and tracking pipeline	2 (2)	n/a
LiDAR obstacle detection, classification and tracking pipeline	2 (2)	6 (0)
Obstacle fusion pipeline	2 (2)	n/a
LiDAR localization pipeline	2 (2)	n/a
GPS/GNSS localization pipeline	2 (2)	n/a
Localization fusion pipeline	2 (2)	n/a
HD Map	2 (2)	n/a
IMU	2 (2)	n/a
LiDAR	2 (2)	n/a
Radar	2 (2)	n/a
Camera	2 (2)	n/a
GPS/GNSS	2 (2)	n/a

4.3.2 Results

We identified 58 safety requirements for the different subsystems of the perception system, covering the failure of a module or data corruption (30) and limitations of ML systems in adverse weather and illumination conditions (28). The cross-product of these requirements with 23 design choices relating to safety (thirteen architecture tactics crossed with 30 requirements pertaining to the failure of a component or data corruption, and ten ML specific design choices crossed with 28 requirements relating to ML based systems) led to 698 design choices. After removing infeasible design choices, we had 477 design choices.

A detailed list of the feasible design decisions for each requirement is presented in our replication package [130].

Table 4.3: Summary table

Stage and intermediate or final result	# or status
Safety requirements elicitation	
Hazardous events	69
Aggregated safety goals	18
Discarded safety goals (risk level ASIL QM)	3
Safety requirements	58
on failure or data corruption	30
on limitation of ML components	28
Design assessment	
Feasible design choices	477
on traditional software	225
on ML components	252
Status of safety requirements	38 fulfilled 20 unknown
on traditional software	30 fulfilled
on ML components	8 fulfilled 20 unknown

We search for design choices associated with each requirement in the components related to the requirement (also presented in our replication package). For each safety requirement, we reach one of the following three conclusions: (1) there exists evidence that a requirement is fulfilled; (2) there does not exist evidence that a requirement is fulfilled; and (3) unknown. We reach the second conclusion when despite searching all the associated components, the evidence is non-conclusive. We arrive at the third conclusion if a resource is not found or we cannot comprehend the code or its structure. For example, the dataset or the description of dataset characteristics is required to assess choices related to a dataset used for training an ML model. If the dataset or its characteristics are not specified, we reach the third conclusion.

We found evidence that Apollo's design fulfilled 38 out of 58 safety requirements. We noticed that all the requirements relating to architecture are fulfilled in the design. The status of the rest 20 requirements was concluded to be unknown, and all the 20 requirements were specific to ML components and related explicitly to three pipelines: (1) lane detection;

(2) camera obstacle detection, classification, and tracking; and (3) LiDAR obstacle detection, classification, and tracking pipeline. The only pipeline that contains ML components and is found to satisfy ML specific safety requirements is the traffic light detection and recognition pipeline. We identified 180 design choices for the 20 requirements with unknown status. Out of these 180 choices, 119 were concluded as not used. The other 61 were concluded as unknown primarily due to the non-availability of the dataset or its characteristics and no comments, and unknown structure of the code. An overview of the number of (un-)fulfilled requirements, and related components are shown in Table 4.2. More details on how Apollo's design decisions do (not) fulfill each requirement are available in the replication package [130]. A summary of the results of the entire design assessment is presented in Table 4.3.

4.4 Discussion

This section presents interpretations of our findings, their potential use and implications, the role of the choice of methods, and the applicability of our findings to other contexts.

Interpretation of our findings. It might be obvious to some readers that all requirements related to failure and data corruption are satisfied, especially since this is the seventh (major) version of the Apollo stack. This points out the maturity of the stack from a traditional software safety standpoint. However, the same is not valid for ML based components. Our study shows that 20 out of 28 safety requirements specific to ML systems are not found satisfied in Apollo's design (7.0) [11]. The lack of data relating to ML systems (e.g., specification of datasets on which the ML models are trained and documentation of the ML models and their code) has rendered the decision-making inconclusive, thus making the satisfaction of these requirements unknown. If these requirements are not met, it can point to an acute shortage of research related to the safety assessment at the design level for ML based systems. This corroborates with the literature suggesting that quality attribute safety is not yet one of the highest priorities in developing ML systems [150].

Understanding unfulfilled safety requirements in design is the first step to safety. If design issues are not corrected in time, they transfer to implementation. Since design deficiencies cannot be fixed in implementation, they can cause catastrophe, risking the lives of passengers and other traffic participants. For example, the infamous Uber self-driving car accident leading to the death of a pedestrian was caused by the decision system failing to act after the perception system identified a pedestrian well before safety margins. Also, fixing a design issue later in the product life cycle is orders of magnitude costlier than in the design or early prototype stage.

To the best of our knowledge, this study is the first one in the scientific literature to present the safety design assessment for the perception system of a mature software stack for automated driving in a real-life setting. Nonetheless, based on the insights derived from our study, we suggest the industry to provide their ML models' and datasets'

characteristics, like their representatives in different situations, training and test data, and resultant accuracy. Specifically, our study identifies areas that might need more work that can inform the planning of tech leads and managers. For example, the LiDAR obstacle detection, classification, and tracking pipeline may require more work than the traffic light detection and recognition pipeline. Further, the industry can use our list of requirements and our curated list of practices to generate documentation relating to ML components.

Recommendations & Future directions: This study brings the high amount of human effort required to elicit safety requirements to the limelight. In its current form, each automotive stakeholder that plans to sell an automated driving stack directly or indirectly to an end user has to perform requirements elicitation. Then ideally, safety certification bodies in respective countries have to examine the entire process. One key takeaway that we saw in this case study (which might already be known in the community) is that many steps (e.g., hazard analysis, risk assessment) of requirement elicitation are common irrespective of the underlying stack, given the end functionality (in our context, automated highway driving) is the same. Instead of each entity performing the same steps separately, we recommend all the entities, especially safety certification bodies in respective places, to perform such common steps together and make the results available to all interested parties. Such a practice can not only remove the unnecessary waste of resources but also update those steps consistently in the future. Note that such common steps (due to their very nature as *"common"*) will not affect the exposure of the intellectual property and any related advantages of any of the stakeholders involved. We also believe that vehicle users have the right to an unbiased understanding of the safety of the vehicles' software, especially in automated driving settings.

Another important part we noticed is the lack of clarity and the highly distributed nature of documentation and associated resources. To make a detailed architecture, we (and similar prior research [13]) have to create a detailed architecture using a multitude of resources and code spread across multiple domains (e.g., image recognition, neural networks, localization methods). Yet, many details are missing; in our case leading to the fulfillment status unknown for 20 requirements. This not only hampers the safety requirement elicitation and assessment but also the idea of open sourcing, which is to elicit community participation, and overall under-stability, usability, and maintenance. We strongly recommend updating the missing details (see our replication package for details [130]).

This study is primarily qualitative, while future studies can consider the analysis on the impacts of the obtained results in a quantitative way. Another future direction studies can explore is safety requirement elicitation on other automated driving stacks and how they compare to Apollo. Techniques to reduce the human effort (and the resulting subjectivity) of the requirement elicitation and assessment are another dimension to explore.

Applications. We foresee many applications of our findings for industry, research, education, traffic authorities, and lawmakers. Currently, every company that develops or

uses an automated driving framework for a specific location has to repeat its own safety requirements elicitation due to the current proprietary nature. Then the authorities will need to assess each of them (if this is required for certification for use on the road for automated vehicles). Open sourcing safety requirement elicitation can reduce this rework (by both companies and traffic authorities), attract community participation, make the process more transparent, easier, and reduce cost and effort.

The research community can use our results as a first step to identify weak points in automotive perception systems and identify directions for future research from a safety perspective. For education, including safety in a curriculum might avoid catastrophic events[19] resulting from considering safety as an afterthought.

For lawmakers and traffic authorities, one major challenge is identifying who is responsible in case of a catastrophic event involving an automated driving vehicle: the automotive company, the software suppliers, tool vendors, or the users themselves?[20] A publicly available safety analysis can be the first step to ensure that basic steps are taken to avoid such situations.

Method. As the first study on assessing safety in the design of Apollo, we chose the de-facto method in the automotive domain for safety requirement elicitation and banked on literature for design assessment. However, alternative techniques, for instance, failure mode effect analysis [43] or system-theoretic process analysis [37], can be used instead of fault tree analysis. Future research should validate whether the choice of a method can influence findings and, if so, how.

Generalizability. This study is on a 3.4 km stretch of a Dutch highway and Apollo's perception system. Since the highways in the Netherlands are relatively standardized with minor variations and similar weather and illumination conditions, our findings are more likely to generalize to highways in the Netherlands than a similar study in a higher variability country like the United States of America. We suggest investigating other highway segments before exploring an entire highway-wide safety requirement elicitation. This also means a similar generalization may not hold across Europe or beyond Europe since the traffic environment, traffic rules, weather, and illumination conditions can vary drastically. We also expect similar results if other automated driving frameworks were to be used instead of Apollo. More research is required to test these scenarios.

In retrospect, the validity should improve when independent researchers replicate our work. For reproducibility, data and step-by-step results are publicly available [130].

[19] https://www.bbc.com/news/business-50312340

[20] https://www.theguardian.com/technology/2022/jan/26/self-driving-car-users-should-have-immunity-from-offences-report

4.5 Threats to validity

Construct validity. Many steps in our case study rely on human judgment, which can introduce researcher bias. For instance, many steps in requirement elicitation require brainstorming and manual inspection. While researcher bias remains a valid threat; we tried to mitigate it by using systematic methods and inter-researcher agreements where possible (e.g., using HAZOP for hazard analysis). Two authors performed each step that required manual analysis under the broad supervision of a subject matter expert from the industry. The industry expert (and co-author) has more than 10 years of experience in the automotive industry and more than 15 years of experience in safety-related and safety-critical systems development and certification according to IEC 61508 [38] and ISO 26262 [5].

We identified ML specific design practices from secondary studies. So, if these studies systematically missed a subset of design practices (e.g., linked to their scope), they are missing from our study too. To minimize this threat, we selected the most recent secondary studies for the latest and most comprehensive list of design practices. We also noticed that these studies used systematic and mixed methods approaches, reinforcing our belief that the aggregated list of practices is comprehensive.

Note that we followed the current state of practice in requirement elicitation and architecture assessment, which is human effort intensive. At the same time, we have employed systematic methods and qualitative evaluation to improve reproducibility and soundness. Another way to reduce human effort and improve reproducibility might have been automating the entire process. However, it is out of the scope of this work and a research direction on its own.

Internal Validity. Many steps in our design assessment make assumptions (e.g., assumptions for risk assessment). As long as these assumptions hold, our results are likely accurate. To limit the risk of introducing unjustified assumptions, we only made assumptions that are grounded in literature.

Our design assessment relies on publicly available documents. While we tried to be as comprehensive as possible, the results in this chapter are as sound as the documentation, code structure, error and logging code, and pointers to the base papers.

External validity. Our case study uses Apollo's automated driving framework on a Dutch highway segment. While Apollo is one of the most advanced automated driving frameworks available in the open-source and the highway scenarios we choose are generic, our findings may not generalize. To improve the external validity of our findings, our solution should be tried on other Dutch highway segments, other highways, and automated driving frameworks.

4.6 Related work

There is an industry-wide consensus on the importance of the safety of automated driving systems, especially after the catastrophic uber automated driving vehicle crash, which led to the death of a pedestrian.[20] Nowadays, every manufacturer and software vendor, who tests their vehicles on public roads, releases a safety report for the public [181–183], further acknowledging the relevance of safety. Unfortunately, these reports neither disclose safety requirements nor how they are assessed, making it hard to gauge their usefulness. Our study is an attempt to bring safety assessment into the public domain. Making the safety assessment available publicly will be a first step in showing that the basic steps to avoid potential catastrophic events are taken right from the design stage.

The safety of automated driving systems can be assessed in many stages of product development including design [70, 71], development [184], validation & verification [125–127], and deployment [185]. Currently, the vast majority of literature focus on safety assessment in validation & verification stage including testing [125–127], particularly for ML based systems [128]. While coding standards [184], design patterns [85, 91, 92], and best practices [186] to address safety during the design and development stages of traditional software exists, a similar set of guidelines are still in their inception phase for ML-based systems. Given automated driving systems are being used in highly dynamic settings with proximity to other traffic participants, without operator (human driver) supervision, and in safety-critical settings, their safety assessment at every product life-cycle stage requires immediate attention.

Literature has shown the cost of fixing any issue in software increases exponentially with every product life-cycle stage [187]. The design is likely a better stage to start making automated driving safe. While relatively unexplored, studies on design assessment for safety offered methods to elicit and assess requirements in settings such as connected driving [70, 71, 81]. To the best of our knowledge, the scientific literature on safety assessment in the design of a mature automated driving framework for complete automated driving has not been explored, nor has the design assessment of limitations of ML-systems considering environmental factors [188]. Note that environmental factors, including adverse weather and illumination conditions, have been shown to cause functional limitations for ML systems that process data from various sensors [188]. Building on the prior works, this study presents a design safety assessment of an automated driving system for its use in a Dutch highway segment.

4.7 Conclusions

This chapter presents a case study assessing the safety of the Apollo automated driving framework's perception system in design. We elicited 58 safety requirements to enable automated driving in a Dutch highway segment. For the assessment of safety requirements, we used 23 design choices; thirteen relating to traditional software and the other ten

specific to ML based systems. We found design evidence that 38 out of 58 requirements are met. While all requirements relating to traditional software systems are satisfied, many requirements specific to ML based systems are not found satisfied. This points to the higher maturity of the stack from a safety standpoint of traditional software than ML based software.

To the best of our knowledge, this study is the first study in the scientific literature to present the safety design assessment for the perception system of a mature software stack for automated driving in a real-life setting. Our study opens up a multitude of future research directions, including safety requirement elicitation on other automated driving stacks and their comparison to Apollo, and techniques to reduce the human effort (and the resulting subjectivity) of the requirement elicitation and assessment. For practitioners, our contributions include the parts of Apollo which need more work and possible design choices to consider for closing the safety gap. We have shared our data, including results from its intermediate steps for transparency, replicability, and reusability of our work for research and practice.

5

Painting the Landscape of Automotive Software in GitHub

The automotive industry has transitioned from being an electro-mechanical to a software-intensive industry. A current high-end production vehicle contains 100 million+ lines of code surpassing modern airplanes, the Large Hadron Collider, the Android OS, and Facebook's front-end software, in code size by a huge margin. Today, software companies worldwide, including Apple, Google, Huawei, Baidu, and Sony are reportedly working to bring their vehicles to the road. This chapter ventures into the automotive software landscape in open source, providing a first glimpse into this multi-disciplinary industry with a long history of closed source development. We paint the landscape of automotive software on GitHub by describing its characteristics and development styles.

This chapter is based on
S. Kochanthara, Y. Dajsuren, L. Cleophas, M. van den Brand. Painting the Landscape of Automotive Software in GitHub, MSR'22 [12]

Today, automotive is a software-intensive industry [96, 189]. The latest innovations in this 5 trillion dollar industry[1] (including automated driving, intuitive infotainment, and electrification) depend less on mechanical ingenuity and more on software innovations. In 2020, the software in a car and hardware it runs on is estimated to cost from $4,800 up to $10,650.[2] By 2030, this cost is expected to double to an estimated 50% of the total car cost.[3]

The recent entry of the automotive industry in Open Source Software (OSS) is a landmarking change for an industry primarily driven commercially and dependent heavily on protecting their intellectual property. This exposes the automotive software industry, consisting of original equipment manufacturers (or car makers in short), their different tiers of suppliers, and tool vendors to a wide network of contributors worldwide, in addition to interesting and relevant projects. To understand what exists and what opportunities this landmark change can offer, this study explores the *landscape of automotive software projects in OSS*, as seen on GitHub.

Many studies have explored the landscape of OSS, albeit for different domains. There are studies on AI-ML software [190], software from large tech companies [191], and even specific application domains like video games [192] and bots [193]. To this, we add automotive software with its distinctive and unique blend of non-safety critical, safety critical, and infotainment software, bunched together into a single system. We investigate:

What characterizes automotive software projects in open source?

We explore the following two dimensions:
(1) Categories & characteristics: We identify what types of automotive software projects are open sourced and compare them to each other. We also compare the automotive projects to non-automotive projects. Further, we explore the characteristics of automotive projects (e.g., size and maturity of the field) and their stakeholders (e.g., key players and affiliations).
(2) Software development styles: We investigate different aspects of software development like collaboration (e.g., types of contributors, their contributions and interactions) and contribution style (e.g., independent vs. dependent).

Our analyses are based on ≈600 automotive and a similar count of non-automotive projects on GitHub created in a span of 12 years from 2010 to 2021.
Our main contributions are:

[1] https://www.carsguide.com.au/car-advice/how-many-cars-are-there-in-the-world-70629
[2] https://www.eetimes.com/projections-for-rising-auto-software-cost-for-carmakers/
[3] https://www.statista.com/statistics/277931/automotive-electronics-cost-as-a-share-of-total-car-cost-worldwide/

- A manually curated, first of its kind dataset of actively developed automotive software and their classification along four popular dimensions including safety-critical software and tools [18]. This dataset facilitates the replication of this study and future explorations into automotive software.

- A characterization of automotive software including its temporal trends, popularity, programming languages, user distributions, and development activities.

To the best of our knowledge, this study is the first in presenting the automotive software landscape in open source. Insights presented in this study are relevant for the field growing at a fast pace and yet little is known from a software engineering perspective.

The rest of the chapter is organized as follows: Section 5.1 presents our design choices for data collection and analysis. Section 5.2 and 5.3 present our findings and insights along with our approach to derive these insights. Section 5.4 presents the implication of this study for automotive and software engineering research and practice. We review the threats to validity in Section 5.5, describe the related research in Section 5.6, and conclude the chapter in Section 5.7.

5.1 Study design

Our choice of GitHub for the exploration of the automotive software landscape is motivated by the sheer volume of open source software projects hosted on the platform, and its prevalence worldwide. In 2021 alone, 64 million new repositories were created, with more than 73 million contributors from over 200 countries around the globe and 84% of the Fortune 100 companies using GitHub.[4]

There are three parts to this investigation. First, we define automotive software and propose criteria to distinguish automotive software from general software systems; and criteria to identify general repositories serving as the baseline for comparison. The metadata of the two sets of selected repositories are used for the second and third part. In the second part, we present descriptive statistics of the repositories (in Section 5.2) while in the third part we explore user statistics as well as contribution patterns (in Section 5.3). For the second and third parts, we derive insights from the automotive domain and compare it against the baseline. Particularly, we mine archival data via the GitHub API (using PyGithub - a python wrapper for GitHub API search[5]) for the second part. We further enrich this data with the GHTorrent data [194] for the third part. Generally, our study design takes inspiration from recent landscape studies relating to OSS (e.g., [190, 191]).

[4] https://octoverse.github.com/
[5] https://pygithub.readthedocs.io/en/latest/introduction.html

5.1.1 What is automotive software?

There are many definitions of automotive software prevalent in different scientific communities (e.g., [97, 189, 195]). Some common elements of these definitions are: (a) the software that forms part of a vehicle, (b) the software that interacts with a vehicle via APIs or other similar mechanisms, and (c) the software specifically used for creating (a) and (b) [40, 189, 195, 196]. A more detailed characterization of automotive software is presented in Section 5.2.

5.1.2 Identify automotive software projects

To identify a specific type of software projects on GitHub, conventional methods like topic modelling [197, 198] are found to be inefficient [190]. Another approach uses the 'topics' feature on GitHub.[6] Topics are labels defined by a project or suggested to a project (by GitHub) that can be used to discover a network of similar repositories.[7] Our preliminary manual analysis showed that unlike previous study [190], several automotive repositories did not use GitHub's 'topics' feature. Therefore, in addition to looking at 'topics' to identify repositories, we searched GitHub for specific keywords which if found in the 'README' file are likely to identify an automotive software repository.

To identify automotive software using the 'topics' feature of GitHub, first we defined seed terms. We choose 'automotive', 'automobile', 'drive', 'driving', 'vehicle', 'vehicular', and 'car' as the seed terms. To capture a range of related terms, we transformed the seeds terms to their base terms. For example, 'automo' for automobile and automotive. Likewise, the other keywords became: driv, vehic, and car. Using these base terms, we composed a search string excluding the terms that are not related to automotive software. Examples are google-drive, e-commerce, and device-driver related topics. Our final (4) search queries were:

- *automo,*
- *vehic,*
- *driv NOT driven NOT drives NOT license NOT google-drive NOT linux-driver,*
- *car NOT cart NOT card NOT caro NOT carp NOT care*

Using these search queries we identified topics which collectively defined the search space for automotive software repositories. In total, we identified 2,797 topic labels. We manually analyzed each topic to decide whether it is related to automotive software or not. If a topic label was not informative, we looked at the name and description of the top 10 repositories linked to the topic to make the decision. Ultimately, we identified 286 topics and selected all their linked repositories. A complete list of the topics (along with its search term) is available as a part of our replication package [18].

[6] https://github.blog/2017-01-31-introducing-topics/
[7] https://github.com/topics

Further, to identify relevant repositories that do not use topics, we selected the top five topic results based on repository count (from the 286 topics in the prior step), from each of the four search queries, that are selected in the prior step (yielding a total of 20). For a better signal-to-noise ratio in the search results, we removed the most common terms (e.g., car, cars) which resulted in 12 terms. We searched for these terms in the 'README' file of repositories which do not use 'topic' labels, in order to identify additional repositories. Notably, only up to 50% of the repositories relating to automotive were found using the 'topics' feature and 301 out of 585 selected automotive repositories did not use this feature. Note that in each of the above manual analysis steps, a random sample and borderline cases were analyzed by two researchers independently, to ensure rigor and repeatability.

5.1.3 Selection and elimination criteria

To curate a representative sample of active projects, we apply the following filtering criteria (inspired by [190]):

Size: The size of a repository should be greater than 0 KB.

Popular: Stars and forks are indicators of the popularity of a repository. To collect a representative sample of repositories (and not just the popular ones), we select repositories with at least 5 forks OR 5 stars.

Activity: We use commits as a proxy of development activities and select repositories where the last commit was in 2021, a criterion for selecting actively developed projects.

Data: The repository data should be available via the GitHub API. The above four criteria when applied to the shortlisted software repositories, resulted in a subset of 1981 repositories.

Content: To gauge whether a repository is an automotive software one or not, the first author manually examined the project title, description, and README file based on the following inclusion and exclusion criteria.
Inclusion criteria

- Select automotive-specific software
- Select software that aids in the development of automotive-specific software
- Select software related to on-road vehicles only
- The text is written in English and has a README file

Exclusion criteria

- Repositories that are not automotive related or relate to (automotive) sales and marketing, tutorials, course projects, bachelor and master theses, documentations, data-sets, toy cars, games, traffic infrastructure, maps, and ones that do not directly interact with vehicles.

We adopted a conservative approach for selecting repositories. This means that cases which fall in a grey area were excluded. For the repeatability of the procedure, another researcher with experience in conducting empirical software engineering independently classified a subset of randomly selected repositories (approximately 100) using the above inclusion and exclusion criteria. The inter-rater agreement between the two classifications was 0.83 as calculated using Cohen's Kappa [199] indicating an almost perfect agreement. The two researchers discussed their disagreements until a decision was reached. In the end, we identified 585 active and popular automotive software repositories.

5.1.4 Identify baseline repositories

To compare our insight against a baseline, we needed actively developed repositories that are not automotive-related. Our first choice was reusing the baseline from a related prior study [190]. This dataset, however, had three issues: (1) does not contain recent repositories (created after mid-2019), (2) systematically excludes AI-ML repositories, and (3) represents most popular repositories which are not necessarily representative of general software projects. To mitigate these concerns, we created our baseline with the following characteristics. First, we identified actively developed projects using the same criteria (size, popularity, activity, and data availability) as for the automotive software projects (refer to Section 2.3). The only deviation we made is selecting repositories with five or more stars and forks. This decision was made to mitigate the practical implementation limits of the search API. Then, for each year (from 2010 until 2021), we sub-sampled repositories proportional to the percentage distribution of all the actively developed GitHub repositories over the years, and selected based on most recent activity from each sub-sample. We selected repositories such that their aggregate count is closer to 600 repositories. To avoid overlap with the automotive software, we excluded repositories with the terms *automotive, car,* and *vehicle.* Our resulting dataset had 566 repositories as baseline.

5.1.5 Data analysis

There are two parts to our data analysis: (1) We report descriptive statistics on the selection of automotive and baseline repositories. We describe the types of automotive software systems and how they relate to baseline software systems. This part of our analyses is based on the meta-data extracted using PyGithub. For details, refer to Section 5.2. (2) We offer deeper insights into development styles by combining insights from PyGithub and GHTorrent [194]. Since GHTorrent dataset contained developement data only upto July 2021 (we collected data using PyGitHub in December 2021), some of the repositories from

PyGitHub based data was not available in GHTorrent. Consequently, we were left with 436 out of 585 automotive repositories and 503 out of 565 the baseline repositories. Refer to Section 5.3 for deeper implementation details along with obtained insights.

5.2 Categories and characteristics

This section presents the types of automotive software available on GitHub and their characteristics. First, we introduce the different ways to classify automotive software. The next subsection presents our findings and the distinctive characteristics of automotive software with reference to the comparison set of general software systems. This analysis is based on the 584 automotive repositories (extracted using PyGitHub) created in a span of 12 years between 2010 and 2021. The most recent one was created on 30th December 2021.

5.2.1 Approach

Informally, automotive software can be defined as: (1) the software that runs or interacts with a vehicle; and (2) the tools to support different life cycle stages (e.g., development, validation & verification) of the software that runs or interacts with a vehicle. We refer to the above two categories as *in-vehicle software* and *tools*, respectively.

In-vehicle software: In literature, there are many ways to categorize in-vehicle software. We use the following two schemes:
(1) *Safety critical & safety critical based on application: Safety critical software* is defined as the software that carries out tasks, which if not properly performed, could lead to human injury, death, or harm to the environment [200–205]. During the manual classification of automotive software, we noticed that in addition to safety critical and non-safety critical software systems, there is a third type of software systems: *safety critical based on application.* These software systems can be safety critical depending on the (intended) application context. For instance, a software system for perception is safety critical when used in fully automated driving (i.e., without an active human driver). In this case, any failure, malfunction, or unintended function of the perception software system can lead to a crash, injury to the traffic participants, and harm to its surroundings. The same system when used as a driver-warning system, in which human driver is in charge, can be classified as non-safety critical. In this scenario, the responsibility of maneuvering the vehicle is with the human driver. We classified such software repositories as *safety critical based on application.*
(2) *Broy's classification:* In 2007, Broy et al. [40] classified automotive software into the following 5 categories: (a) *Human Machine Interface (HMI), multimedia, and telematics* related software; (b) *Body/comfort software*, for instance, the software for controlling various aspects of car doors; (c) *Software for safety electronics*, that are hard real-time, discrete event-based software with strict safety requirements; (d) *Powertrain and chassis control*

software, which include control algorithms and software for controlling the engine; and (e) *Infrastructure software*, like software for diagnosis and software updates.

Since 2007, the field of automotive software and software systems in general has evolved. For example, highly accurate image recognition with (relatively) lower computational power was demonstrated using neural networks in 2012 [206]. Automotive software has advanced in perception systems and automated decision making which makes fully automated driving possible. While other aspects of automated driving like drive-by-wire are captured in the current classification, this aspect of perception and decision making, however, is not. We extend Broy's classification by adding a sixth category: (f) *perception and decision software*. Perception and decision software includes any software that contributes to the perception (understanding the surroundings of a vehicle) and decision making (e.g., deciding actuation, steering, and brake), for any level of automated driving (i.e., driver assistance, partially automated, and fully automated).

Tools: Industry standards (e.g., ISO26262 [5]) define multiple stages, such as validation & verification, in the automotive life-cycle. We consider all software repositories that offer tools for one or more stages of an automotive life-cycle, in this category. For brevity, we exclude from this classification, the stages (and hence the corresponding tool) for which we had exactly one repository. The selected repositories fall into the following four categories: (1) tools for development, (2) tools related to simulation or emulation, (3) tools for validation & verification, and (4) tools for diagnostics.

Please note that traditionally tools relating to simulation (and emulation) are considered part of validation & verification. However, with the advent of neural networks, many simulation tools are used in training (developing) neural networks. Therefore, we study these tools separately here.

To ensure a rigorous and repeatable classification of automotive repositories into the above-mentioned categories, the first author and another researcher with experience in empirical software engineering, independently classified a subset of randomly selected repositories and borderline cases.

In the subsequent subsection, we report the distribution of automotive software based on the above classification to offer an overview of the types of automotive software open sourced on GitHub. We continue venturing into these popular and prominent classes of automotive software throughout the chapter.

To characterize automotive software, we report descriptive statistics on automotive software repositories and compare them against the comparison set of non-automotive repositories, also from GitHub. Our analyses highlights four key areas, starting with the *temporal trends and evolution of the repositories* on GitHub. This analysis indicates the maturity and growth of the field. Next, we discuss the *ownership of the automotive software* (users versus organization) indicating the key players of the field and how they are shaping the landscape of automotive software. Along the same lines, we continue exploring *popular*

automotive software in terms of development activities (inferred from fork count) and in general (using stars and subscribers). We conclude with an exploration into the choice of *primary programming languages* used by different categories of automotive software.

5.2.2 Findings

Genesis - The beginning & temporal trends: In 2010, the first still actively developed automotive project, *Veins*, was created in GitHub (26th-April-2010). This vehicular network simulation framework defined the entry of automotive software development into GitHub, marking a turning point for an industry traditionally closed source in the past 50 years of its software use.

Since then more repositories are added each year to a total of 584 (actively developed) automotive software repositories in a span of 12 years. Figure 5.1 presents the percentage distribution of the automotive software repositories (based on their creation year) with reference to the total actively developed repositories in GitHub. The temporal trend suggests that from 2018 to 2019 the percentage growth of automotive software has doubled which again doubled from 2019 to 2020. In comparison to the actively developed repositories across GitHub (which peaked in 2014), automotive software is still in its infancy and expected to grow in the future.

Origin & temporal trends

Veins - *a simulation tool, is the first automotive software repository created by a user in 2010 that is still actively developed. This study witnesses open source automotive software boom in its infancy.*

The dawn of new opportunities - Ownership: In 2010, no organization was developing any automotive software in GitHub that is still actively developed. In the next 6 years (2011-2016), small organizations, enthusiast groups, and non-profits organizations ventured into open source; owning 14 software projects. The first organization owned (still actively developed) project is *Open-Vehicle-Monitoring-System*, an in-vehicle software from the enthusiast group Open Vehicles.

The year 2017 marked the entry of big players to open source. This year Baidu, the Chinese search engine company, created project *Apollo* - a full software stack for fully automated driving. The other big players, namely, Amazon, Intel, Microsoft, and Udacity joined soon after, each of which open sourced one of their tools to build automated driving solutions. In our data set, one in three (or 194 out of 584) automotive software repositories is owned by an organization. Cumulatively, we are looking at 163 organizations and 343 users owning at least one repository.

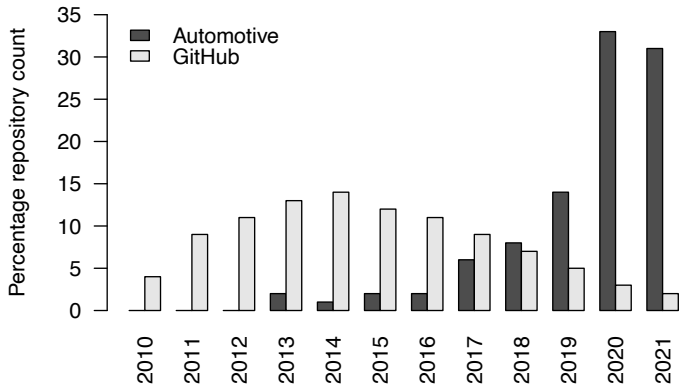


Figure 5.1: Temporal evolution of actively developed automotive software repositories with reference to all actively developed repositories in GitHub, between 2010 and 2021.

The development of automotive OSS today is spearheaded by tool vendors, academic, and industrial research labs with more than 30 repositories owned by academic research groups.

The only car maker in automotive software on GitHub is Toyota with two repositories, a tool and an in-vehicle software. Please note that we might have missed the different tiers of suppliers to car makers since there is no straightforward way to identify suppliers from the GitHub meta-data.

Currently, the top 5 organizations working on automotive software (in terms of repository count) are: VITA lab at EPFL (5 repositories), LG Silicon Valley Lab (4), MathWorks Open Source and Community Projects (4), AutonomouStuff (3), and CARLA (3).

Repository ownership

One in three automotive software repositories is owned by an organization. The field witnesses high participation from academic and industrial (tool vendors) research labs with only one car maker (Toyota) at the forefront.

In the catbird seat - Popularity: The top 5 popular automotive projects from organizations based on the three indicators of popularity (stars, forks, and subscribers; see Table 5.1) are simulation tools (4 in count), followed by in-vehicle software on perception and decision systems (fizyr/keras-retinanet) and the automated driving stack Apollo.

The popular user repositories on automotive software are more diverse, including both in-vehicle software and tools. The in-vehicle software in the top 5 relate to (a) HMI

and telematics and (b) perception and decision. The tools in the top 5 relates to (a) the development of perception and decision-related software and (b) diagnostics.

Figure 5.2 presents the distribution of stars, forks, and subscribers across automotive software along with the baseline repositories. Generally, projects have more stars, than forks and subscribers. This is the same for automotive and baseline repositories. Notably, automotive software is far less popular than the baseline software systems, further reinforcing the notion of infancy of the field. The differences in the distribution among the automotive and baseline repositories are statistically significant as calculated using the Mann-Whitney-Wilcoxon Test [207], a non-parametric test for two independent data samples, calculated at p-value<0.05. The median numbers of stars, forks and subscribers for automotive repositories are 24, 9, and 4 respectively, while the median for baseline repositories are 297, 121, and 36, respectively. Here, we would like to remind the readers that despite our attempts at selecting a representative sample of projects as baseline, our dataset might be somewhat biased towards more popular repositories (see threats to validity in Section 5.5 for details), skewing the distribution further. Even within automotive repositories, organization-owned repositories is at least twice as popular as user-owned software projects.

Popularity

Automotive software as a field is less popular than general software on GitHub. Apollo, Baidu's automated driving software stack, is currently the most popular automotive repository. Generally, organization-owned software projects are twice as popular as user-owned projects.

Genera - Types of automotive software: Broadly, there is an abundance of in-vehicle software (375) in comparison to tools (233). A detailed distribution of the types of automotive software (both in-vehicle software and tools) is presented in Table 5.2. Note that a repository can belong to multiple categories. Therefore, the sum of the repository count in all the categories can be greater than the actual repository count. More details on the classification of individual repositories is available in the replication package [18].

Within in-vehicle software, most repositories relate to perception and decision related software. Notably, Broy's classification [40] for in-vehicle software is a small part (108 out of 375) of the whole. Of these 108, HMI (71) and infrastructure (38) are the top categories, and are primarily developed by users.

The development of safety-critical software in open source is still in its initial stages (21 repositories). Although, many software repositories belong to safety-critical based on application category (approximately 60% of in-vehicle software). Most of the safety-critical software relates to perception and decision-based software intended for use in fully

Table 5.1: Top 5 popular organization and user repositories (based on subscribers, forks, and stars) and their count

	Organization	User
	Top 5 repositories based on subscriber count	
1	ApolloAuto/apollo (1103)	stanleyhuangyc/ArduinoOBD (175)
2	microsoft/AirSim (597)	timdorr/tesla-api (119)
3	carla-simulator/carla (236)	Smorodov/Multitarget-tracker (110)
4	udacity/self-driving-car-sim (231)	cedricp/ddt4all (81)
5	autoas/as (145)	fr3ts0n/AndrOBD (68)
	Top 5 repositories based on fork count	
1	ApolloAuto/apollo (8013)	MaybeShewill-CV/lanenet-lane-detection (772)
2	microsoft/AirSim (3557)	Smorodov/Multitarget-tracker (569)
3	carla-simulator/carla (2128)	stanleyhuangyc/ArduinoOBD (486)
4	fizyr/keras-retinanet (1964)	timdorr/tesla-api (474)
5	udacity/self-driving-car-sim (1414)	karlkurzer/path_planner (355)
	Top 5 repositories based on star count	
1	ApolloAuto/apollo (19954)	MaybeShewill-CV/lanenet-lane-detection (1707)
2	microsoft/AirSim (12590)	Smorodov/Multitarget-tracker (1636)
3	carla-simulator/carla (7100)	timdorr/tesla-api (1549)
4	fizyr/keras-retinanet (4252)	poodarchu/Det3D (1220)
5	udacity/self-driving-car-sim (3595)	yangyanli/PointCNN (1200)

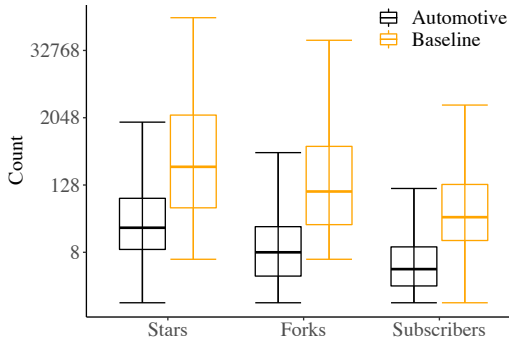


Figure 5.2: Distribution of the popularity of automotive repositories (in terms of stars, forks, and subscribers) with reference to the comparison set (outliers removed)

automated driving systems. These include neural-network based semantic segmentation, path planning, and object, pedestrian, and intent detection related software.

Table 5.2: Types of automotive software on GitHub and their distribution.

Category	Org	User	Total
In-vehicle software	**97**	**278**	**375**
Safety-critical	11	10	21
Safety-critical based on application	57	167	224
Extended Broy's classification			
HMI, multimedia, & telematics	17	54	71
Body/comfort software	10	8	18
Software for safety electronics	8	4	12
Power train and chassis control software	10	7	17
Infrastructure software	9	29	38
All Broy's categories combined	23	85	108
Perception and decision software	68	180	248
Tools	**100**	**133**	**233**
For development	32	31	63
For validation & verification	18	22	40
Related to Simulation (and emulation)	48	52	100
For diagnostics	5	28	33

The category 'tools' in automotive software is dominated by simulators and related software. This is evident in the top five automotive software from industry (see Table 5.1), three of which are simulators. For tools overall, there is near to equal ownership from users (100) and organizations (133). We see a similar trend in the ownership of *development tools* (32 from organization and 31 from users) and *validation & verification tools* (18 from organizations and 22 from users). The only exception is *diagnostic tools* which are five times more prominent among users (28) than organizations (5). Other than the above, we also notice a small number of automotive software repositories (32) relating to (driver) safety (like drowsiness detection) and security (tools for security testing or in-vehicle software for the security of the vehicle).

Types of automotive software

The most popular type of automotive software developed open-source is in-vehicle software *(375 repositories) followed by* tools *(233 repositories). Within in-vehicle software, perception and decision software are most popular while in tools, simulations are prominent. Traditional vehicle software and safety critical software are underrepresented in open source.*

Two worlds; two languages - Languages: Automotive software is developed in 33 primary languages and 96 languages when considering all the languages for development. The most popular programming language is Python with 291 projects using it as a primary language and up to 415 projects using it as one of the languages. The top 5 primary programming languages are Python (291), C++ (98), C (33), Jupyter Notebook (33), and MATLAB (30). Technically Jupyter Notebook is not a programming languages, rather a blend of text and code. We do not make any assumptions in the programming languages used inside the Notebooks rather considered them according to the tagging by GitHub. Organizations generally prefer (based on repository count) Python (82), C++ (37), C (16), MATLAB (10), and Jupyter Notebook (7) for their projects. Notably, users also prefer the same languages but in slightly different order: Python (209), C++ (61), Jupyter Notebook (26), MATLAB (20), and C (17). Most safety critical software is written in C++ (14), followed by Python (4), MATLAB (1), and C (1).

The distribution of programming languages across projects shows a shift from MATLAB as preferred language of development [208] to Python. Likewise, traditionally most safety critical software were developed in C or Ada which has now shifted to C++ in GitHub [209].

Languages

The preferred language for open source automotive software development has shifted to Python (291 repositories) from MATLAB (30 repositories). Similarly, safety critical software development has moved from C or Ada to C++.

5.3 Software development style

Building on the insights derived in the previous section, this section delves into the user distribution, types of development activities, and the choice of development models in automotive software. We compare it against the baseline to understand the unique characteristics of automotive software, if any. Note that since this analysis combines data acquired using PyGithub (in December 2021) with GHTorrent's data (data available until

July 2021), we missed repositories which do not exist on GHTorrent. Further, depending on the development activities of individual repositories and missing data in GHTorrent dataset, the total count of the repositories may vary across different analyses.

5.3.1 Approach

User distribution: In this section, we explore the types and distribution of users across projects. We study two types of users: external and internal [190], based on their activities in automotive software. *Internal users* contribute directly to the development of a project by making changes to the actual software (commits) and moderating the decision to include/exclude the proposed changes (like merging and closing pull requests and closing issues). *External users*, on the other hand, contribute indirectly by requesting features, reporting issues, and commenting. We believe that investigating the distribution of internal and external users across projects indicates how a community works. For deeper insights, we also explore changes in contribution patterns, if any, across organization and user projects.

A natural next step to understand developer contribution and collaboration patterns is to examine developer roles (e.g., maintainer, or reporter) and their distribution. However, given the small community size and limited development activities, it is infeasible to offer meaningful insights and conclusive statistical analyses. Therefore, we do not report collaboration patterns.

Development activities: Development activities on GitHub can be broadly classified into commits, issues, and pull requests. Issue events indicate participation from the broader user base (beyond contributors) requesting additional features or indicating problems. Participation in issue events indicate how the users of the software interact with developers, influencing its development. The next group of development activities are pull requests which indicate a relatively stronger influence on the software by proposing changes for inclusion into the software system or its associated artifacts. These activities log the decisions to include or exclude proposed changes. Finally, a commit is an even more involved activity dealing with the technical aspects of creating desired changes in the software.[8] Here too, we explore whether project ownership influences the development activity patterns across projects.

Development models & autonomy: Finally, we analyze the choice of development model in automotive software. There are two types of development models in GitHub: (1) shared repository model and (2) fork & pull model.[9] The two models are different in the level of autonomy of contributors. In the shared repository model, an author can merge their

[8] https://docs.github.com/en/get-started/quickstart/github-flow

[9] https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/getting-started/about-collaborative-development-models

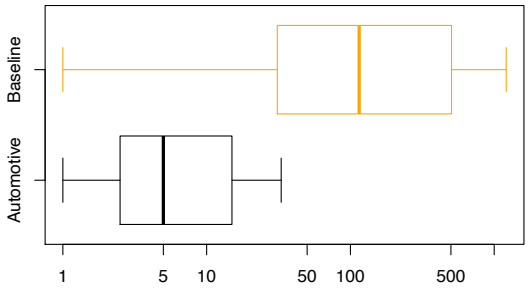


Figure 5.3: User distributions across repositories compared with baseline (outliers omitted)

proposed code changes themselves, indicating their autonomy. The fork & pull model implies that the changes proposed by an author are reviewed by a maintainer. In this model, an author is dependent on the actions of a reviewer for the decision to include or exclude the proposed changes. To study the level of autonomy or dependence in a project, we aggregate the distribution of pull requests and commits for which the author merged the changes (self-merge) versus other contributor (other-merge). We refer to the projects with more self-merges as practicing a shared-repository model and fork and pull model otherwise.

5.3.2 Findings

The real stars - Users & their distribution: All the automotive repositories cumulatively have 15,260 unique users where as the baseline set of repositories have 439,032 unique users. In automotive software, the median count of users per repository is 5 while 115 for general projects from our baseline (refer to Figure 5.3 for distribution). The two distributions are significantly different as calculated using the Mann-Whitney-Wilcoxon test at p-value<0.05. Here again, we warn our readers that our baseline is somewhat skewed towards actively developed and popular software systems. Therefore, the differences may appear larger than they are. Note that our identification of individual contributors relies on unique user identifiers from GitHub. However, one individual can have several unique identities [210]. Consequently, the actual number of users might be lower than the reported count.

Looking at the distribution of users in automotive software, each project has a median count of 3 internal and 5 external users. When we further segregated the user distribution on the ownership type (see Figure 5.4), we observe that organizations have more users per repository. The two distributions are different as calculated using the Mann-Whitney-Wilcoxon test at p-value<0.05. Organizations recorded a median of 6 internal and 8 external users in comparison to 3 and 5 respectively for the user owned automotive repositories.

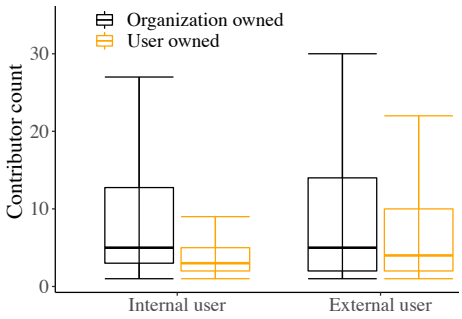


Figure 5.4: Internal and external users across automotive repositories owned by users and organizations (outliers omitted)

User distribution across repositories

Open source automotive software has a small developer community with a median of 5 users per repository. Notably organization repositories solicit more participation internally and externally in comparison to the user repositories.

Abiogenesis - Development activities: We notice that most common development activities (based on the median count of activities) in automotive software are in the form of commits (32), followed by issues (9), and then pull requests (6). We further investigated the distribution of development activities based on the ownership and the types of automotive software.

Figure 5.5 presents the development activities of automotive software repositories owned by organizations in comparison with the repositories owned by users. Generally, organization projects have more development activities (in terms of median) than user projects do, although there are a few user projects where the commit activity level matches that of the organizational repositories. This indicates variability among user projects with extremes in the distribution of development activities. Please note that the two distributions are different as calculated using the Mann-Whitney-Wilcoxon test (p-value<0.05).

A comparison of the development activities across in-vehicle software and tools is shown in Figure 5.6. Here, the development activities across issues (p-value = 0.29) and pull requests (p-value = 0.40) are comparable with major differences in the commits (p-value = 0.0013). These p-values are calculated using the Mann-Whitney-Wilcoxon test such that a p-value<0.05 indicates differences in the distribution and no differences otherwise. We notice that there are more commit activities in tools in comparison to in-vehicle software. We believe that this difference is attributed to the higher participation of tool vendor

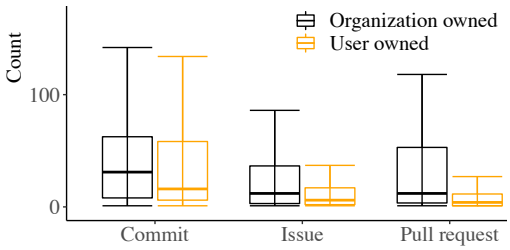


Figure 5.5: Development activities in organization owned automotive repositories versus user owned repositories (outliers omitted)

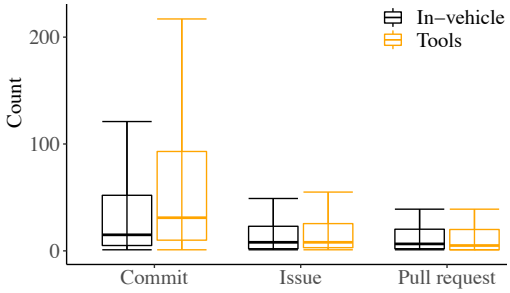


Figure 5.6: Development activities for in-vehicle software versus tools (outliers omitted)

organizations in automotive software. To remind, there are 100 out of 233 organization owned repositories in the tools category versus 97 out of 375 in the in-vehicle software category.

We also explored the distribution of development activities in perception and decision related software, which form a majority of the in-vehicle software (with sizable number of repositories for statistical comparison) with respect to the development activities with the rest of repositories in in-vehicle software. Figure 5.7 shows that traditional software is more actively developed in terms of commits, issues, and pull requests than perception and decision related software. The differences in the distribution of development activities on repositories belonging to all the Broy's categories combined versus repositories belonging to perception and decision software, are significantly different as measured using the Mann-Whitney-Wilcoxon test at p-value<0.05. This means that while perception and decision related software are more in count, they have fewer development activities than traditional software.

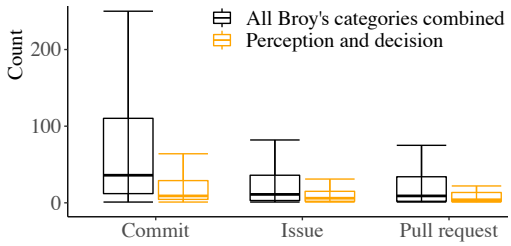


Figure 5.7: Development activities for perception and decision related in-vehicle software versus other in-vehicle software (outliers omitted)

Development activities

The organization-owned repositories attract more contribution across all categories. However, there are smaller number of user owned repositories matching the level of development activities as in organization owned repositories. Tools are more actively developed than in-vehicle software. Within in-vehicle software, traditional in-vehicle software is more actively developed than perception related software.

Guardians - Development model: We classified each project as a shared repository or fork-and-pull development model by looking at their collaboration patterns. We noticed that while 41 repositories followed the shared repository model, 182 repositories practice the fork-and-pull model. This indicates that fewer projects are autonomously developed and a majority of the development teams (contributors of a repository) do not have autonomy. This observation matches the baseline with fork-and-pull being the most prominent model and also prior work on other software sub-communities [190].

Development model

Automotive software—despite being a small development community—mostly follows the fork-and-pull model for its development activities.

5.4 Implications

While automotive open source software is still in its infancy (in comparison to our baseline), the field is 584 repositories and 15,260 contributors strong. With industry players entering the field and choosing to open source their projects, we believe that automotive software

is the next promising area with an expected multi-fold growth in the next decade. In this section, we discuss how the insights from this study can be applied to research and practice.

5.4.1 Research

Academia is a prominent contributor of automotive software in GitHub with one of the top 5 organizations in terms of count of repositories owned. We also identified 108 repositories linked to scientific articles in automotive software. Given the close link of automotive software and academic research, in this section, we present the ways in which our study can inspire future research.

Manually curated automotive software. To the best of our knowledge this chapter is the first in presenting a manually curated dataset of automotive software in open source. We have classified each repository using four categorization schemes namely (1) safety critical, (2) safety-critical based on application, (3) Broy's classification extended with perception and decision related software, and (4) tools. One immediate future research direction is strengthening the classification with inputs from experts in the automotive and related domains. Also, our manually curated dataset along with the classification can be used (in training and testing) to automate the process of identifying and classifying automotive software by automated (algorithmic) approaches. Further, we believe that future research can use this classification for the characterization of automotive software and in-depth explorations into prominent software systems.

Motivation to open source. Relating to the types of automotive software open-sourced by companies, we noticed that organizations mostly open source their tools and some experimental projects. It will be interesting to see what types of companies come to open source and their motivation (similar to a recent study exploring motivations of Chinese companies to open source [191]). While our results show that one in three automotive software repositories is owned by an organization, it is possible that at least some of the GitHub projects in large organizations might have started as personal projects because there was not yet a company policy on how to open source projects. Given the entry of organizations to open source in the automotive domain is still in its infancy, it is possible that some individual projects might actually belong to organizations and the transfer of ownership to the organization has yet to be made. This phenomenon in itself and the subsequent changes in our results form another direction to explore for future research.

Safety-critical. Most automotive software in open source relates to perception and decision and tools. These software systems can be safety-critical depending on the application. Given the limited attention to safety-critical software in open source, it remains a future work to see whether the current and future automotive software in these categories are developed and/or tested according to safety critical standards. Also, our study forms a guide and baseline for future studies on open source software in (other) safety critical domains.

Multi-disciplinary software versus general software. Automotive software developed in GitHub is multi-disciplinary in nature. We observed repositories that model vehicle dynamics;[10] develop firmware and drivers for sensors like LiDARs and Camera; develop algorithms for perception and motion control; and repositories on complete operating systems integrating the above. It will be interesting to see differences in the contribution and collaboration patterns of such multi-disciplinary software projects in comparison to general software systems.

5.4.2 Practice

This study shows that the industry is more interested in open sourcing tools (43% or|-ganization-owned) in comparison to in-vehicle software (26%). Three possible explanations for this observation are: (1) the revenue stream for the tools is dominated by car-makers or their suppliers who will pay for their support irrespective of open-sourcing; (2) tools might be less intellectual property intensive than in-vehicle software and thus easier to open source without losing the edge to competition; and (3) in-vehicle software typically runs on less-standardized hardware (application-specific embedded hardware for various in-vehicle functionality) than tools. These explanations however, cannot be derived from the data used in this study and need to be validated.

In addition, our list of automotive repositories and their categorization can aid future research into identifying the stakeholders in each category, their motivations to participate in open source, and other domains where these tools are used. Our study also provides the first list of automotive software tools and in-vehicle software in open source, with the former available for practitioners to use (and contribute to) and the latter to learn from. Further, our data and insights can be used to identify (a) software for reuse, (b) attract talent and/or increase the adoption of software and standards, and (c) new directions, companies, and trends in the automotive domain. Three potential implications of our findings for practice are discussed below.

Language of choice. Automotive software in open source is now developed mostly in Python, replacing MATLAB as reported in prior studies (see Table 12 in [208]). A similar trend of C++ domination replacing C or Ada for safety critical software development [209] is seen. Our advice to the readers interested in venturing into automotive software is to consider these findings while choosing programming languages.

Companies. Prior study has shown that one reason for companies to open source their software is to attract talent and internationalization [191]. We believe that start-up automotive companies can benefit by open sourcing their projects.

Safety certification & car makers. Safety certification of vehicles is obligatory to allow them on road. The current trend of more software dependent functionalities in vehicles is challenging for certification bodies. These bodies can benefit from open software stacks.

[10] `https://github.com/TUMFTM/velocity_optimization`

Such a change can subsequently encourage car makers to contribute to open source software. Our study can be used by the certification bodies to get insights on the characteristics of automotive software developed in open source. To car makers our study offers trends in the open source automotive software.

5.5 Threats to validity

Construct Validity: There are threats to the representativeness of the automotive software systems selected for analyses. To mitigate this concern, we identified automotive software repositories using two approaches (using topics and keyword search in README files) and adopted best practices for selecting actively developed software systems [211–213]. That being said, we might have systematically missed the repositories that do not use the search terms, uses different topic labels, misses README file, or a meaningful description. Also, in case of doubt, we discarded a repository, i.e. we followed a conservative approach. The same threat applies to the categories of automotive software systems presented in Section 5.2.

For counting unique contributors, we used their GitHub identifier and counted every user with a distinct identifier as a unique user. However, prior studies have shown that one individual can have several identities [210], thus the actual number of users might be lower than the reported count.

Our baseline set of repositories might be skewed towards more popular repositories. For representativeness, we sub-sampled repositories based on the number of actively developed repositories created every year. However, given the smaller size of automotive repositories as compared to the total number of actively developed repositories in GitHub, the random selection from each sub-sample based on recent activity might have resulted in the selection of popular repositories as the baseline.

Another threat is the introduction of researcher bias in the manual selection of repositories and the classification of the automotive software. While these threats cannot be eliminated, we tried to minimize them by (a) clearly documenting the inclusion-exclusion criteria for the selection of repositories, and (b) using an independent rater for a subset of the repositories. For the classification of software, we borrowed the definitions from literature for reference.

Our insights into development styles rest on the GHTorrent dataset. The activities that are not present in the dataset are systematically excluded from our analyses [214].

External Validity: This chapter is based on the publicly available software repositories on Github. While GitHub is a popular and widely used platform, there are other platforms (e.g., Gerrit and Phabricator) with their distinctive characteristics, private projects hosted on GitHub, and closed source systems. They might add a different perspective to the automotive software landscape. We leave it to future research to pick these topics to improve the generalizability of the findings here.

5.6 Related work

Related studies around automotive software landscape in GitHub can be divided into two parts. The first part focuses on the literature on automotive software and its engineering. In the second part we explore studies offering characterization of other software communities.

5.6.1 Automotive software

There are many studies on automotive software, exploring the different dimensions of the topic. Some areas focused in the last five years (as identified using Google Scholar search) include automotive software architecture [70–72, 96, 215], AI-based solutions [216, 217], model-based solutions [218–220] and blockchain [221]. These studies touch on aspects such as complexity [222], safety [222], security [220, 223], privacy [221], and testing [223, 224] that are relevant for automotive software.

Another line of research on automotive software is in terms of their development and development processes [96, 97, 225]. These studies focus on the different steps in automotive software development [96, 97] and applicability of process models like agile development [225] to automotive industry. That said, to the best of our knowledge, there is little to no study characterizing automotive software development and its process in terms of its development activities (like pull requests, issues, and commits) on closed or open source software systems.

5.6.2 Non-automotive software landscape

Even though the software development process or its characterization is not studied for automotive domain, characterization of other software engineering communities has been presented in literature. The software engineering communities whose characteristics are explored in the past can be classified based on application domain [190, 192, 193] and those based on other factors like geography and closed source [191, 191]. We present four studies that have explored software landscape from different perspectives and inspired our study.

The most recent exploration is on the open source software systems developed by large Chinese technology companies namely Baidu, Alibaba, and Tencent [191]. Unlike the open source software studied in general, this exploration is regional. It presents a characterization of open source software developed by Chinese technology companies, their objectives for open sourcing, and a comparison to other software systems [191].

The second study explores a decade of ML and AI software systems developed in open source [190]. The study characterizes the trend of ML/AI evolution in addition to their collaboration and autonomy, and contrasts it against the general software systems [190]. These two studies are our primary inspirations.

Another study characterizes video game development and how it is different from traditional software development [192]. Based on interviews, the study identifies differences between the two types of software system and how researcher can help [192].

Finally, studies on bots explore its use, and how these special software systems can help the development of other software systems [193].

Along these lines, this study offers an exploration into the landscape of open source automotive software projects. Open sourcing is a recent phenomina in automotive industry as shown in the temporal trends in Section 5.2. Taking inspiration from the previous studies and combining elements from many sources, we quantitatively analyze the repository data. We hope that like the previous studies, the findings from our study inspires future research and improve the state of automotive software development.

5.7 Conclusions

This chapter presents a landscape of automotive software projects publicly available on GitHub. We identified and categorized ≈600 automotive repositories grounded in definitions from literature and well-defined empirical methods. We also identified a similar number of non-automotive projects for comparison. We analyzed the origin, temporal trends, key players, popularity of projects, languages for development, user distribution across repositories, and development activities. We also present, a first of its kind, manually curated dataset of automotive projects and a comparison set of non-automotive projects, for replication and future research.

For an industry traditionally being in closed source in its half a century history of software use, open sourcing software projects marks a landmark change. This chapter shows that automotive domain is undergoing a shift in multiple dimensions including the prevalence of automated driving software development, change in preferred language from MATLAB to Python, and entry of software companies and startups to the domain.

We foresee that the recent developments in software engineering, that enables automated driving, will further accelerate open source automotive software development. We believe that the software stacks for automated driving will benefit from perception and decision software currently developed in open source. Since these systems are developed independent of car makers, involving the open source community for the acceleration of their development, is a logical step.

6

Conclusions

In this chapter, we revisit the research questions presented in Section 1.3 and summarize the main contributions of this thesis. This thesis discusses how to ensure safety in the requirement elicitation and design stages of automotive systems and software, focusing on automated and connected driving systems. The thesis also discovered a new landscape shift in the automotive industry with the open-source development of automotive software and presented the first characterization of this trend. In the rest of this section, we present the research questions and corresponding conclusions.

> ***RQ 1.1:*** *What processes are used for or applicable to safety requirement elicitation in the automotive domain?*

> ***RQ 1.2:*** *What techniques are used for safety requirement elicitation in the automotive domain?*

We addressed these two research questions in Chapter 2. We characterized the safety requirement elicitation in the automotive domain through a systematic literature review. We taxonomized the processes and techniques (techniques are different alternatives to conduct individual steps in processes) creating an overview of the current landscape of safety requirement elicitation. This systematic literature review was based on 102 primary studies published between 2014 and 2020.

We identified nine distinct safety requirement elicitation processes and 38 distinct techniques. We observed that the process outlined in the ISO 26262 standard forms the basis for the automotive industry. Other processes are proposed to complement, extend, or

replace the process outlined in the standard. Chapter 2 offered an overview, comparison, and taxonomy of processes and techniques for safety requirement elicitation in the automotive domain. Based on this information, temporal adoption trend, usage context, and scope, we presented research gaps and discussed upcoming domain trends.

RQ 2.1: *How to derive safety requirements for connected driving?*

RQ 2.2: *How to assess safety requirements in the software architecture of connected driving vehicles?*

Chapter 3 investigated whether the architecture of a single-vehicle meets the functional safety requirements for cooperative driving. We proposed a method to ensure that an automotive architecture is functionally safe to operate in given scenarios. The proposed method derives functional safety requirements for a cooperative driving scenario and checks whether they are fulfilled in the technical architecture of a vehicle. The method combines methods adapted from the safety engineering and software architecture domains. We show the usability of our method for a cooperative driving scenario, platooning, on an academic prototype; and how this resulted in uncovering functional safety requirements not fulfilled by the software architecture. Our method is motivated by and reinforces the notion that functional safety should not be an afterthought in the design of automotive architectures, but that it should rather be used for defining the architecture of the automotive system.

RQ 3.1: *What safety requirements shall be fulfilled by a vehicle's perception system for autonomous driving on a Dutch highway?*

RQ 3.2: *How to assess the safety requirements in the design of a perception system?*

Chapter 4 presented a case study assessing the safety of the Apollo automated driving framework's perception system in design. We elicited 58 safety requirements for a Dutch highway segment with a focus on its weather and illumination conditions. For the assessment of safety requirements, we used 23 design choices; 13 relating to traditional software and the other ten specific to ML based systems. We found design evidence that 38 out of 58 requirements are met. While all requirements relating to traditional software systems are satisfied, many requirements specific to ML based systems are not found satisfied.

RQ 4: *What characterizes automotive software projects in open source?*

Chapter 5 presented a landscape of automotive software projects publicly available on GitHub. We identified and categorized ≈600 automotive repositories grounded in definitions from literature and well-defined empirical methods. We also identified a similar number of non-automotive projects for comparison. We analyzed the origin, temporal trends, key players, popularity of projects, languages for development, user distribution across repositories, and development activities. Based on this analysis, we also present a first-of-its-kind manually curated dataset of automotive projects and a comparison set of non-automotive projects for replication and future research.

For an industry traditionally closed source in its half-a-century history of software use, open sourcing software projects marks a landmark change. Chapter 5 showed that the automotive domain is undergoing a shift along multiple dimensions, including the prevalence of automated driving software development, change in preferred language from MATLAB to Python, and entry of software companies and startups into the domain.

We foresee that recent developments in software engineering, which enable automated driving, will further accelerate open-source automotive software development. We believe that the software stacks for automated driving will benefit from perception and decision software currently developed in open source. Since these systems are developed independent of car makers, involving the open-source community for the acceleration of their development is a logical step.

6.1 Future work

The last decade marks arguably the most significant paradigm shift in the automotive industry since its inception. Four dimensions of this shift are the transition from internal combustion to electrification, automated and connected driving, the ongoing shift to open source software development, and start-ups entering the field and finding success bringing disruptive ideas and new business models. This means profound changes in almost all dimensions of automotive software, its development, and the electronics that run them.

While we empirically showed (in Section 2.5) that safety requirement elicitation of the traditional components like the steering system has matured, the safety requirement elicitation of newer age concepts and components like ML based perception systems are not. Future research focusing on these newer concepts and components is of immediate importance, especially since such systems are on the verge of entering production.

Relatedly, whether the safety requirements elicitation is catching up to the developments of newer concepts and components is still an open question. The open-sourcing trend is still unfolding [12] and its safety (requirement elicitation) side might be too early to research. However, the safety of communication-side that enables connected driving and

the ML based components that will allow automated driving, might already be at a stage that needs immediate research.

There can be three kinds of communications that enable connected driving: vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-road user (V2R). The first two kinds are more established than the third kind. As we mentioned in Section 2.3, there is a lack of methods that integrates V2I along with the traditional automotive safety requirement elicitation. There are at least two challenges here, (1) safety requirement elicitation for communication intermediaries like cloud, for connected driving and (2) requirements spanning across multiple vendors (e.g., the manufacturers for smart traffic infrastructure and vehicles might be different). Further research is needed in these directions.

Cyber-security is another prominent direction to explore in connected driving alongside safety. The communications part that enables connected driving increases the potential attack surfaces and can compromise the system's safety. Integral approaches for requirement elicitation considering both safety and security are a potential future research direction. On the architecture assessment side, this thesis focused on software architecture. However, safety is often achieved via hardware architecture or a combination of hardware and software. The second part of our method focused only at the software level. The logical next step is to extend our software architecture assessment approach (from Section 3.2) to address safety requirements that are fulfilled specifically in hardware and the combination of hardware and software.

Also, the current research in architecture assessment does not account for different risk levels across different requirements. This means that with current assessment techniques, we can only conclude whether a safety requirement is addressed but not whether the specific level associated with the requirement is addressed. Augmenting the assessment method with risk levels is a potential future research direction. This will allow prioritizing safety requirements based on the risk associated with them. This may also be a step towards a trade-off analysis where each safety requirement can be traded off with other requirements based on the risk associated.

Our case studies, especially on the Apollo stack, are the first studies in the scientific literature to present the safety design assessment for the perception system of a mature software stack for automated driving in a real-life setting. This opens up a multitude of future research directions, including safety requirement elicitation on other automated driving stacks and their comparison to Apollo; and techniques to reduce the human effort (and the resulting subjectivity) of the requirement elicitation and assessment. Our case studies also provide practitioners with the parts which need more work and possible design choices to consider for closing safety gaps. For all of our case studies, we have shared our data, including results from its intermediate steps for transparency, replicability, and reusability of our work for research and practice.

Regarding the open sourcing trend in automotive, we noticed that organizations mostly open source their tools and some experimental projects. It will be interesting to see what

types of companies come to open source and their motivation (similar to a recent study exploring motivations of Chinese companies to open source [191]). Given the entry of organizations to open source in the automotive domain is still in its infancy, it is possible that some individual projects[1] might actually belong to organizations and the transfer of ownership to the organization has yet to be made. This phenomenon in itself and the subsequent changes in our results form another direction to explore for future research. Further, our data and insights from Chapter 5 can be used to identify (a) software for reuse, (b) attract talent and/or increase the adoption of software and standards, and (c) new directions, companies, and trends in the automotive domain.

From an education perspective, in the context of software engineering, most curricula do not focus on safety, let alone the safety of the newer types of systems.Since software systems are becoming more safety-critical, this trend needs to change, and safety has to be incorporated as a topic. We believe it is equally important to educate on the caveats and limitations of current techniques since it is crucial to understand the scope of existing processes and techniques for their correct usage in real-life. Also, in typical software engineering curricula, a systems view is seldom included but is crucial in the context of safety requirements. We strongly encourage including this in the curricula for educating future software engineers.

We firmly believe that every stakeholder involved, including car makers, safety certification bodies, automotive software developers, and vehicle users, has the right to an unbiased understanding of the safety of the vehicles' software, especially in automated driving settings. This thesis is a step in that direction.

[1]While our results show that one in three automotive software repositories is owned by an organization, it is possible that at least some of the GitHub projects in large organizations might have started as personal projects because there was not yet a company policy on how to open source projects.

A

Appendix: Safety tactics

Table A.1: Safety tactics used in Chapters 3 and 4; verbatim from the work of Preschern et al. [92].

Tactic	Aim	Description
Simplicity	Avoid failures through keeping the system as simple as possible.	*Simplicity* reduces the system complexity. It includes structuring methods or cutting unnecessary functionality and organizes system elements or reduces them to their core safety functionality, thus, eliminating hazards. An example for the application of the *Simplicity tactic* is an emergency stop switch system which is usually kept as simple as possible.
Substitution	Avoid failures though usage of more reliable components.	Components or methods are replaced by other components or methods one has higher confidence in. For hardware and software this can mean usage of existing components which are well-proven in the safety domain.

A

Tactic	Aim	Description
Sanity Check (Checking)	Detection of implausible system outputs or states.	The *Sanity Check tactic* checks whether a system state or value remains within a valid range which can be defined in the system specification or which is based on knowledge about the internal structure or nature of the system. An example for a *Sanity Check* is a stuck-at fault RAM-test which checks the proper functionality of the memory during system runtime. The test is based on the understanding of the memory behavior (if we write data to the memory, we should later on be able to read the same data). Faults are detected if the memory behaves differently.
Condition Monitoring (Checking)	Detect deviations from the intended system outputs or states.	*Condition Monitoring* checks whether a system value remains within a reasonable range compared to a more reliable, but usually less accurate, reference value. The reference value is computed at runtime by a redundant part in the implementation which can be based on system input values and is not pre-known from the specification (like it would be the case for *Sanity Check*). An example for *Condition Monitoring* is a system which has to be time-synchronized via the Internet and which checks if the synchronized time is feasible by comparing it to an internal clock.
Comparison	Detection of discrepancies of redundant system outputs.	*Comparison* tests if the outputs of fully redundant subsystems are equal in order to detect failures. The *Comparison tactic* usually implies the usage of a redundancy tactic. An example for the application of the *Comparison tactic* is a dual-core processor running in lockstep mode. The processor runs the same software on both cores and compares their outputs after each cycle.

Tactic	Aim	Description
Diverse Redundancy (Redundancy)	Introduction of a redundant system which allows detection or masking of failures in the specification or implementation as well as random hardware failures.	*Diverse Redundancy* can be applied to the specification or to the implementation level. In a system using *Diverse Redundancy* on the implementation level, redundant components use different implementations which were developed independently from the same specification. *Diverse Redundancy* on a specification level goes one step further and additionally requires that even the requirement specifications for the redundant components have to be set up by individual teams.
Replication Redundancy (Redundancy)	Introduction of a redundant systems which allows detection or masking of random hardware failures (not systematic failures).	*Replication Redundancy* means introduction of a redundant system of the same implementation. The redundant systems maintain the same functionality, use identical hardware, and run the same software implementation. An example for *Replication Redundancy* is the RAID1 data storage technology.
Repair (Recovery)	Bring a failed system back to a state of full functionality.	The full system functionality is manually or automatically restored if a system failure occurs.
Degradation (Recovery)	*Degradation* brings a system with an error into a state with reduced functionality in which the system still maintains the core safety functions.	*Degradation* systems define a core safety functionality. The systems maintain this safety functionality and additional non-critical functions. In case of an error, the system falls back into a degraded mode in which it just maintains the core safety functionality. An example where the *Degradation tactic* is often applied are automation systems. These systems control safety-critical processes and often visualize these processes in a GUI. If the system has too few resources (e.g. processing time), then the system stops the GUI service and just focuses on its core functionality to control the safety-critical processes.

A

Tactic	Aim	Description
Voting (Masking)	Mask the failure of a subsystem so that the failure does not propagate to other systems.	*Voting* makes a failure transparent. The tactic does not try to repair the failure, but it hides the failure through choosing a correct result from redundant subsystems. It decides for the majority of the output values.
Override (Masking)	Mask the failure of a subsystem so that the failure does not propagate to other systems.	The *Override tactic* forces the system output to a safe state. For example, if we have a system which is in a safe state when shut off, we can apply the *Override tactic* to shut off the system if we have doubt about the system output (e.g. if an output validity check fails). In this scenario overriding the system output with a safe output value decreases the availability of the system. Another form of the *Override tactic,* which does not decrease the availability and is closely related to the Voting tactic, chooses the output of redundant subsystems by preferring one subsystem or one output state over another.
Barrier	Protect a subsystem from influences or influencing other subsystems.	The *Barrier tactic* provides a mechanism to protect from unintentional influences between subsystems. To apply *Barrier,* the interfaces between subsystems have to be analyzed and specified. These interfaces are controlled at runtime by a trustworthy component (the *Barrier*) which often is an already existing reliable mechanism. An example for a *Barrier* is a memory protection unit which controls and restricts the communication between different tasks.

Bibliography

[1] Robert N. Charette. How software is eating the car. https://spectrum.ieee.org/software-eating-car, 2021. Accessed: 06 September 2022.

[2] Ralph W. Carp, Harold E. Weissler, and Paul R. Kudlaty. Hybrid electronic control unit for fuel management systems, 1980. US Patent 4,212,066.

[3] Joe Barkai. The Fallacy Behind Counting Lines of Code. http://joebarkai.com/fallacy-behind-counting-lines-of-code/, 2015. Accessed: 06 September 2022.

[4] Erwin Schoitsch. Autonomous vehicles and automated driving status, perspectives and societal impact. In *Proceedings of the Information Technology, Society and Economy Strategic Cross-Influences - 24th Interdisciplinary Information Management Talks*, pages 405–424, 2016.

[5] ISO. ISO 26262:2018 – Road vehicles – Functional safety. Standard, International Organization for Standardization, 2018. https://www.iso.org/standard/68383.html.

[6] ISO. ISO/PAS 21448:2019 – Road vehicles – Safety of the intended functionality. Standard, International Organization for Standardization, 2019. https://www.iso.org/standard/70939.html.

[7] Krzysztof Czarnecki. Automated driving system (ADS) high–level quality requirements analysis–driving behavior safety. Technical report, University of Waterloo, Canada, 2018. https://doi.org/10.13140/RG.2.2.23280.76800.

[8] Commission of the European Communities. Commission recommendation of 22 December 2006 on safe and efficient in-vehicle information and communication systems: update of the European statement of principles on human machine interface. Technical report, The Commission of the European Communities, 2006. https://op.europa.eu/en/publication-detail/-/publication/00e7ffec-49e3-492b-8e8e-8839cae806bc/.

[9] SAE. SAE J3061_202112 – Cybersecurity Guidebook for Cyber-Physical Vehicle Systems. Standard, Society of Automotive Engineers, 2021. https://www.sae.org/standards/content/j3061_202112/.

[10] SAE. ISO/SAE 21434:2021 Road vehicles – Cybersecurity engineering. Standard, Society of Automotive Engineers & International Organization for Standardization, 2021. https://www.iso.org/standard/70918.html.

[11] Apollo Auto. Apollo automated driving platform. https://github.com/ApolloAuto/apollo. Accessed: 06 September 2022.

[12] Sangeeth Kochanthara, Yanja Dajsuren, Loek Cleophas, and Mark van den Brand. Painting the landscape of automotive software in GitHub. In *Proceedings of the 2022 Mining Software Repositories Conference*, pages 215–226, 2022.

[13] Zi Peng, Jinqiu Yang, Tse-Hsun Chen, and Lei Ma. A first look at the integration of machine learning models in complex autonomous driving systems: A case study on apollo. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1240–1250, 2020.

[14] Sagar Behere and Martin Torngren. A functional architecture for autonomous driving. In *Proceedings of the First International Workshop on Automotive Software Architecture*, pages 3–10, 2015.

[15] Jinhan Kim, Jeongil Ju, Robert Feldt, and Shin Yoo. Reducing DNN labelling cost using surprise adequacy: An industrial case study for autonomous driving. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1466–1476, 2020.

[16] Automated Vehicle Safety Consortium (AVSC). AVSC best practice for describing an operational design domain: Conceptual framework and lexicon. Technical report, SAE Industry Technologies Consortia, 2020. https://www.sae.org/standards/content/avsc00002202004/.

[17] Eric Thorn, Shawn C. Kimmel, Michelle Chaka, Booz Allen Hamilton, et al. A framework for automated driving system testable cases and scenarios. Technical report, US DOT National Highway Traffic Safety, United States of America, 2018. https://www.nhtsa.gov/sites/nhtsa.gov/files/documents/13882-automateddrivingsystems_092618_v1a_tag.pdf.

[18] Sangeeth Kochanthara, Yanja Dajsuren, Loek Cleophas, and Mark van den Brand. Replication package: Painting the landscape of automotive software in GitHub. https://doi.org/10.5281/zenodo.5885013, 2022. Accessed: 06 September 2022.

[19] Luiz Eduardo G. Martins and Tony Gorschek. Requirements engineering for safety-critical systems: A systematic literature review. In *Information and Software Technology*, volume 75, pages 71–89. Elsevier, 2016.

[20] Tarcísio Pereira, Deivson Albuquerque, Aêda Sousa, Fernanda M. R. Alencar, and Jaelson Castro. Retrospective and Trends in Requirements Engineering for Embedded Systems: A Systematic Literature Review. In *Proceedings of the 20th Ibero-American Conference on Software Engineering*, pages 427–440, 2017.

[21] Jessyka Vilela, Jaelson Castro, Luiz Eduardo G. Martins, and Tony Gorschek. Integration between requirements engineering and safety analysis: A systematic literature review. In *Journal of Systems and Software*, volume 125, pages 68–92. Elsevier, 2017.

[22] Darko Bozhinoski, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Ivica Crnkovic. Safety for mobile robotic systems: A systematic mapping study from a software engineering perspective. In *Journal of Systems and Software*, volume 151, pages 150–179. Elsevier, 2019.

[23] Stephan Baumgart and Joakim Froberg. Functional Safety in Product Lines - A Systematic Mapping Study. In *Proceedings of the 42nd Euromicro Conference on Software Engineering and Advanced Applications*, pages 313–322, 2016.

[24] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University, United Kingdom, 2007. `https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf`.

[25] Paul Garner, Sally Hopewell, Jackie Chandler, Harriet MacLehose, Elie A. Akl, Joseph Beyene, Stephanie Chang, Rachel Churchill, Karin Dearness, Gordon Guyatt, et al. When and how to update systematic reviews: consensus and checklist. In *British Medical Journal*, volume 354, pages 1–10. British Medical Journal Publishing Group, 2016.

[26] Mark Petticrew and Helen Roberts. *Systematic reviews in the social sciences: A practical guide*. John Wiley & Sons, 2008.

[27] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. In *Information and Software Technology*, volume 64, pages 1–18. Elsevier, 2015.

[28] Saurabh Tiwari and Atul Gupta. A systematic literature review of use case specifications research. In *Information and Software Technology*, volume 67, pages 128–158. Elsevier, 2015.

[29] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. In *Requirements engineering*, volume 11, pages 102–107. Springer, 2006.

[30] Emilia Mendes, Claes Wohlin, Katia Felizardo, and Marcos Kalinowski. When to update systematic literature reviews in software engineering. In *Journal of Systems and Software*, volume 167, pages 1–24. Elsevier, 2020.

[31] Vahid Garousi and João M. Fernandes. Highly-cited papers in software engineering: The top-100. In *Information and Software Technology*, volume 71, pages 108–128. Elsevier, 2016.

[32] Lutz Bornmann. How are excellent (highly cited) papers defined in bibliometrics? A quantitative analysis of the literature. In *Research Evaluation*, volume 23, pages 166–173. Oxford University Press, 2014.

[33] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–10, 2014.

[34] Jacob Cohen. Weighted kappa: nominal scale agreement provision for scaled disagreement or partial credit. In *Psychological bulletin*, volume 70, pages 213–220. American Psychological Association, 1968.

[35] Jacob Cohen. A coefficient of agreement for nominal scales. In *Educational and psychological measurement*, volume 20, pages 37–46. Sage Publications, 1960.

[36] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. In *Biometrics*, volume 33, pages 159–174. JSTOR, 1977.

[37] Nancy G. Leveson and John P. Thomas. *STPA handbook*. , 2018. `https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf`.

[38] IEC. IEC 61508:2010 – Functional safety of electrical/electronic/programmable electronic safety-related systems. Standard, International Electrotechnical Commission, 2010. `https://webstore.iec.ch/publication/5515`.

[39] Stephan Baumgart, Joakim Fröberg, and Sasikumar Punnekkat. Analyzing hazards in system-of-systems: Described in a quarry site automation context. In *Proceedings of the 2017 Annual IEEE International Systems Conference*, pages 1–8, 2017.

[40] Manfred Broy, Ingolf H. Kruger, Alexander Pretschner, and Christian Salzmann. Engineering automotive software. In *Proceedings of the IEEE*, volume 95, pages 356–373. IEEE, 2007.

[41] Emil Gracic, Fredrik Svensson, Jesko Ehrich, Oliver Beck, and Maximilian Jansen. Concept for Safety-Related Development of Deep Neural Networks in the Automotive. In *Proceedings of the Fourth International Conference on Multimedia Computing, Networking and Applications*, pages 10–15, 2020.

[42] Wen-Shing Lee, Doris L. Grosh, Frank A. Tillman, and Chang H. Lie. Fault tree analysis, methods, and applications a review. In *IEEE transactions on reliability*, volume 34, pages 194–203. IEEE, 1985.

[43] Diomidis H. Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. Quality Press, 2003.

[44] Trevor A. Kletz. HAZOP–past and future. In *Reliability Engineering & System Safety*, volume 55, pages 263–266. Elsevier, 1997.

[45] Terje Aven and Bjørnar Heide. Reliability and validity of risk analysis. In *Reliability Engineering & System Safety*, volume 94, pages 1862–1868. Elsevier, 2009.

[46] B.D. Johnston. A structured procedure for dependent failure analysis (DFA). In *Reliability Engineering*, volume 19, pages 125–136. Elsevier, 1987.

[47] W. M. Goble and A.C. Brombacher. Using a failure modes, effects and diagnostic analysis (FMEDA) to measure diagnostic coverage in programmable electronic systems. In *Reliability engineering & system safety*, volume 66, pages 145–148. Elsevier, 1999.

[48] Anjali Joshi, Mats P.E. Heimdahl, Steven P. Miller, and Mike W. Whalen. Model-based safety analysis. Technical report, University of Minnesota, United States of America, 2006. `https://shemesh.larc.nasa.gov/fm/Model-BasedSafetyAnalysis.pdf`.

[49] Marc Wilikens, Marcelo Masera, and Davide Vallero. Integration of safety requirements in the initial phases of the project lifecycle of hardware/software systems. In *Safe Comp 97*, pages 83–97. Springer, 1997.

[50] Amer Saeed, Rogério de Lemos, and Tom Anderson. On the safety analysis of requirements specifications for safety-critical software. In *ISA Transactions*, volume 34, pages 283–295. Elsevier, 1995.

[51] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. A survey of accelerator architectures for deep neural networks. In *Engineering*, volume 6, pages 264–274. Elsevier, 2020.

[52] Michael J. Maloni, Craig R. Carter, and Amelia S. Carr. Assessing logistics maturation through author concentration. In *International Journal of Physical Distribution & Logistics Management*, volume 39, pages 250–268. Emerald Group Publishing Limited, 2009.

[53] Myun J. Cheon, Varun Groven, and Rajiv Sabherwal. The evolution of empirical research in IS: A study in IS maturity. In *Information & Management*, volume 24, pages 107–119. Elsevier, 1993.

[54] Andy Neely. The evolution of performance measurement research: developments in the last decade and a research agenda for the next. In *International journal of operations & production management*, volume 25, pages 1264–1277. Emerald Group Publishing Limited, 2005.

[55] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[56] Paulo Gabriel Gadelha Queiroz and Rosana Teresinha Vaccare Braga. Development of critical embedded systems using model-driven and product lines techniques: A systematic review. In *Proceedings of the Eighth Brazilian Symposium on Software Components, Architectures and Reuse*, pages 74–83, 2014.

[57] Sunil Nair, Jose Luis De La Vara, Mehrdad Sabetzadeh, and Lionel Briand. An extended systematic literature review on provision of evidence for safety certification. In *Information and Software Technology*, volume 56, pages 689–717. Elsevier, 2014.

[58] Rajwinder Kaur Panesar-Walawege, Mehrdad Sabetzadeh, Lionel Briand, and Thierry Coq. Characterizing the chain of evidence for software safety cases: A conceptual model based on the IEC 61508 standard. In *Proceedings of the Third International Conference on Software Testing, Verification and Validation*, pages 335–344, 2010.

[59] Jose Luis de la Vara and Rajwinder Kaur Panesar-Walawege. Safetymet: A metamodel for safety standards. In *Proceedings of the ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems*, pages 69–86, 2013.

[60] Jose Luis De la Vara, Alejandra Ruiz, Katrina Attwood, Huáscar Espinoza, Rajwinder Kaur Panesar-Walawege, Ángel López, Idoya del Río, and Tim Kelly. Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel. In *Information and software technology*, volume 72, pages 16–30. Elsevier, 2016.

[61] Victor Bolbot, Gerasimos Theotokatos, Luminita Manuela Bujorianu, Evangelos Boulougouris, and Dracos Vassalos. Vulnerabilities and safety assurance methods in Cyber-Physical Systems: A comprehensive review. In *Reliability Engineering & System Safety*, volume 182, pages 179–193. Elsevier, 2019.

[62] Johann Thor Mogensen Ingibergsson, Ulrik Pagh Schultz, and Marco Kuhrmann. On the use of safety certification practices in autonomous field robot software development: A systematic mapping study. In *Proceedings of the 16th International Conference on Product-Focused Software Process Improvement*, pages 335–352, 2015.

[63] S. Manoj Kannan, Yanja Dajsuren, Yaping Luo, and Ion Barosan. Analysis of ISO 26262 Compliant Techniques for the Automotive Domain. In *Proceedings of the International Workshop on Modelling in Automotive Software Engineering co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*, pages 1–10, 2015.

[64] Mukul Anil Gosavi, Benjamin B. Rhoades, and James M. Conrad. Application of functional safety in autonomous vehicles using ISO 26262 standard: A survey. In *Proceedings of the 2018 SoutheastCon*, pages 1–6, 2018.

[65] Youcef Gheraibia, Sohag Kabir, Khaoula Djafri, and Habiba Krimou. An Overview of the Approaches for Automotive Safety Integrity Levels Allocation. In *Journal of Failure Analysis and Prevention*, volume 18, pages 707–720. Springer Science and Business Media LLC, 2018.

[66] Markus Borg, Cristofer Englund, Krzysztof Wnuk, Boris Duran, Christoffer Levandowski, Shenjian Gao, Yanwen Tan, Henrik Kaijser, Henrik Lönn, and Jonas Törnqvist. Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry. In *Journal of Automotive Software Engineering*, volume 1, pages 1–19. Springer Nature, 2019.

[67] Jakob Axelsson. Safety in vehicle platooning: A systematic literature review. In *IEEE Transactions on Intelligent Transportation Systems*, volume 18, pages 1033–1045. IEEE, 2016.

[68] Guoqi Xie, Yanwen Li, Yunbo Han, Yong Xie, Gang Zeng, and Renfa Li. Recent Advances and Future Trends for Automotive Functional Safety Design Methodologies. In *IEEE Transactions on Industrial Informatics*, volume 16, pages 5629–5642. IEEE, 2020.

[69] Alessandra Nardi, Samir Camdzic, Antonino Armato, and Francesco Lertora. Design-For-Safety for Automotive IC Design: Challenges and Opportunities. In *Proceedings of the Custom Integrated Circuits Conference*, pages 1–8, 2019.

[70] Sangeeth Kochanthara, Niels Rood, Loek Cleophas, Yanja Dajsuren, and Mark van den Brand. Semi-automatic architectural suggestions for the functional safety of cooperative driving systems. In *Companion proceedings of the 2020 IEEE International Conference on Software Architecture*, pages 55–58, 2020.

[71] Sangeeth Kochanthara, Niels Rood, Arash Khabbaz Saberi, Loek Cleophas, Yanja Dajsuren, and Mark van den Brand. A functional safety assessment method for cooperative automotive architecture. In *Journal of Systems and Software*, volume 179, pages 1–14. Elsevier, 2021.

[72] Sangeeth Kochanthara, Niels Rood, Arash Khabbaz Saberi, Loek Cleophas, Yanja Dajsuren, and Mark van den Brand. Summary: A functional safety assessment method for cooperative automotive architecture. In *Companion proceedings of the 15th European Conference on Software Architecture*, pages 1–5, 2021.

[73] Pablo Alvarez, Iosu Lerga, Adrian Serrano, and Javier Faulin. Considering congestion costs and driver behaviour into route optimisation algorithms in smart cities. In *Proceedings of the 14th International Conference on Smart Cities*, pages 39–50, 2017.

[74] Todd Trego and Dan Murray. An analysis of the operational costs of trucking. In *Proceedings of the Transportation Research Board 2010 Annual Meetings*, pages 20–22, 2010.

[75] Jeroen Ploeg. *Analysis and design of controllers for cooperative and automated driving*. PhD thesis, Eindhoven University of Technology, The Netherlands, 2014. `https://research.tue.nl/en/publications/analysis-and-design-of-controllers-for-cooperative-and-automated-`.

[76] Kuo-Yun Liang, Jonas Mårtensson, and Karl H Johansson. Heavy-duty vehicle platoon formation for fuel efficiency. In *IEEE Transactions on Intelligent Transportation Systems*, volume 17, pages 1051–1061. IEEE, 2015.

[77] Patrizio Pelliccione, Eric Knauss, S Magnus Ågren, Rogardt Heldal, Carl Bergenhem, Alexey Vinel, and Oliver Brunnegård. Beyond connected cars: A systems of systems perspective. In *Science of Computer Programming*, volume 191, pages 1–21. Elsevier, 2020.

[78] Yanja Dajsuren and Guido Loupias. Safety analysis method for cooperative driving systems. In *Proceedings of the 2019 IEEE International Conference on Software Architecture*, pages 181–190, 2019.

[79] Piergiuseppe Mallozzi, Patrizio Pelliccione, Alessia Knauss, Christian Berger, and Nassar Mohammadiha. Autonomous vehicles: State of the art, future trends, and challenges. In *Automotive Systems and Software Engineering*, pages 347–367. Springer, 2019.

[80] Josef Nilsson, Carl Bergenhem, Jan Jacobson, Rolf Johansson, and Jonny Vinter. Functional safety for cooperative systems. Technical report, Society of Automotive Engineers, 2013. `https://www.sae.org/publications/technical-papers/content/2013-01-0197/`.

[81] Arash Khabbaz Saberi, Eric Barbier, Frank Benders, and Mark van den Brand. On functional safety methods: A system of systems approach. In *Proceedings of the 2018 Annual IEEE International Systems Conference*, pages 1–6, 2018.

[82] Muhammad Ali Babar, Liming Zhu, and Ross Jeffery. A framework for classifying and comparing software architecture evaluation methods. In *Proceedings of the 2004 Australian Software Engineering Conference*, pages 309–318, 2004.

[83] Liliana Dobrica and Eila Niemela. A survey on software architecture analysis methods. In *IEEE Transactions on software Engineering*, volume 28, pages 638–653. IEEE, 2002.

[84] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carriere. The architecture tradeoff analysis method. In *Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems*, pages 68–78, 1998.

[85] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2012.

[86] PerOlof Bengtsson and Jan Bosch. Scenario-based software architecture reengineering. In *Proceedings of the Fifth International Conference on Software Reuse*, pages 308–317, 1998.

[87] Christopher Stoermer, Felix Bachmann, and Chris Verhoef. SACAM: the software architecture comparison analysis method. Technical report, Carnegie Mellon University, United States of America, 2003. `https://resources.sei.cmu.edu/asset_files/TechnicalReport/2003_005_001_14219.pdf`.

[88] Klaus Bergner, Andreas Rausch, Marc Sihling, and Thomas Ternité. DoSAM–Domain-specific Software Architecture comparison Model. In *Quality of Software Architectures and Software Quality*, pages 4–20. Springer, 2005.

[89] Neil Harrison and Paris Avgeriou. Pattern-based architecture reviews. In *IEEE Software*, volume 28, pages 66–71. IEEE, 2010.

[90] Jan Bosch and Peter Molin. Software architecture design: evaluation and transformation. In *Proceedings of the 1999 IEEE Conference and Workshop on Engineering of Computer-Based Systems*, pages 4–10, 1999.

[91] Weihang Wu and Tim Kelly. Safety tactics for software architecture design. In *Proceedings of the 28th Annual International Computer Software and Applications Conference*, pages 368–375, 2004.

[92] Christopher Preschern, Nermin Kajtazovic, and Christian Kreiner. Building a safety architecture pattern system. In *Proceedings of the 18th European Conference on Pattern Languages of Program*, pages 1–55, 2015.

[93] Manfred Broy, Mario Gleirscher, Peter Kluge, Wolfgang Krenzer, Stefano Merenda, and Doris Wild. Automotive architecture framework: Towards a holistic and standardised system architecture description. Technical report, Technical University of Munich, Germany, 2009. https://mediatum.ub.tum.de/doc/1094456/file.pdf.

[94] Alessio Bucaioni and Patrizio Pelliccione. Technical architectures for automotive systems. In *Proceedings of the 2020 IEEE International Conference on Software Architecture*, pages 46–57, 2020.

[95] Yanjindulam Dajsuren. *On the design of an architecture framework and quality evaluation for automotive software systems.* PhD thesis, Eindhoven University of Technology, The Netherlands, 2015. https://pure.tue.nl/ws/files/15934981/20160307_Dajsuren.pdf.

[96] Miroslaw Staron. *Automotive software architectures.* Springer, 2021.

[97] Yanja Dajsuren and Mark van den Brand, editors. *Automotive Systems and Software Engineering: State of the Art and Future Trends.* Springer International Publishing, 2019.

[98] Qi Van Eikema Hommes. Review and assessment of the ISO 26262 draft road vehicle-functional safety. Technical report, Society of Automotive Engineers, 2012. https://www.sae.org/publications/technical-papers/content/2012-01-0025/.

[99] Christopher Preschern, Nermin Kajtazovic, Christian Kreiner, et al. Catalog of safety tactics in the light of the IEC 61508 safety lifecycle. In *Proceedings of the VikingPLoP 2013 Conference*, pages 79–95, 2013.

[100] Andreas Riel, Christian Kreiner, Richard Messnarz, and Alexander Much. An architectural approach to the integration of safety and security requirements in smart products and systems design. In *CIRP Annals*, volume 67, pages 173–176. Elsevier, 2018.

[101] ISO. ISO 26262: 2011 – Road vehicles — Functional safety. Standard, International Organization for Standardization, 2011. https://www.iso.org/standard/43464.html.

[102] Yuting Fu. Fault injection mechanisms for validating dependability of automotive systems. Master's thesis, Eindhoven University of Technology, The Netherlands, 2018. https://research.tue.nl/en/studentTheses/fault-injection-mechanisms-for-validating-dependability-of-automo.

[103] Sangeeth Kochanthara, Niels Rood, Arash Khabbaz Saberi, Loek Cleophas, Yanja Dajsuren, and Mark van den Brand. A case study on ISO 26262 extension for connected driving. https://github.com/SangeethNila/casestudy_ISO26262_extension_connected_driving, 2021. Accessed: 06 September 2022.

[104] Frans Hoogeboom. *Safety of automated vehicles: design, implementation, and analysis*. PhD thesis, Eindhoven University of Technology, The Netherlands, 2020. https://research.tue.nl/nl/publications/safety-of-automated-vehicles-design-implementation-and-analysis.

[105] Alex Serban, Erik Poll, and Joost Visser. A standard driven software architecture for fully autonomous vehicles. In *Companion proceedings of the 2018 IEEE International Conference on Software Architecture*, pages 120–127, 2018.

[106] Shahriar Hasan. *Fail-Operational and Fail-Safe Vehicle Platooning in the Presence of Transient Communication Errors*. PhD thesis, Mälardalen University, Sweden, 2020. https://www.diva-portal.org/smash/get/diva2:1646357/FULLTEXT01.pdf.

[107] Oliver Sawade, Matthias Schulze, and Ilja Radusch. Robust communication for cooperative driving maneuvers. In *IEEE Intelligent Transportation Systems Magazine*, volume 10, pages 159–169. IEEE, 2018.

[108] PerOlof Bengtsson, Nico Lassing, Jan Bosch, and Hans van Vliet. Architecture-level modifiability analysis (alma). In *Journal of Systems and Software*, volume 69, pages 129–147. Elsevier, 2004.

[109] Banani Roy and T.C. Nicholas Graham. Methods for evaluating software architecture: A survey. Technical report, Queen's University at Kingston, Canada, 2008. https://research.cs.queensu.ca/TechReports/Reports/2008-545.pdf.

[110] Neil B. Harrison and Paris Avgeriou. Using pattern-based architecture reviews to detect quality attribute issues–An exploratory study. In *Transactions on Pattern Languages of Programming III*, pages 168–194. Springer, 2013.

[111] Mario Barbacci, Paul C. Clements, Anthony Lattanze, Linda Northrop, and William Wood. Using the Architecture Tradeoff Analysis Method (ATAM) to evaluate the software architecture for a product line of avionics systems: A case study. Technical report, Carnegie Mellon University, United States of America, 2003. https://resources.sei.cmu.edu/asset_files/TechnicalNote/2003_004_001_14150.pdf.

[112] Kristian Beckers, Isabelle Côté, Thomas Frese, Denis Hatebur, and Maritta Heisel. Systematic derivation of functional safety requirements for automotive systems. In *Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security*, pages 65–80, 2014.

[113] Joakim Oscarsson, Max Stolz-Sundnes, Naveen Mohan, and Viacheslav Izosimov. Applying systems-theoretic process analysis in the context of cooperative driving. In *Proceedings of the 11th IEEE Symposium on Industrial Embedded Systems*, pages 1–5, 2016.

[114] Max Stoltz-Sundnes. STPA–Inspired safety analysis of driver-vehicle interaction in cooperative driving automation. Master's thesis, KTH Royal Institute of Technology, Sweden, 2019. http://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1371216&dswid=-9346.

[115] P. Cuenot, C. Ainhauser, N. Adler, S. Otten, and F. Meurville. Applying model based techniques for early safety evaluation of an automotive architecture in compliance with the ISO 26262 standard. In *Proceedings of the 2014 Embedded Real Time Software and Systems Congress*, pages 1–11, 2014.

[116] Helmut Martin, Z. Ma, Ch Schmittner, Bernhard Winkler, Martin Krammer, Daniel Schneider, Tiago Amorim, Georg Macher, and Ch Kreiner. Combined automotive safety and security pattern engineering approach. In *Reliability Engineering & System Safety*, volume 198, page 106773. Elsevier, 2020.

[117] Irfan Sljivo, Garazi Juez Uriagereka, Stefano Puri, and Barbara Gallina. Guiding assurance of architectural design patterns for critical applications. In *Journal of Systems Architecture*, volume 110, pages 1–11. Elsevier, 2020.

[118] Matthias Althoff and John M Dolan. Online verification of automated road vehicles using reachability analysis. In *IEEE Transactions on Robotics*, volume 30, pages 903–918. IEEE, 2014.

[119] Zeeshan E. Bhatti, Partha S. Roop, and Roopak Sinha. Unified functional safety assessment of industrial automation systems. In *IEEE Transactions on Industrial Informatics*, volume 13, pages 17–26. IEEE, 2016.

[120] Piergiuseppe Mallozzi, Massimo Sciancalepore, and Patrizio Pelliccione. Formal verification of the on-the-fly vehicle platooning protocol. In *Proceedings of the 8th International Workshop on Software Engineering for Resilient Systems*, pages 62–75, 2016.

[121] Andrzej Zalewski, Klara Borowa, and Andrzej Ratkowski. On cognitive biases in architecture decision making. In *Proceedings of the 11th European Conference on Software Architecture*, pages 123–137, 2017.

[122] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. In *IEEE access*, volume 8, pages 58443–58469. IEEE, 2020.

[123] Vard Antinyan. Revealing the complexity of automotive software. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1525–1528, 2020.

[124] Miltos Kyriakidis, Riender Happee, and Joost C.F. de Winter. Public opinion on automated driving: Results of an international questionnaire among 5000 respondents. In *Transportation research part F: traffic psychology and behaviour*, volume 32, pages 127–140. Elsevier, 2015.

[125] Stephanie Abrecht, Lydia Gauerhof, Christoph Gladisch, Konrad Groh, Christian Heinzemann, and Matthias Woehrle. Testing deep learning-based visual perception for automated driving. In *ACM Transactions on Cyber-Physical Systems*, volume 5, pages 1–28. ACM, 2021.

[126] Vincenzo Riccio and Paolo Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 876–888, 2020.

[127] Alessio Gambi, Tri Huynh, and Gordon Fraser. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 257–267, 2019.

[128] Alex Serban, Erik Poll, and Joost Visser. Adversarial examples on object recognition: A comprehensive survey. In *ACM Computing Surveys*, volume 53, pages 1–38. ACM, 2020.

[129] Dutch Ministry of Infrastructure and the Environment. Road traffic signs and regulations in the Netherlands. `https://www.universiteitleiden.nl/binaries/content/assets/customsites/study-abroad-exchange-students/road_traffic_signs_and_regulations_jan_2013_uk.pdf`, 2013. Accessed: 06 September 2022.

[130] Sangeeth Kochanthara, Tajinder Singh, Alexandru Forrai, and Loek Cleophas. Replication package: Safety of Perception Systems for Automated Driving: A Case Study on Apollo. `https://doi.org/10.5281/zenodo.6532255`, 2022. Accessed: 06 September 2022.

[131] Guowei Wan, Xiaolong Yang, Renlan Cai, Hao Li, Yao Zhou, Hao Wang, and Shiyu Song. Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes. In *Proceedings of the 2018 IEEE international conference on robotics and automation*, pages 4670–4677, 2018.

[132] Matthew Schwall, Tom Daniel, Trent Victor, Francesca Favaro, and Henning Hohnhold. Waymo public road safety performance data. *arXiv preprint arXiv:2011.00038*, 2020.

[133] Institute for Road Safety Research (SWOV). Road design – which road categories are distinguished in the Netherlands? https://www.swov.nl/en/facts-figures/fact/road-design-which-road-categories-are-distinguished-netherlands. Accessed: 06 September 2022.

[134] Gerrit Bagschik, Andreas Reschka, Torben Stolte, and Markus Maurer. Identification of potential hazardous events for an unmanned protective vehicle. In *Proceedings of the 2016 IEEE Intelligent Vehicles Symposium*, pages 691–697, 2016.

[135] Christopher Becker, John C. Brewer, Larry Yount, et al. Safety of the intended functionality of lane-centering and lane-changing maneuvers of a generic level 3 highway chauffeur system. Technical report, US DOT National Highway Traffic Safety, United States of America, 2020. https://rosap.ntl.bts.gov/view/dot/53628.

[136] Mohamed Aladem, Stanley Baek, and Samir A. Rawashdeh. Evaluation of image enhancement techniques for vision-based navigation under low illumination. In *Journal of Robotics*, volume 2019, pages 1–16. Hindawi, 2019.

[137] Shih-Chia Huang, Trung-Hieu Le, and Da-Wei Jaw. Dsnet: Joint semantic learning for object detection in inclement weather conditions. In *IEEE transactions on pattern analysis and machine intelligence*, volume 43, pages 2623–2633. IEEE, 2020.

[138] Vishwanath A. Sindagi, Poojan Oza, Rajeev Yasarla, and Vishal M Patel. Prior-based domain adaptive object detection for hazy and rainy conditions. In *Proceedings of the 16TH European Conference on Computer Vision*, pages 763–780, 2020.

[139] Georg Volk, Stefan Müller, Alexander von Bernuth, Dennis Hospach, and Oliver Bringmann. Towards robust CNN-based object detection through augmentation with synthetic rain variations. In *Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference*, pages 285–292, 2019.

[140] Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S. Ecker, Matthias Bethge, and Wieland Brendel. Benchmarking robustness in object detection: Autonomous driving when winter is coming. *arXiv preprint arXiv:1907.07484*, 2019.

[141] Keisuke Yoneda, Naoki Suganuma, Ryo Yanase, and Mohammad Aldibaja. Automated driving recognition technologies for adverse weather conditions. In *IATSS research*, volume 43, pages 253–262. Elsevier, 2019.

[142] Alexander Carballo, Jacob Lambert, Abraham Monrroy, David Wong, Patiphon Narksri, Yuki Kitsukawa, Eijiro Takeuchi, Shinpei Kato, and Kazuya Takeda. Libre: The multiple 3D LiDAR dataset. In *Proceedings of the 2020 IEEE Intelligent Vehicles Symposium*, pages 1094–1101, 2020.

[143] Mario Bijelic, Tobias Gruber, and Werner Ritter. A benchmark for LiDAR sensors in fog: Is detection breaking down? In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 760–767. IEEE, 2018.

[144] Li Tang, Yunpeng Shi, Qing He, Adel W. Sadek, and Chunming Qiao. Performance test of autonomous vehicle LiDAR sensors under different weather conditions. In *Transportation research record*, volume 2674, pages 319–329. SAGE Publications, 2020.

[145] Ji-Il Park, Jihyuk Park, and Kyung-Soo Kim. Fast and accurate desnowing algorithm for LiDAR point clouds. In *IEEE Access*, volume 8, pages 160202–160212. IEEE, 2020.

[146] Jonas Westman and Mattias Nyberg. A reference example on the specification of safety requirements using ISO 26262. In *Proceedings of the ERCIM/EWICS Workshop on Dependable Embedded and Cyber-physical Systems of the 32nd International Conference on Computer Safety, Reliability and Security*, pages 1–8, 2013.

[147] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. Adoption and effects of software engineering best practices in machine learning. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–12, 2020.

[148] Sina Mohseni, Mandar Pitale, Vasu Singh, and Zhangyang Wang. Practical solutions for machine learning safety in autonomous vehicles. *arXiv preprint arXiv:1912.09630*, 2019.

[149] Hironori Washizaki, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. Studying software engineering patterns for designing machine learning systems. In *Proceedings of the 10th International Workshop on Empirical Software Engineering in Practice*, pages 49–495, 2019.

[150] Alex Serban and Joost Visser. Adapting software architectures to machine learning challenges. In *Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering*, pages 152–163, 2022.

[151] Denis Baylor, Kevin Haas, Konstantinos Katsiapis, Sammy Leong, Rose Liu, Clemens Menwald, Hui Miao, Neoklis Polyzotis, Mitchell Trott, and Martin Zinkevich. Continuous training for production ML in the TensorFlow extended (TFX) platform. In *Proceedings of the 2019 USENIX Conference on Operational Machine Learning*, pages 51–53, 2019.

[152] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. The ML test score: A rubric for ML production readiness and technical debt reduction. In *Proceedings of the 2017 IEEE International Conference on Big Data*, pages 1123–1132, 2017.

[153] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1723–1726, 2017.

[154] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, page 2503–2511, 2015.

[155] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[156] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

[157] Sanjit A. Seshia, Ankush Desai, Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte, and Xiangyu Yue. Formal specification for deep neural networks. In *Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis*, pages 20–34, 2018.

[158] Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 4885–4894, 2017.

[159] Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *Proceedings of the Thirty-sixth International Conference on Machine Learning*, pages 2151–2159, 2019.

[160] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the Thirty-fourth International Conference on Machine Learning*, pages 1321–1330, 2017.

[161] Sina Mohseni, Mandar Pitale, J.B.S. Yadawa, and Zhangyang Wang. Self-supervised learning for generalizable out-of-distribution detection. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 5216–5223, 2020.

[162] Izhak Golan and Ran El-Yaniv. Deep anomaly detection using geometric transformations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 9781–9791, 2018.

[163] Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L. Willke. Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In *Proceedings of the European Conference on Computer Vision*, pages 550–564, 2018.

[164] Xiaofeng Zhang, Zhangyang Wang, Dong Liu, Qifeng Lin, and Qing Ling. Deep adversarial data augmentation for extremely low data regimes. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 31, pages 15–28. IEEE, 2020.

[165] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1180–1189, 2015.

[166] Hyungtae Lee, Sungmin Eum, and Heesung Kwon. Me R-CNN: Multi-expert R-CNN for object detection. In *IEEE Transactions on Image Processing*, volume 29, pages 1030–1044. IEEE, 2019.

[167] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *Proceedings of the Sixth International Conference on Learning Representations*, pages 1–22, 2018.

[168] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *Proceedings of the Sixth International Conference on Learning Representations*, pages 1–14, 2018.

[169] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *Proceedings of the Thirty-sixth International Conference on Machine Learning*, pages 2712–2721, 2019.

[170] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6405–6416, 2017.

[171] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1050–1059, 2016.

[172] Linda Northrop, Ipek Ozkaya, George Fairbanks, and Michael Keeling. Designing the software systems of the future. In *ACM SIGSOFT Software Engineering Notes*, volume 43, pages 28–30. ACM, 2019.

[173] Angela Horneman, Andrew Mellinger, and Ipek Ozkaya. AI Engineering: 11 foundational practices. Technical report, Carnegie Mellon University, United States of America, 2020. https://resources.sei.cmu.edu/asset_files/WhitePaper/2019_019_001_634648.pdf.

[174] Adina Aniculaesei, Jörg Grieser, Andreas Rausch, Karina Rehfeldt, and Tim Warnecke. Toward a holistic software systems engineering approach for dependable autonomous systems. In *Proceedings of the IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems*, pages 23–30, 2018.

[175] Alex Serban, Erik Poll, and Joost Visser. Towards using probabilistic models to design software systems with inherent uncertainty. In *Proceedings of the 14th European Conference on Software Architecture*, pages 89–97, 2020.

[176] Alexandru Constantin Serban. Designing safety critical software systems to manage inherent uncertainty. In *Companion proceedings of the 2019 IEEE International Conference on Software Architecture*, pages 246–249, 2019.

[177] Alessandro Biondi, Federico Nesti, Giorgiomaria Cicero, Daniel Casini, and Giorgio Buttazzo. A safe, secure, and predictable software architecture for deep learning in safety-critical systems. In *IEEE Embedded Systems Letters*, volume 12, pages 78–82. IEEE, 2019.

[178] Eoin Woods. Software architecture in a changing world. In *IEEE Software*, volume 33, pages 94–97. IEEE, 2016.

[179] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2017.

[180] OpenMMLab. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. https://github.com/open-mmlab/mmdetection3d. Accessed: 06 September 2022.

[181] Waymo. Waymo safety report: On the road to fully self-driving. https://assets.ctfassets.net/sv23gofxcuiz/1xAGjnH0kTxD2Vmvqsv3eS/7da216660cbd9ee7b35eefcac1b28a2f/waymo-safety-report-2018.pdf, 2018. Accessed: 06 September 2022.

[182] General Motors. Self-driving safety report. https://www.gm.com/content/dam/company/docs/us/en/gmcom/gmsafetyreport.pdf, 2018. Accessed: 06 September 2022.

[183] Audi, Aptiv, Baidu, BMW, Continental, Daimler, FCA Group, Here Technologies, Infineon, Intel, and Volkswagon. Safety first for automated driving. https://group.mercedes-benz.com/documents/innovation/other/safety-first-for-automated-driving.pdf, 2019. Accessed: 06 September 2022.

[184] Paolo Panaroni, Giovanni Sartori, Fabrizio Fabbrini, Mario Fusani, and Giuseppe Lami. Safety in automotive software: An overview of current practices. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1053–1058, 2008.

[185] Donal Heffernan, Ciaran MacNamee, and Padraig Fogarty. Runtime verification monitoring for automotive embedded systems using the ISO 26262 functional safety standard as a guide for the definition of the monitored properties. In *IET software*, volume 8, pages 193–203. IET, 2014.

[186] Sergio García, Daniel Strüber, Davide Brugali, Thorsten Berger, and Patrizio Pelliccione. Robotics software engineering: A perspective from the service robotics domain. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 593–604, 2020.

[187] Capers Jones and Olivier Bonsignour. *The economics of software quality*. Addison-Wesley Professional, 2011.

[188] Yuxiao Zhang, Alexander Carballo, Hanting Yang, and Kazuya Takeda. Autonomous driving in adverse weather conditions: A survey. *arXiv:2112.08936*, 2021.

[189] Christof Ebert and John Favaro. Automotive software. In *IEEE Software*, volume 34, pages 33–39. IEEE, 2017.

[190] Danielle Gonzalez, Thomas Zimmermann, and Nachiappan Nagappan. The state of the ML-universe: 10 years of artificial intelligence & machine learning software development on GitHub. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 431–442, 2020.

[191] Junxiao Han, Shuiguang Deng, David Lo, Chen Zhi, Jianwei Yin, and Xin Xia. An empirical study of the landscape of open source projects in Baidu, Alibaba, and Tencent. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice*, pages 298–307, 2021.

[192] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In *Proceedings of the 36th International Conference on Software Engineering*, pages 1–11, 2014.

[193] Mairieli Wessel, Bruno Mendes De Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. The power of bots: Characterizing and understanding bots in OSS projects. In *Proceedings of the ACM on Human-Computer Interaction*, volume 2, pages 1–19. ACM, 2018.

[194] Georgios Gousios. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 233–236, 2013.

[195] Alireza Haghighatkhah, Ahmad Banijamali, Olli-Pekka Pakanen, Markku Oivo, and Pasi Kuvaja. Automotive software engineering: A systematic mapping study. In *Journal of Systems and Software*, volume 128, pages 25–55. Elsevier, 2017.

[196] Dip Goswami, Martin Lukasiewycz, Reinhard Schneider, and Samarjit Chakraborty. Time-triggered implementations of mixed-criticality automotive software. In *Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition*, pages 1227–1232, 2012.

[197] Abram Hindle, Neil A. Ernst, Michael W. Godfrey, and John Mylopoulos. Automated topic naming to support cross-project analysis of software maintenance activities. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 163–172, 2011.

[198] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimilano Di Penta, Denys Poshynanyk, and Andrea De Lucia. How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. In *Proceedings of the 35th International Conference on Software Engineering*, pages 522–531, 2013.

[199] Tarald O. Kvålseth. Note on Cohen's kappa. In *Psychological reports*, volume 65, pages 223–226. SAGE Publications, 1989.

[200] NASA. NASA Software Safety Standard (NASA-STD-8719.13B). Standard, National Aeronautics and Space Administration, 2004. `https://standards.nasa.gov/standard/nasa/nasa-std-871913`.

[201] Geir Kjetil Hanssen, Tor Stålhane, and Thor Myklebust. *SafeScrum®-Agile Development of Safety-Critical Software*. Springer, 2018.

[202] John A. McDermid. *Software engineer's reference book*. Elsevier, 2013.

[203] John C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th international conference on software engineering*, pages 547–550, 2002.

[204] Sangeeth Kochanthara. REVERT: RuntimE VEerification for Real-Time systems. Master's thesis, Indraprastha Institute of Information Technology Delhi, India, 2016. https://repository.iiitd.edu.in/xmlui/bitstream/handle/123456789/524/MT14055%20-%20Sangeeth%20K.pdf.

[205] Sangeeth Kochanthara, Geoffrey Nelissen, David Pereira, and Rahul Purandare. REVERT: a monitor generation tool for real-time systems. In *Proceedings of the 2016 IEEE Real-Time Systems Symposium*, pages 365–365, 2016.

[206] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Communications of the ACM*, volume 60, pages 84–90. ACM, 2017.

[207] Reinhard Bergmann, John Ludbrook, and Will P.J.M. Spooren. Different outcomes of the Wilcoxon—Mann—Whitney test from different statistics packages. In *The American Statistician*, volume 54, pages 72–77. Taylor & Francis, 2000.

[208] Harald Altinger, Franz Wotawa, and Markus Schurius. Testing methods used in the automotive industry: Results from a survey. In *Proceedings of the 2014 Workshop on Joining Academia and Industry Contributions to Test Automation and Model-Based Testing*, pages 1–6, 2014.

[209] Daniel Kästner, Christoph Cullmann, Gernot Gebhard, Sebastian Hahn, Thomas Karos, Laurent Mauborgne, Stephan Wilhelm, and Christian Ferdinand. Safety-critical software development in C++. In *Proceedings of the 39th International Conference on Computer Safety, Reliability, and Security*, pages 98–110, 2020.

[210] Gregorio Robles and Jesus M. Gonzalez-Barahona. Developer identification methods for integrated data from various sources. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[211] Georgios Gousios and Diomidis Spinellis. Mining software engineering data from GitHub. In *Companion proceedings of the 39th IEEE/ACM International Conference on Software Engineering*, pages 501–502, 2017.

[212] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. An in-depth study of the promises and perils of mining GitHub. In *Empirical Software Engineering*, volume 21, pages 2035–2071. Springer, 2016.

[213] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating GitHub for engineered software projects. In *Empirical Software Engineering*, volume 22, pages 3219–3253. Springer, 2017.

[214] Jorge Aranda and Gina Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *Proceedings of the IEEE 31st International Conference on Software Engineering*, pages 298–308, 2009.

[215] Stefan Kugele, David Hettler, and Jan Peter. Data-centric communication and containerization for future automotive software architectures. In *Proceedings of the 2018 IEEE International Conference on Software Architecture*, pages 65–74, 2018.

[216] Fabio Falcini, Giuseppe Lami, and Alessandra Mitidieri Costanza. Deep learning in automotive software. In *IEEE Software*, volume 34, pages 56–63. IEEE, 2017.

[217] Rick Salay and Krzysztof Czarnecki. Using machine learning safely in automotive software: An assessment and adaption of software process requirements in ISO 26262. *arXiv preprint arXiv:1808.01614*, 2018.

[218] Johannes Schlatow, Mischa Möstl, Rolf Ernst, Marcus Nolte, Inga Jatzkowski, and Markus Maurer. Towards model-based integration of component-based automotive software systems. In *Proceedings of the 43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 8425–8432, 2017.

[219] Philipp Obergfell, Stefan Kugele, and Eric Sax. Model-based resource analysis and synthesis of service-oriented automotive software architectures. In *Proceedings of the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems*, pages 128–138, 2019.

[220] Markus Zoppelt and Ramin Tavakoli Kolagari. SAM: a Security Abstraction Model for automotive software systems. In *Proceedings of the 2018 International Workshop on Cyber Security for Intelligent Transportation Systems and 2018 International Workshop on Interplay of Security, Safety and System/Software Architecture*, pages 59–74. Springer, 2018.

[221] Ali Dorri, Marco Steger, Salil S. Kanhere, and Raja Jurdak. Blockchain: A distributed solution to automotive security and privacy. In *IEEE Communications Magazine*, volume 55, pages 119–125. IEEE, 2017.

[222] Stefan Kugele, Philipp Obergfell, Manfred Broy, Oliver Creighton, Matthias Traub, and Wolfgang Hopfensitz. On service-orientation for automotive software. In *Proceedings of the 2017 IEEE International Conference on Software Architecture*, pages 193–202, 2017.

[223] Dennis Kengo Oka, Toshiyuki Fujikura, and Ryo Kurachi. Shift left: Fuzzing earlier in the automotive software development lifecycle using hil systems. In *Proceedings of the 16th Embedded Security in Cars Europe conference*, pages 1–13, 2018.

[224] Dhasarathy Parthasarathy, Karl Bäckstrom, Jens Henriksson, and Sólrún Einarsdóttir. Controlled time series generation for automotive software-in-the-loop testing using gans. In *Proceedings of the 2020 IEEE International Conference On Artificial Intelligence Testing*, pages 39–46, 2020.

[225] Brian Katumba and Eric Knauss. Agile development in automotive software development: Challenges and opportunities. In *Proceedings of the 15th International Conference on International Conference on Product-Focused Software Process Improvement*, pages 33–47, 2014.

Primary studies: Chapter 2

[P1] Daniel Aceituna, Kaushik Madala, and Hyunsook Do. Deriving functional safety requirements using undesired combination state templates. In *Proceedings - 2018 4th International Workshop on Requirements Engineering for Self-Adaptive, Collaborative, and Cyber Physical Systems*, pages 1–8. IEEE, 2018.

[P2] Toshiaki Aoki, Kriangkrai Traichaiyaporn, Yuki Chiba, Masahiro Matsubara, Masataka Nishi, and Fumio Narisawa. Modeling safety requirements of ISO26262 using goal trees and patterns. In *Communications in Computer and Information Science*, volume 596, pages 206–221. Springer, 2016.

[P3] Luís Silva Azevedo, David Parker, Yiannis Papadopoulos, Martin Walker, Ioannis Sorokos, and Rui Esteves Araújo. Exploring the impact of different cost heuristics in the allocation of safety integrity levels. In *Lecture Notes in Computer Science*, volume 8822, pages 70–81. Springer, 2014.

[P4] Luis Silva Azevedo, David Parker, Martin Walker, Yiannis Papadopoulos, and Rui Esteves Araujo. Assisted assignment of automotive safety requirements. *IEEE Software*, 31:62–68, 2014.

[P5] Gerrit Bagschik, Andreas Reschka, Torben Stolte, and Markus Maurer. Identification of potential hazardous events for an Unmanned Protective Vehicle. In *IEEE Intelligent Vehicles Symposium, Proceedings*, volume 2016, pages 691–697. IEEE, 2016.

[P6] Gerrit Bagschik, Torben Stolte, and Markus Maurer. Safety Analysis Based on Systems Theory Applied to an Unmanned Protective Vehicle. *Procedia Engineering*, 179:61–71, 2017.

[P7] Kristian Beckers, Isabelle Côté, Thomas Frese, Denis Hatebur, and Maritta Heisel. Systematic derivation of functional safety requirements for automotive systems. In *Lecture Notes in Computer Science*, volume 8666 LNCS, pages 65–80. Springer, 2014.

[P8] Kristian Beckers, Isabelle Côté, Thomas Frese, Denis Hatebur, and Maritta Heisel. A structured and systematic model-based development method for automotive systems, considering the OEM/supplier interface. *Reliability Engineering and System Safety*, 158:172–184, 2017.

[P9] Carl Bergenhem, Mario Majdandzic, and Stig Ursing. Concepts and risk analysis for a cooperative and automated highway platooning system. In *Communications in Computer and Information Science*, volume 1279, pages 200–213. Springer, 2020.

[P10] Lucas Bressan, Andre L. de Oliveira, and Fernanda Campos. An Approach to Support Variant Management on Safety Analysis using CHESS Error Models. In *2020 16th European Dependable Computing Conference*, pages 135–142. IEEE, 2020.

[P11] Chih Chung Chiu and Kuo Sui Lin. A new design review method for functional safety of automotive electrical systems. In *ACM International Conference Proceeding Series*, ICNCC 2018, pages 318–326. ACM, 2018.

[P12] Shivakumar Chonnad, Radu Iacob, and Vladimir Litovtchenko. A Quantitative Approach to SoC Functional Safety Analysis. In *International System on Chip Conference*, volume 2018, pages 227–232. IEEE, 2019.

[P13] Yanja Dajsuren and Guido Loupias. Safety analysis method for cooperative driving systems. In *Proceedings - 2019 IEEE International Conference on Software Architecture, ICSA 2019*, pages 181–190. IEEE, 2019.

[P14] Nabarun Das and William Taylor. Quantified fault tree techniques for calculating hardware fault metrics according to ISO 26262. In *ISPCE 2016 - Proceedings: IEEE Symposium on Product Compliance Engineering*, pages 1–8. IEEE, 2016.

[P15] Veenesh Dhaked, Vrinda Gupta, and Jyoti Harmalkar. Application of Concept Phase to Design an Electric Powertrain in Compliance with ISO 26262. In *2019 IEEE 5th International Conference for Convergence in Technology, I2CT 2019*, pages 1–5. IEEE, 2019.

[P16] Dominik Domis, Rasmus Adler, and Martin Becker. Integrating variability and safety analysis models using commercial UML-based tools. In *ACM International Conference Proceeding Series*, volume 20-24 of *SPLC '15*, pages 225–234. ACM, 2015.

[P17] Adam Duracz, Ayman Aljarbouh, Ferenc A. Bartha, Jawad Masood, Roland Philippsen, Henrik Eriksson, Jan Duracz, Fei Xu, Yingfu Zeng, and Christian Grante. Advanced Hazard Analysis and Risk Assessment in the ISO 26262 Functional Safety Standard Using Rigorous Simulation. In *Lecture Notes in Computer Science*, volume 11971 LNCS, pages 108–126. Springer, 2020.

[P18] Alessandro Frigerio, Bart Vermeulen, and Kees Goossens. A Generic Method for a Bottom-Up ASIL Decomposition. In *Lecture Notes in Computer Science*, volume 11093 LNCS, pages 12–26. Springer, 2018.

[P19] Alessandro Frigerio, Bart Vermeulen, and Kees Goossens. Component-level ASIL decomposition for automotive architectures. In *Proceedings - 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, DSN-W 2019*, pages 62–69. IEEE, 2019.

[P20] Mohamad Gharib, Paolo Lollini, Marco Botta, Elvio Amparore, Susanna Donatelli, and Andrea Bondavalli. On the Safety of Automotive Systems Incorporating Machine Learning Based Components: A Position Paper. In *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018*, pages 271–274. IEEE, 2018.

[P21] Mohamad Gharib, Paolo Lollini, Andrea Ceccarelli, and Andrea Bondavalli. Dealing with functional safety requirements for automotive systems: A cyber-physical-social approach. In *Lecture Notes in Computer Science*, volume 10707 LNCS, pages 194–206. Springer, 2018.

[P22] Mohamad Gharib, Paolo Lollini, Andrea Ceccarelli, and Andrea Bondavalli. Engineering functional safety requirements for automotive systems: A cyber-physical-social approach. In *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, volume 2019, pages 74–81. IEEE, 2019.

[P23] Youcef Gheraibia, Khaoula Djafri, and Habiba Krimou. Reduction of Solution Space in the Automotive Safety Integrity Levels Allocation Problem. In *Modelling and Implementation of Complex Systems*, pages 67–76. Springer, 2016.

[P24] Youcef Gheraibia, Khaoula Djafri, and Habiba Krimou. Ant colony algorithm for automotive safety integrity level allocation. *Applied Intelligence*, 48:555–569, 2018.

[P25] Gerhard Griessnig and Adam Schnellbach. Development of the 2nd edition of the ISO 26262. In *Communications in Computer and Information Science*, volume 748, pages 535–546. Springer, 2017.

[P26] Manfred Großmann, Mario Hirz, and Jürgen Fabian. Efficient application of multi-core processors as substitute of the E-Gas (Etc) monitoring concept. In *Proceedings of 2016 SAI Computing Conference, SAI 2016*, pages 913–918. IEEE, 2016.

[P27] Chao Huang and Liang Li. Architectural design and analysis of a steer-by-wire system in view of functional safety concept. *Reliability Engineering and System Safety*, 198:106822, 2020.

[P28] Benjamin Peter Jeppesen, Meenakshi Rajamani, and Kevin Mark Smith. Enhancing functional safety in FPGA-based motor drives. *The Journal of Engineering*, 2019:4580–4584, 2019.

[P29] Michael Kaessmeyer, David Santiago Velasco Moncada, and Markus Schurius. Evaluation of a systematic approach in variant management for safety-critical systems development. In *Proceedings - IEEE/IFIP 13th International Conference on Embedded and Ubiquitous Computing, EUC 2015*, pages 35–43. IEEE, 2015.

[P30] Siddartha Khastgir, Stewart Birrell, Gunwant Dhadyalla, Håkan Sivencrona, and Paul Jennings. Towards increased reliability by objectification of Hazard Analysis and Risk Assessment (HARA) of automated automotive systems. *Safety Science*, 99:166–177, 2017.

[P31] Sebastiaan Klaasse, Geert Kwintenberg, and Ion Barosan. Development of a Functional Safety Software Layer for the Control of an Electric In-Wheel Motor Based Powertrain. In *Proceedings - 2018 IEEE 15th International Conference on Software Architecture Companion, ICSA-C 2018*, pages 144–147. IEEE, 2018.

[P32] Sangeeth Kochanthara, Niels Rood, Loek Cleophas, Yanja Dajsuren, and Mark Van Den Brand. Semi-automatic Architectural Suggestions for the Functional Safety of Cooperative Driving Systems. In *Proceedings - 2020 IEEE International Conference on Software Architecture Companion, ICSA-C 2020*, pages 55–58. IEEE, 2020.

[P33] Greta Carlotta Kolln, Michael Klicker, and Stephan Schmidt. Comparison of hazard analysis methods with regard to the series development of autonomous vehicles. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pages 2969–2975, 2019.

[P34] Birte Kramer, Christian Neurohr, Matthias Büker, Eckard Böde, Martin Fränzle, and Werner Damm. Identification and Quantification of Hazardous Scenarios for Automated Driving. In *Lecture Notes in Computer Science*, volume 12297 LNCS, pages 163–178. Springer, 2020.

[P35] Kuen Long Leu, Hsiang Huang, Yung Yuan Chen, Li Ren Huang, and Kung Ming Ji. An intelligent brake-by-wire system design and analysis in accordance with ISO-26262 functional safety standard. In *2015 International Conference on Connected Vehicles and Expo, ICCVE 2015 - Proceedings*, pages 150–156. IEEE, 2016.

[P36] Hong Peng Li and Yan Wen Li. The research of electric vehicle's MCU system based on ISO26262. In *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems, ACIRS 2017*, pages 336–340. IEEE, 2017.

[P37] Christian Lidstrom, Carl Bondesson, Mattias Nyberg, and Jonas Westman. Improved Pattern for ISO 26262 ASIL Decomposition with Dependent Requirements. In *Proceedings - Companion of the 19th IEEE International Conference on Software Quality, Reliability and Security, QRS-C 2019*, pages 28–35. IEEE, 2019.

[P38] Boyu Liu and Yanwen Li. Research on Vehicle Control Unit based on functional safety. In *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems, ACIRS 2017*, pages 160–164. IEEE, 2017.

[P39] R. Mader, H. Martin, R. Obendrauf, P. Prinz, B. Winkler, and G. Grießnig. A framework for model-based safety requirements round-trip engineering. In *IET Conference Publications*, volume 2015, pages 1–6. IET, 2015.

[P40] Archana Mallya, Vera Pantelic, Morayo Adedjouma, Mark Lawford, and Alan Wassyng. Using STPA in an ISO 26262 compliant process. In *Lecture Notes in Computer Science*, volume 9922 LNCS, pages 117–129. Springer, 2016.

[P41] David Marcos, Jon Perez, Pello Zubizarreta, Maitane Garmendia, Igor Perez De Arenaza, Jon Crego, and Jose Antonio Cortajarena. A Safety Concept for an Automotive Lithium-based Battery Management System. In *2019 Electric Vehicles International Conference, EV 2019*, pages 1–6. IEEE, 2019.

[P42] Helmut Martin, Bernhard Winkler, Stephanie Grubmuller, and Daniel Watzenig. Identification of performance limitations of sensing technologies for automated driving. In *2019 8th IEEE International Conference on Connected Vehicles and Expo, ICCVE 2019 - Proceedings*, pages 1–6. IEEE, 2019.

[P43] Luiz Eduardo G. Martins and Tony Gorschek. Requirements engineering for safety-critical systems: An interview study with industry practitioners. *IEEE Transactions on Software Engineering*, 46(4):346–361, 2020.

[P44] Pierre Mauborgne, Samuel Deniaud, Éric Levrat, Éric Bonjour, Jean Pierre Micaëlli, and Dominique Loise. The Determination of Functional Safety Concept coupled with the definition of Logical Architecture: a framework of analysis from the automotive industry. *IFAC-PapersOnLine*, 50(1):7278–7283, 2017.

[P45] Richard Messnarz and Harald Sporer. Functional Safety Case with FTA and FMEDA Consistency Approach. In *Communications in Computer and Information Science*, volume 896, pages 387–397. Springer, 2018.

[P46] Helen E. Monkhouse, Ibrahim Habli, and John McDermid. An enhanced vehicle control model for assessing highly automated driving safety. *Reliability Engineering and System Safety*, 202:107061, 2020.

[P47] Helen Monkhouse, Ibrahim Habli, John McDermid, Siddartha Khastgir, and Gunwant Dhadyalla. Why functional safety experts worry about automotive systems having increasing autonomy. In *2017 IEEE SmartWorld Ubiquitous Intelligence and Computing, Advanced and Trusted Computed, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovation, SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI 2017*, pages 1–6. IEEE, 2018.

[P48] Peter Munk, Andreas Abele, Eike Thaden, Arne Nordmann, Rakshith Amarnath, Markus Schweizer, and Simon Burton. INVITED: Semi-Automatic Safety Analysis and Optimization. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[P49] Pramit Nag, Umesh Ghanekar, and Jyoti Harmalkar. A Novel Multi-Core Approach for Functional Safety Compliance of Automotive Electronic Control Unit According to ISO 26262. In *2019 IEEE 5th International Conference for Convergence in Technology, I2CT 2019*, pages 1–5. IEEE, 2019.

[P50] Alessandra Nardi and Antonino Armato. Functional safety methodologies for automotive applications. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, volume 2017, pages 970–975. IEEE, 2017.

[P51] Minseok Park, Hyun Chul Koag, and Hyun Sik Ahn. Functional Safety Improvement of Electric Power Steering System by Using Electronic Stability Control System. In *IEEE International Symposium on Industrial Electronics*, volume 2018, pages 230–234. IEEE, 2018.

[P52] Qing Rao and Jelena Frtunikj. Deep learning for self-driving cars: Chances and challenges: Extended Abstract. In *Proceedings - International Conference on Software Engineering*, SEFAIS '18, pages 35–38. ACM, 2018.

[P53] Lena Rogovchenko-Buffoni, Andrea Tundis, Muhammed Zoheb Hossain, Mattias Nyberg, and Peter Fritzson. An integrated toolchain for model based functional safety analysis. *Journal of Computational Science*, 5(3):408–414, 2014.

[P54] Alejandra Ruiz, Alberto Melzi, and Tim Kelly. Systematic application of ISO 26262 on a SEooC: Support by applying a systematic reuse approach. In *Proceedings of the 2015 Design, Automation and Test in Europe, DATE*, volume 2015, pages 393–396. IEEE, 2015.

[P55] Ravindra Reddy Sabbella and Maheswaran Arunachalam. Functional Safety Development of Motor Control Unit for Electric Vehicles. In *2019 IEEE Transportation Electrification Conference, ITEC-India 2019*, pages 1–6. IEEE, 2019.

[P56] Aleksandra Salikiryaki, Iliana Petrova, and Stephan Baumgart. Graphical Approach for Modeling of Safety and Variability in Product Lines. In *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, pages 410–417. IEEE, 2015.

[P57] Georg Scharfenberg, Ludek Elis, and Gerhard Hofmann. New Design Methodology - Using VHDL-AMS Models to Consider Aging Effects in Automotive Mechatronic Circuits for Safety Relevant Functions. In *International Conference on Applied Electronics*, volume 2019, pages 1–5. IEEE, 2019.

[P58] Tobias Schmid, Stefanie Schraufstetter, and Stefan Wagner. An Approach for Structuring a Highly Automated Driving Multiple Channel Vehicle System for Safety Analysis. In *Proceedings - 2018 3rd International Conference on System Reliability and Safety, ICSRS 2018*, pages 362–367. IEEE, 2019.

[P59] Valerij Schönemann, Hermann Winner, Thomas Glock, Stefan Otten, Eric Sax, Bert Boeddeker, Geert Verhaeg, Fabrizio Tronci, and Gustavo G. Padilla. Scenario-based functional safety for automated driving on the example of valet parking. In *Advances in Intelligent Systems and Computing*, volume 886, pages 53–64. Springer, 2019.

[P60] Moga Natha Shankar Kumar and Karthikeyan Balakrishnan. Functional Safety Development of Battery Management System for Electric Vehicles. In *2019 IEEE Transportation Electrification Conference, ITEC-India 2019*, pages 1–6. IEEE, 2019.

[P61] Jacopo Sini, M. D'Auria, and Massimo Violante. Towards Vehicle-Level Simulator Aided Failure Mode, Effect, and Diagnostic Analysis of Automotive Power Electronics Items. In *21st IEEE Latin-American Test Symposium, LATS 2020*, pages 1–6. IEEE, 2020.

[P62] Jacopo Sini and Massimo Violante. A simulation-based methodology for aiding advanced driver assistance systems hazard analysis and risk assessment. *Microelectronics Reliability*, 109:113661, 2020.

[P63] Jacopo Sini, Massimo Violante, and Riccardo Dessi. ISO26262-Compliant Development of a High Dependable Automotive Powertrain Item. In *Lecture Notes in Electrical Engineering*, volume 604, pages 315–326. Springer, 2020.

[P64] Martin Skoglund, Hans Svensson, Henrik Eriksson, Thomas Arts, Rolf Johansson, and Alex Gerdes. Checking verification compliance of technical safety requirements on the AUTOSAR platform using annotated semi-formal executable models. In *Lecture Notes in Computer Science*, volume 8696 LNCS, pages 19–26. Springer, 2014.

[P65] Torben Stolte, Gerrit Bagschik, and Markus Maurer. Safety goals and functional safety requirements for actuation systems of automated vehicles. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 2191–2198. IEEE, 2016.

[P66] Torben Stolte, Gerrit Bagschik, Andreas Reschka, and Markus Maurer. Hazard analysis and risk assessment for an automated unmanned protective vehicle. In *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 1848–1855. IEEE, 2017.

[P67] Robert Tan. A safety concept for camera based ADAS based on multicore MCU. In *2014 IEEE International Conference on Vehicular Electronics and Safety, ICVES 2014*, pages 1–6. IEEE, 2015.

[P68] Chen Tao. Functional safety concept design of hybrid electric vehicle following ISO 26262. In *IEEE Transportation Electrification Conference and Expo, ITEC Asia-Pacific 2014 - Conference Proceedings*, pages 1–6. IEEE, 2014.

[P69] Sagar Sahebrao Tikar. Compliance of ISO 26262 safety standard for lithium ion battery and its battery management system in hybrid electric vehicle. In *2017 IEEE Transportation Electrification Conference, ITEC-India 2017*, volume 2018, pages 1–5. IEEE, 2018.

[P70] Mohamed Tlig, Mathilde Machin, Romain Kerneis, Emmanuel Arbaretier, Linda Zhao, Florent Meurville, and Jean Van Frank. Autonomous Driving System : Model Based Safety Analysis. In *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018*, pages 2–5. IEEE, 2018.

[P71] Raphael Fonte Boa Trindade, Lukas Bulwahn, and Christoph Ainhauser. Automatically generated safety mechanisms from semi-formal software safety requirements. In *Lecture Notes in Computer Science*, volume 8666 LNCS, pages 278–293. Springer, 2014.

[P72] Ellen van Nunen, Francesco Esposto, Arash Khabbaz Saberi, and Jan Pieter Paardekooper. Evaluation of safety indicators for truck platooning. In *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 1013–1018. IEEE, 2017.

[P73] Yang Wang, Daniel Graziotin, Stefan Kriso, and Stefan Wagner. Communication channels in safety analysis: An industrial exploratory case study. *Journal of Systems and Software*, 153:135–151, 2019.

[P74] Yang Wang, Yanwen Li, Chunshu Li, and Xiyang Wang. Analysis and application of functional safety based on modified FMEA method. In *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems, ACIRS 2017*, pages 98–103. IEEE, 2017.

[P75] Yang Wang and Stefan Wagner. On groupthink in safety analysis: An industrial case study. In *Proceedings - International Conference on Software Engineering*, pages 266–275. IEEE, 2018.

[P76] Yung Chen Wang, Chi Seng Lee, Po Chan Kuo, and Yi Ling Lin. Overcurrent protection design, failure mode and effect analysis of an electric vehicle inverter. In *Proceedings of the IEEE International Conference on Industrial Technology*, volume 2016, pages 1287–1292. IEEE, 2016.

[P77] Fredrik Warg, Martin Gassilewski, Jörgen Tryggvesson, Viacheslav Izosimov, Anders Werneman, and Rolf Johansson. Defining autonomous functions using iterative hazard analysis and requirements refinement. In *Lecture Notes in Computer Science*, volume 9923 LNCS, pages 286–297. Springer, 2016.

[P78] Fredrik Warg, Martin Skoglund, Anders Thorsen, Rolf Johansson, Mattias Brannstrom, Magnus Gyllenhammar, and Martin Sanfridson. The Quantitative Risk Norm - A Proposed Tailoring of HARA for ADS. In *Proceedings - 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-W 2020*, pages 86–93. IEEE, 2020.

[P79] Ralph Weissnegger, Markus Schuß, Christian Kreiner, Markus Pistauer, Kay Römer, and Christian Steger. Seamless integrated simulation in design and verification flow for safety-critical systems. In *Lecture Notes in Computer Science*, volume 9923 LNCS, pages 359–370. Springer, 2016.

[P80] Jonas Westman and Mattias Nyberg. Providing tool support for specifying safety-critical systems by enforcing syntactic contract conditions. *Requirements Engineering*, 24:231–256, 2019.

[P81] Franz Wotawa, Bernhard Peischl, Florian Klück, and Mihai Nica. Quality assurance methodologies for automated driving. *Elektrotechnik und Informationstechnik*, 135:322–327, 2018.

[P82] Zhihong Wu, Xiezu Su, Yuan Zhu, and Luke. Functional safety system design on EPS. In *Lecture Notes in Electrical Engineering*, volume 418, pages 647–664. Springer Singapore, 2017.

[P83] Alison Young and Alastair Walker. Qualifying Dependent Failure Analysis Within ISO26262: Applicability to Semiconductors. In *Communications in Computer and Information Science*, volume 896, pages 331–340. Springer, 2018.

[P84] Jiyu Zhang, Giorgio Rizzoni, Andrea Cordoba-Arenas, Alessandro Amodio, and Bilin Aksun-Guvenc. Model-based diagnosis and fault tolerant control for ensuring

torque functional safety of pedal-by-wire systems. *Control Engineering Practice*, 61:255–269, 2017.

[P85] Yaling Zhou, Jing Guan, and Hanwen Sun. The Functional Safety Analysis and Design of Dual-Motor Hybrid Bus Clutch System. In *Proceedings of 2018 IEEE International Conference of Safety Produce Informatization, IICSPI 2018*, pages 299–304. IEEE, 2019.

[P86] Arash Khabbaz Saberi, Eric Barbier, Frank Benders, and Mark van den Brand. On functional safety methods: A system of systems approach. In *12th Annual IEEE International Systems Conference, SysCon 2018 - Proceedings*, pages 1–6. IEEE, 2018.

[P87] Asim Abdulkhaleq and Stefan Wagner. A software safety verification method based on system-theoretic process analysis. In *International Conference on Computer Safety, Reliability, and Security*, volume 8696 LNCS, pages 401–412. Springer, 2014.

[P88] Asim Abdulkhaleq, Stefan Wagner, and Nancy Leveson. A comprehensive safety engineering approach for software-intensive systems based on STPA. *Procedia Engineering*, 128:2–11, 2015.

[P89] Asim Abdulkhaleq and Stefan Wagner. A controlled experiment for the empirical evaluation of safety analysis techniques for safety-critical software. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, volume 27-29-2015, pages 1–10, 2015.

[P90] Asim Abdulkhaleq, Daniel Lammering, Stefan Wagner, Jürgen Röder, Norbert Balbierer, Ludwig Ramsauer, Thomas Raste, and Hagen Boehmert. A systematic approach based on STPA for developing a dependable architecture for fully automated driving vehicles. *Procedia Engineering*, 179:41–51, 2017.

[P91] Asim Abdulkhaleq, Markus Baumeister, Hagen Böhmert, and Stefan Wagner. Missing no interaction—Using STPA for identifying hazardous interactions of automated driving systems. *International Journal of Safety Science*, 2:115–124, 2018.

[P92] Majdi Ghadhab, Sebastian Junges, Joost-Pieter Katoen, Matthias Kuntz, and Matthias Volk. Model-based safety analysis for vehicle guidance systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 3–19. Springer, 2017.

[P93] Youcef Gheraibia, Abdelouahab Moussaoui, Luis S Azevedo, David Parker, Yiannis Papadopoulos, and Martin Walker. Can aquatic flightless birds allocate automotive safety requirements? In *2015 IEEE Seventh international conference on intelligent computing and information systems (ICICIS)*, pages 1–6. IEEE, 2015.

[P94] Joakim Oscarsson, Max Stolz-Sundnes, Naveen Mohan, Viacheslav Izosimov, and Ninla Elmawati Falabiba. Applying systems-theoretic process analysis in the context of cooperative driving. In *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–5. IEEE, 2016.

[P95] Jacopo Sini and Massimo Violante. An automatic approach to perform FMEDA safety assessment on hardware designs. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 49–52. IEEE, 2018.

[P96] Morayo Adedjouma, Gabriel Pedroza, and Boutheina Bannour. Representative safety assessment of autonomous vehicle for public transportation. *Proceedings - 2018 IEEE 21st International Symposium on Real-Time Computing, ISORC 2018*, pages 124–129, 2018.

[P97] Stephan Baumgart, Joakim Froberg, and Sasikumar Punnekkat. A Process to Support Safety Analysis for a System-of-Systems. *Proceedings - 2020 IEEE 31st International Symposium on Software Reliability Engineering Workshops, ISSREW 2020*, pages 61–66, 2020.

[P98] Junyi Chen, Shan Wang, Tangrui Zhou, Lu Xiong, and Xingyu Xing. Study on Safety Analysis Method for Take-over System of Autonomous Vehicles. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 1972–1977, 2020.

[P99] Alessio Di Sandro, Sahar Kokaly, Rick Salay, and Marsha Chechik. Querying Automotive System Models and Safety Artifacts: Tool Support and Case Study. *Journal of Automotive Software Engineering*, 1:34, 2020.

[P100] Niklas Grabbe, Anna Kellnberger, Beyza Aydin, and Klaus Bengler. Safety of automated driving: The need for a systems approach and application of the Functional Resonance Analysis Method. *Safety Science*, 126:104665, 2020.

[P101] Greta Koelln, Michael Klicker, and Stephan Schmidt. Comparison of the Results of the System Theoretic Process Analysis for a Vehicle SAE Level four and five. *2020 IEEE 23rd International Conference on Intelligent Transportation Systems, ITSC 2020*, 2020.

[P102] Ioannis Sorokos, Luis P. Azevedo, Yiannis Papadopoulos, Martin Walker, and David J. Parker. Comparing Automatic Allocation of Safety Integrity Levels in the Aerospace and Automotive Domains. *IFAC-PapersOnLine*, 49:184–190, 2016.

Curriculum Vitæ

Sangeeth Kochanthara was born on January 14th, 1991 in Vattamkulam, Kerala, India. After finishing secondary school in 2008 in Vattamkulam, he enrolled for a Bachelor's in Computer Science and Engineering at Calicut University in Kerala, India. After his Bachelor's, Sangeeth joined Indraprastha Institute of Information Technology Delhi (IIIT-Delhi) for his Master's in Computer Science and Engineering. During his Master's study, Sangeeth performed an internship at CISTER research lab, based at the School of Engineering of the Polytechnic Institute of Porto, Portugal. He also worked in the Program Analysis group at IIIT-Delhi. His Master's thesis, is on a domain-specific language, a monitor generation method, and a toolchain to generate monitors for real-time systems. Sangeeth obtained his Master's degree in 2016 with the gold medal and the best Master's thesis award. His Master's thesis was supervised by Dr. Geoffrey Nellisen, Dr. David Pereira, and Dr. Rahul Purandare. Sangeeth has received multiple scholarships and fellowships including the GATE scholarship by the All India Council for Technical Education, under the Ministry of Human Resource and Development, India; and the Junior Research Fellowship by the Council of Scientific & Industrial Research, India. In 2017, Sangeeth started his Ph.D. at Eindhoven University of Technology, The Netherlands, under the supervision of prof.dr. Mark van den Brand, dr. Yanja Dajsuren EngD, and dr.ir. Loek Cleophas. In his Ph.D. research, Sangeeth explored the software and system engineering aspects of automotive, predominantly from the lenses of requirements engineering, safety engineering, architecture, and mining software repositories. The results of the Ph.D. work are presented in this dissertation.

Titles in the IPA Dissertation Series since 2020

M.A. Cano Grijalba. *Session-Based Concurrency: Between Operational and Declarative Views.* Faculty of Science and Engineering, RUG. 2020-01

T.C. Nägele. *CoHLA: Rapid Co-simulation Construction.* Faculty of Science, Mathematics and Computer Science, RU. 2020-02

R.A. van Rozen. *Languages of Games and Play: Automating Game Design & Enabling Live Programming.* Faculty of Science, UvA. 2020-03

B. Changizi. *Constraint-Based Analysis of Business Process Models.* Faculty of Mathematics and Natural Sciences, UL. 2020-04

N. Naus. *Assisting End Users in Workflow Systems.* Faculty of Science, UU. 2020-05

J.J.H.M. Wulms. *Stability of Geometric Algorithms.* Faculty of Mathematics and Computer Science, TU/e. 2020-06

T.S. Neele. *Reductions for Parity Games and Model Checking.* Faculty of Mathematics and Computer Science, TU/e. 2020-07

P. van den Bos. *Coverage and Games in Model-Based Testing.* Faculty of Science, RU. 2020-08

M.F.M. Sondag. *Algorithms for Coherent Rectangular Visualizations.* Faculty of Mathematics and Computer Science, TU/e. 2020-09

D. Frumin. *Concurrent Separation Logics for Safety, Refinement, and Security.* Faculty of Science, Mathematics and Computer Science, RU. 2021-01

A. Bentkamp. *Superposition for Higher-Order Logic.* Faculty of Sciences, Department of Computer Science, VU. 2021-02

P. Derakhshanfar. *Carving Information Sources to Drive Search-based Crash Reproduction and Test Case Generation.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2021-03

K. Aslam. *Deriving Behavioral Specifications of Industrial Software Components.* Faculty of Mathematics and Computer Science, TU/e. 2021-04

W. Silva Torres. *Supporting Multi-Domain Model Management.* Faculty of Mathematics and Computer Science, TU/e. 2021-05

A. Fedotov. *Verification Techniques for xMAS.* Faculty of Mathematics and Computer Science, TU/e. 2022-01

M.O. Mahmoud. *GPU Enabled Automated Reasoning.* Faculty of Mathematics and Computer Science, TU/e. 2022-02

M. Safari. *Correct Optimized GPU Programs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2022-03

M. Verano Merino. *Engineering Language-Parametric End-User Programming Environments for DSLs.* Faculty of Mathematics and Computer Science, TU/e. 2022-04

G.F.C. Dupont. *Network Security Monitoring in Environments where Digital and Physical Safety are Critical.* Faculty of Mathematics and Computer Science, TU/e. 2022-05

T.M. Soethout. *Banking on Domain Knowledge for Faster Transactions*. Faculty of Mathematics and Computer Science, TU/e. 2022-06

P. Vukmirović. *Implementation of Higher-Order Superposition*. Faculty of Sciences, Department of Computer Science, VU. 2022-07

J. Wagemaker. *Concurrent Separation Logics for Safety, Refinement, and Security*. Faculty of Science, Mathematics and Computer Science, RU. 2022-08

R. Janssen. *Refinement and Partiality for Model-Based Testing*. Faculty of Science, Mathematics and Computer Science, RU. 2022-09

M. Laveaux. *Accelerated Verification of Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2022-10

S. Kochanthara. *A Changing Landscape: On Safety & Open Source in Automated and Connected Driving*. Faculty of Mathematics and Computer Science, TU/e. 2023-01