The Journal of Systems & Software 179 (2021) 110991

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A functional safety assessment method for cooperative automotive architecture☆,☆☆

Sangeeth Kochanthara *, Niels Rood, Arash Khabbaz Saberi, Loek Cleophas, Yanja Dajsuren, Mark van den Brand

Eindhoven University of Technology, The Netherlands

ARTICLE INFO

Article history:
Received 13 September 2020
Received in revised form 17 April 2021
Accepted 25 April 2021
Available online 4 May 2021

Keywords:
Functional safety
Cooperative driving
Platooning
ISO 26262
Automotive software architecture
Safety engineering

ABSTRACT

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 cooperative driving. The current automotive safety standard, ISO 26262, is designed for single vehicles. With the increasing number of cooperative 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 cooperative driving scenario are fulfilled in the technical architecture of a vehicle. We apply our method on a real-life academic prototype for a cooperative driving scenario, platooning, and discuss our insights.

1. Introduction

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 (Alvarez et al., 2017), 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 (Trego and Murray, 2010). One potential solution to reduce traffic congestion and such operational costs is *cooperative driving*.

Cooperative 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 (Ploeg, 2014). Cooperative driving can improve traffic efficiency, reduces cost, and increases comfort (Davila, 2013; Liang et al., 2015; Pelliccione et al., 2020). 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 cooperative driving capabilities, are projected to hit the roads.[2,3] With millions of cooperative driving capable vehicles on roads, the safety of these vehicles needs urgent attention.[4]

A majority of the cooperative 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 (Dajsuren and Loupias, 2019). 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, cooperative 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 (ISO, 2018). The ISO 26262 standard offers systematic methods from the safety engineering domain to identify safety requirements.

☆ This work is a part of the i-CAVE research programme (14897 P14-18) funded by NWO (Netherlands Organisation for Scientific Research), Netherlands.

☆☆ Editor: Neil Ernst.

* Corresponding author.
E-mail addresses: s.kochanthara@tue.nl (S. Kochanthara), n.rood@tue.nl (N. Rood), a.khabbaz.saberi@tue.nl (A.K. Saberi), l.g.w.a.cleophas@tue.nl (L. Cleophas), y.dajsuren@tue.nl (Y. Dajsuren), m.g.j.v.d.brand@tue.nl (M. van den Brand).

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://bit.ly/volkswagen-includes-nxp-v2x.

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

https://doi.org/10.1016/j.jss.2021.110991

Any automotive software architecture that fulfills these safety requirements is deemed safe-by-design.

ISO 26262 standard neither considers cooperative driving nor prescribes methods for architecture assessment. The standard is designed for single vehicles and does not include a cooperative perspective in which a set of vehicles is seen as a single entity (Mallozzi et al., 2019; Nilsson et al., 2013). This can mean that a low-risk safety requirement from a single-vehicle perspective can have catastrophic effects on other cooperating vehicles (Pelliccione et al., 2020). To create a functionally safe architecture from a cooperative perspective, existing studies have extended the standard guidelines (Kochanthara et al., 2020; Saberi et al., 2018) or presented an architecture framework (Pelliccione et al., 2020). Yet, checking the safety of software architecture of an existing vehicle for cooperative driving, remains an open question.

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 (Babar et al., 2004; Dobrica and Niemela, 2002; Kazman et al., 1998; Bass et al., 2012; Bengtsson and Bosch, 1998; Stoermer et al., 2003; Bergner et al., 2005; Harrison and Avgeriou, 2010). However, only some of these methods are designed for operational quality attributes like performance (in contrast to development quality attributes like maintainability) (Bosch and Molin, 1999; Babar et al., 2004). To the best of our knowledge, none of these methods are designed to assess the operational quality attribute *functional safety* of automotive systems.

This paper presents a method to assess the functional safety of existing automotive architecture for cooperative driving, by combining methods from the safety engineering and software architecture domains. Our method has two parts:
(*i*) derive Functional Safety Requirements (FSRs) for cooperative driving scenarios — an extension of our earlier work (Kochanthara et al., 2020);
(*ii*) check whether the (technical) software architecture fulfills the derived functional safety requirements—a combination of techniques (Wu and Kelly, 2004; Preschern et al., 2015; Kazman et al., 1998) adapted from the software architecture domain.

This paper 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 apply our method on the architecture of an academic prototype capable of cooperative driving. The cooperative driving scenario used for demonstration of our method is *platooning*, in which a manually driven vehicle is autonomously followed by a train of vehicles.

The rest of the paper is organized as follows. Section 2 presents the background relevant to the study. Section 3 describes the proposed method to derive FSRs and check for their fulfillment in vehicles' technical software architecture. Section 4 details the application of the proposed approach on an academic prototype for the cooperative driving use case, platooning, and interpreting the results from this case study. Section 5 discusses our implicit assumptions, applicability, and scope of our approach. Section 6 outlines related research. Section 7 presents threats to validity, followed by future research directions in Section 8 and the conclusion in Section 9.

2. Background

In this section, we discuss the three basic concepts upon which we build the contributions of this paper. 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.

2.1. Functional safety

Functional safety is defined as *"an absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems"* (ISO, 2018) 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 (ISO, 2019), 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 (ISO, 2018, 2019). ISO 26262 (ISO, 2018) is the current safety standard with its latest revision from 2018. In contrast, ISO 21448 (ISO, 2019), 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 (IEC, 2010), which outlines the functional safety guidelines for developing electrical/electronic/programmable electronic systems that are used to carry out safety functions (IEC, 2010).

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"* (ISO, 2018).

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 (ISO, 2018). Each safety goal is decomposed into one or more FSRs (ISO, 2018). 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 (Bass et al., 2012).

2.2. Safety tactics and patterns

Architectural tactics encapsulate design decisions that can influence the behavior of a system with respect to a quality attribute (Bass et al., 2012). 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 paper employs safety tactics and patterns (Wu and Kelly, 2004; Preschern et al., 2015) which are architectural tactics and patterns to address safety.

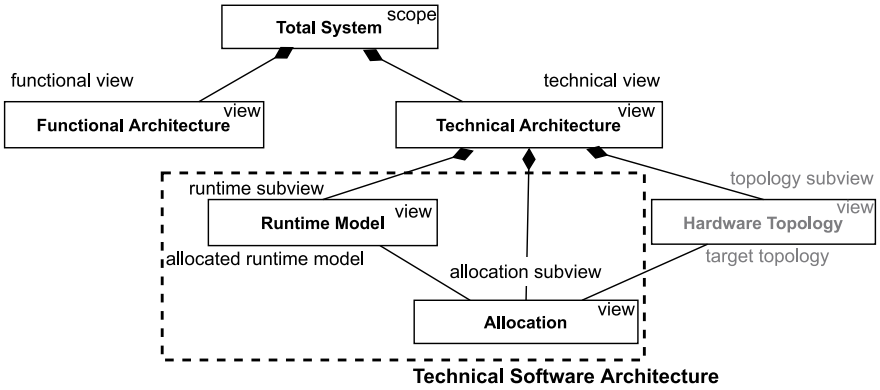


Fig. 1. Functional and technical architecture views and their scope, adapted from Broy et al. (2009). Functional and technical architecture are views of the same system at different architectural abstraction levels, with functional being the highest abstraction level. Runtime model describes system behavior 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.

2.3. Architecture views

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 (ISO, 2018), known as functional architecture view (Broy et al., 2009; Bucaioni and Pelliccione, 2020; Dajsuren, 2015; Staron, 2017). In the automotive domain, the functional architecture view outlines functional composition, functional entities, their interfaces, interactions, inter-dependencies, behavior, and constraints in a vehicle (Broy et al., 2009). 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 (Broy et al., 2009). 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) (Broy et al., 2009; Dajsuren, 2015; Dajsuren and van den Brand, 2019; Staron, 2017). 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 (Broy et al., 2009). Clearly, there is strong conformity between the technical architecture view and the functional architecture view (Broy et al., 2009). A pictorial depiction of these two architectural views is shown in Fig. 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 (Broy et al., 2009). In contrast, other views might lack necessary detail or may be outdated. In the rest of this paper, we discuss the runtime model and allocation part of the technical architecture view, together termed as technical software architecture.

3. Methodology

We propose a method that checks whether the technical software architecture of a vehicle fulfills the FSRs for cooperative driving scenarios. The method consists of two parts: (*i*) derive FSRs for cooperative driving scenarios (see Section 3.1 and Fig. 2), and (*ii*) check whether the derived FSRs are fulfilled in the technical software architecture of a vehicle (see Section 3.2 and Fig. 3).

FSRs for cooperative driving shall be implemented in individual vehicles. The ISO 26262 standard recommends mapping of FSRs (or breaking down FSRs) to individual system architecture components (ISO, 2018). Further, such a mapping is crucial given the complexity and scale of the system. Referring to the existing solutions from the safety engineering discipline (Hommes, 2012), the current methods do not map derived FSRs for cooperative driving scenarios to individual vehicle components (Kochanthara et al., 2020). Our solution bridges this gap by integrating a cooperative 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 3.1.

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 (Bass et al., 2012; Kazman et al., 1998) and employs the safety tactic framework (Wu and Kelly, 2004; Preschern et al., 2015, 2013) to leverage existing architecture knowledge. This part of our method is presented in Section 3.2.

Alongside functional safety, cyber-security is another area that is increasingly addressed together with functional safety (Riel et al., 2018). 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 Fig. 1) is beyond the scope of our method.

3.1. Derive FSRs for cooperative 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 cooperative driving. Note that functional architecture is the overall system architecture (see Fig. 1), which includes both hardware and software components.

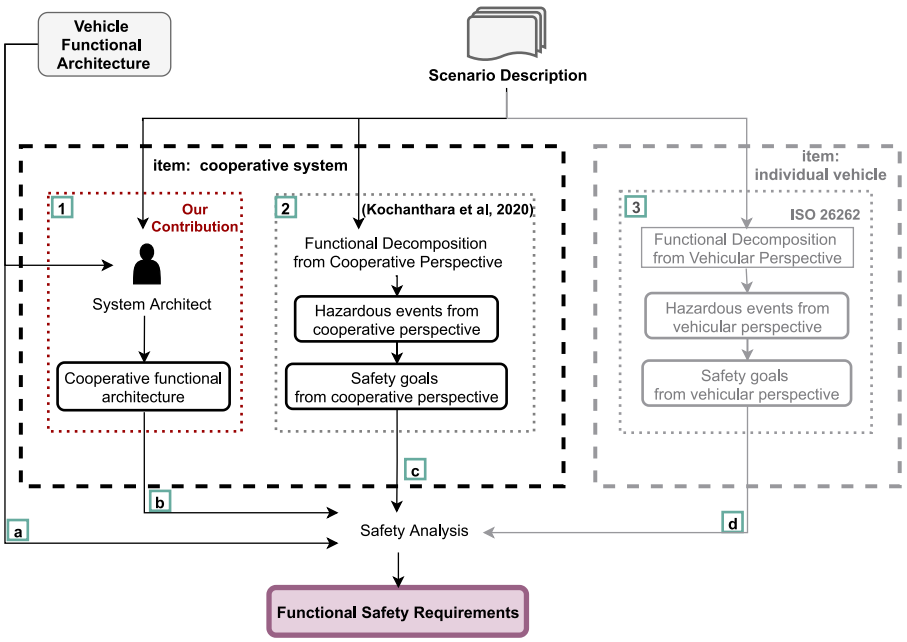


Fig. 2. Method to derive FSRs for cooperative driving scenarios. The gray part on the right is the traditional method from ISO 26262 (ISO, 2018), and the black part on the left is our addition to the traditional approach. System architect represents external entities involved in creating the cooperative architecture.

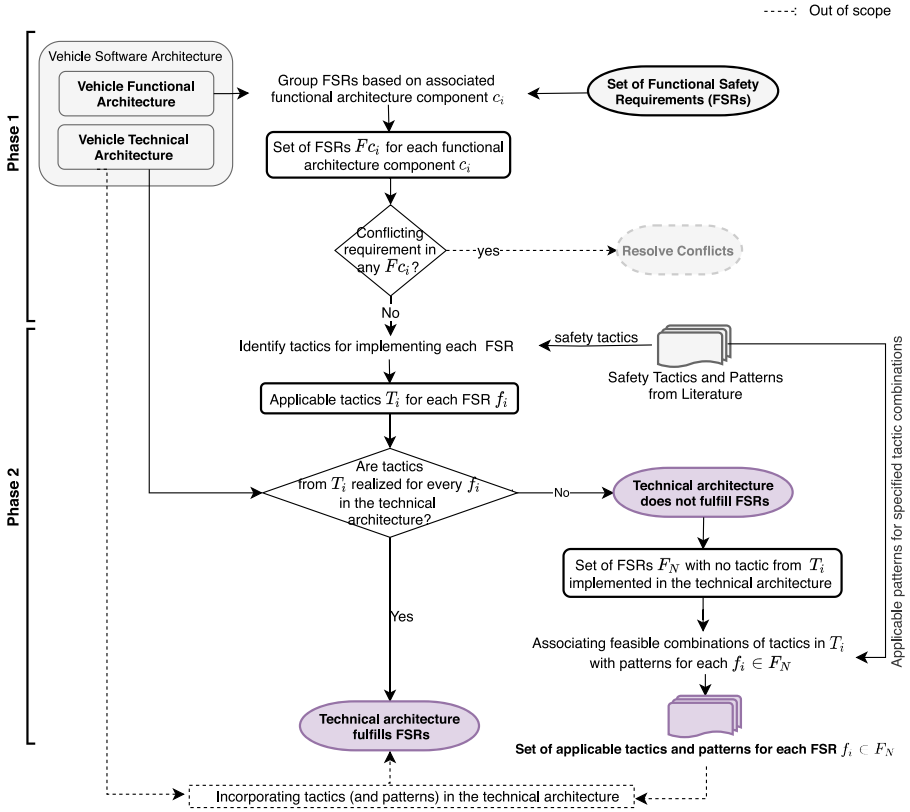


Fig. 3. Method to check the fulfillment of FSRs in technical architecture.

We extend the traditional method outlined by the ISO 26262 standard (ISO, 2018) to derive FSRs for cooperative 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 cooperative system in parallel. Fig. 2 presents an overview of the proposed method. We first outline the traditional method, followed by our prior work on its extension (Kochanthara et al., 2020) and our new contribution. For the rest of the paper, we use the term *vehicular perspective* for an individual vehicle as a unit under consideration

and *cooperative 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) (Lee et al., 1985) or failure mode effect analysis (FMEA) (Stamatis, 2003). 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 (ISO, 2018), 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) (ISO, 2018). This method to derive FSRs is depicted by part 3 and flows *a* and *d* of Fig. 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 (ISO, 2018) for at least a decade (ISO, 2011).

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 (ISO, 2018). 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.

In a cooperative system, a safety goal for one vehicle can lead to an FSR in another vehicle. For example, consider a simple cooperative 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 cooperative perspective.

In the proposed method, we have one item per individual vehicle type, and an item for the entire cooperative system of which the vehicles are part. A cooperative system can have more than one type of vehicle (for example, two vehicles with different functional architectures forming a cooperative system) and other entities like a cloud, enabling cooperative 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 cooperative system, the traditional ISO 26262 analysis described above is applicable. We believe that two items, as shown in Fig. 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 Fig. 2) for each additional item (i.e., each unique functional architecture). In any case, there will only be one cooperative functional architecture and thus only a cooperative item. For the rest of this section, we consider two items: an individual vehicle (representative of all vehicle functional architectures) and the cooperative system.

We propose that FSRs for a cooperative system are derived from: (*i*) safety goals from the vehicular perspective (as in the traditional method), and (*ii*) safety goals from the cooperative perspective. Along these lines, our prior work (Kochanthara et al., 2020) extended the traditional process to derive safety goals for the vehicular perspective to the cooperative perspective (annotated as part 2 in Fig. 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 cooperative functional architecture should be built using individual vehicle functional components. This will preserve the mapping between functional architecture of cooperative system and its implementation view (in the technical architecture of the vehicles). A cooperative functional architecture is required for safety analysis techniques like FTA (Lee et al., 1985) to derive FSRs, by mapping safety goals to components of functional architecture. We propose that the cooperative functional architecture be built from (*i*) the functional architecture of individual vehicles that constitute the cooperative system and (*ii*) the cooperative scenario description of the interaction between individual vehicles. With these requirements, system architects can create a functional architecture of the cooperative 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 Fig. 2; the complete process of deriving FSRs from the cooperative perspective is shown by the labels 1, 2, *b*, and *c*.

In summary, the presented method maps each individual cooperative 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 paper, we assume that each FSR can be mapped to a functional architecture component.

3.2. 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. Fig. 3 depicts an outline of the process.

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 (Lee et al., 1985; Fu, 2018; ISO, 2018). 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 4, 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 Fig. 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 (Preschern et al., 2015; Wu and Kelly, 2004) 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) (Bass et al., 2012; Preschern et al., 2015; Wu and Kelly, 2004) or by matching the FSR description to the descriptions of each tactic (Preschern et al., 2015). 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 (Wu and Kelly, 2004)—an applicable safety tactic for failure containment using redundancy is *diverse redundancy* (Wu and Kelly, 2004). The same tactic can be identified by matching the FSR description to the tactic description (Preschern et al., 2015). 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"* (Preschern et al., 2015)—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 cooperative 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 (Preschern et al., 2015). 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.

4. Case study

This section presents an application of the proposed method on a cooperative 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 (Kochanthara, 2021).

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 (Liang et al., 2015); (*ii*) improve safety—for example, by preventing rear end collisions by enabling platoon-wide braking (Pelliccione et al., 2020); 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 cooperative 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]

A simplified functional architecture of i-CAVE demonstrator is shown in Fig. 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. Some of the functional architecture components are grouped to classes based on their functionality (as shown in Fig. 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:

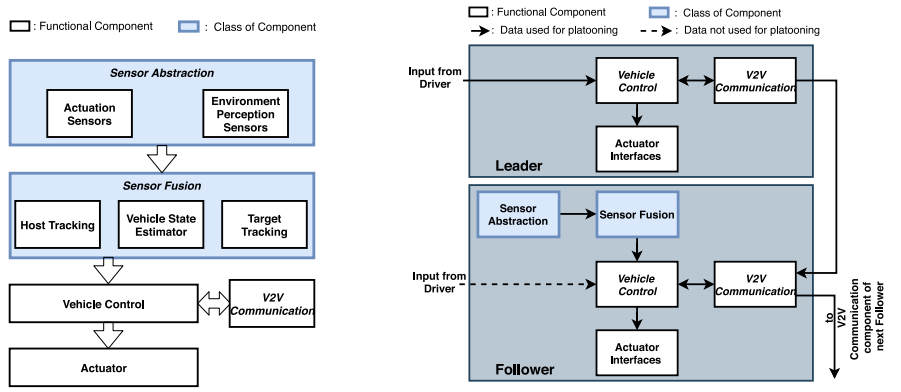
(*a*) *Sensor abstraction* consists of hardware sensors and their encapsulation via its software interfaces. Two classes of sensors are functionally distinguished: (*i*) *actuation sensors* that monitor

[5] https://i-cave.nl/.

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

[7] https://bit.ly/AdvantechARK-3520P.

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



(a) Vehicle functional architecture (b) Cooperative functional architecture for platooning

Fig. 4. A simplified functional architecture of i-CAVE demonstrator and the platooning architecture (cooperative architecture) derived from it.

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 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.

4.1. Derive FSRs for platooning

Following are the steps in the first part of our method, depicted in Fig. 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 cooperative perspective. These functions are listed in Table 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*) (IEC, 2010) 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 lead to crash inside a platoon. The list of hazards is available online (Kochanthara, 2021) and the count of hazards derived from each function is shown in Table 1.

These 57 hazards (26 from cooperative perspective and 31 from vehicular perspective) when combined with operational modes (7 from cooperative perspective and 6 from vehicle perspective) and operational situations (2 per perspective) resulted in 340 hazardous events, 140 from vehicle perspective and 200 from cooperative 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 cooperative 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 cooperative 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,

Table 1
Hazards for functions identified from platooning description.

Cooperative 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

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 (Kochanthara, 2021).

Cooperative functional architecture: Fig. 4b shows a simplified cooperative functional architecture for platooning with functional components for platooning as well as the working of vehicles within a platoon at the functional level. The cooperative 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 cooperative functional architecture contains the same functional components as the vehicular functional architecture (see Figs. 4a and 4b), but only the components that are used to accomplish the cooperative 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 abstraction* and *sensor fusion* class of functional components are not used for cooperative driving functions. Also, these functional components are not used for leader's own driving functions since the leader is manually driven. Therefore, in the cooperative functional architecture, in the leader block, these components are not shown for leader (see the leader block at the top of Fig. 4b).

Safety analysis: Finally, FSRs are derived by mapping safety goals to the functional architectures using fault tree analysis (FTA) (Lee et al., 1985). The FTA generated 16 FSRs from the vehicle perspective and 15 FSRs from the cooperative perspective. i.e., 31 in total. The count of FSRs for each functional component is presented in Fig. 5 along with some example FSRs in the second column of Table 2.

4.1.1. 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 cooperative 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 cooperative 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 cooperative perspective) as assigned with ASIL D. This difference in ASILs between safety goals and FSRs is caused by the fact that most functional safety requirements are related to multiple functional safety goals, and FSRs inherit the highest ASIL of their related safety goals.

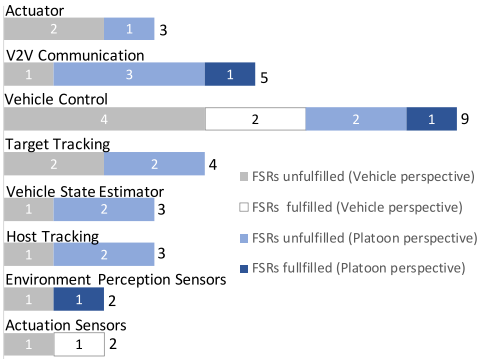


Fig. 5. The 31 FSRs, grouped by associated functional component. The total FSRs for each group is shown at the end of each stacked bar.

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 Serban et al. (2018) 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 cooperative and vehicular perspectives. Our case study, however, did not result in any overlapping FSRs.

4.2. Check fulfillment of FSRs

Following are the steps in second part of our method, depicted in Fig. 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 2 (see details in the third–fifth row, first and second column). The overall count of FSRs grouped on associated component is shown in Fig. 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 (Kochanthara, 2021).

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 (Preschern et al., 2015; Wu and Kelly, 2004): *simplicity, substitution, sanity check, condition monitoring, comparison, diverse redundancy, replication redundancy, repair, degradation, voting, override, barrier* and *heartbeat* (Preschern et al., 2015; Wu and Kelly, 2004).

Each safety tactic has an aim and a description of its scope (Preschern et al., 2015). For example, the aim of safety tactic

Table 2
FSRs that are found to be fulfilled in the technical architecture of i-CAVE demonstrator. FSRs in blue cells are derived from cooperative (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*.

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."* Preschern et al. (2015).

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 2. Table 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 (Kochanthara, 2021).

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 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 (Kochanthara, 2021).

For each functional component, Fig. 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 4.1 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 cooperative 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.

4.2.1. 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 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 (Preschern et al., 2015). 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 cooperative perspective, 12 were unfulfilled. Notably, we found almost as many unfulfilled FSRs from the vehicle perspective as from the cooperative 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 (2338 mm × 1381 mm) to be used in bicycle lanes, is lightweight (gross weight of 690 kg) 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 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 cooperative perspective to improve the i-CAVE technical architecture.

In summary, our case study found unfulfilled FSRs from the cooperative 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 (ISO, 2018). The current safety engineering methods to derive FSRs are outlined by ISO 26262 standard (ISO, 2018; Nilsson et al., 2013) which lacks the cooperative perspective. It provided valuable insights in the context of i-CAVE project. In our case study, we found 15 FSRs from the cooperative 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 (Dajsuren and Loupias, 2019; Nilsson et al., 2013; Saberi et al., 2018) in showing that the current safety standard misses FSRs from the cooperative perspective.

The results of our case study show better coverage of safety goals by providing additional FSRs as compared to the ISO 26262 process (ISO, 2018). The current safety engineering methods to derive FSRs are outlined by ISO 26262 standard (ISO, 2018; Nilsson et al., 2013) which lacks the cooperative perspective. In our case study, we found 15 FSRs from the cooperative perspective, making up 48% of all FSRs. Our results corroborate the existing body of knowledge (Dajsuren and Loupias, 2019; Nilsson et al., 2013; Saberi et al., 2018) in showing that the current safety standard misses FSRs from the cooperative perspective.

5. 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 cooperative driving scenarios in real-life. Below we discuss the implicit assumptions in our method, the scalability, generalizability, and the scope of our method.

5.1. Assumptions

The proposed method borrows some assumptions applicable to single-vehicle and applies them to cooperative 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 (Lee et al., 1985). This separation is also underlined by the product development standard in the automotive domain—ISO26262 (ISO, 2011, 2018) and automotive architecture frameworks (Broy et al., 2009). Nonetheless, the applicability of this assumption in cooperative driving is not established.

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 (Broy et al., 2009). 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 (Bass et al., 2012). Mature architecture assessment methods like ATAM also rely on this assumption, albeit in the context of tactics for the quality attributes they focus on Kazman et al. (1998) and Bass et al. (2012). Nonetheless, the applicability of this assumption in the automotive domain is not established.

5.2. Applicability

Our case study presents a simplified version of real-life cooperative driving use-cases. In real-life, the proposed method should work on a bigger scale and apply to cooperative 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 cooperative 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 cooperative driving function. This ensures that the safety requirements for cooperative 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 cooperative 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 cooperative driving functionalities can be beyond participating vehicles. One such example is cloud communication. The cooperative architecture and corresponding item definition in the first phase of our approach are specifically introduced to ensure that all the entities involved in enabling cooperative 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 cooperative architecture.

Fail-operational and fail-safe designs: Our work is designed for cooperative systems irrespective of their operational design

9 https://www.renault.co.uk/electric-vehicles/twizy/specifications.html.
10 https://www.tesla.com/model3.
11 https://i-cave.nl/.

domain and whether they are designed to be fail-operational or fail-safe (Hasan, 2020; Sawade et al., 2018). The proposed method is generic for both fail-operational as well as fail-safe systems. Our method ensures that both cooperative and vehicular perspectives are covered while deriving FSRs.

6. Related work

The proposed method has two parts: (i) deriving FSRs for cooperative 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.

6.1. 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" (Bass et al., 2012) 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, reusability) (Bosch and Molin, 1999). This paper focuses on *functional safety* as an operational quality attribute.

The software architecture assessment techniques proposed for operational quality attributes (Babar et al., 2004; Dobrica and Niemela, 2002) 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 (Bengtsson et al., 2004). The assessment techniques that use mathematical modeling mainly focus on reliability and performance as quality attributes (Roy and Graham, 2008). Some studies have shown that these techniques are not scalable and hence not suitable for complex systems of systems like cooperative driving systems that are built by multiple inter-disciplinary teams (Roy and Graham, 2008).

The prominent scenario-based architecture assessment methods for operational quality attributes are the Architecture Tradeoff Analysis Method (ATAM) (Kazman et al., 1998; Bass et al., 2012), Scenario-Based software Architecture Re-engineering (SBAR) (Bengtsson and Bosch, 1998), Software architecture Comparison Analysis Method (SCAM) (Stoermer et al., 2003), Domain Specific software Architecture comparison Model (DoSAM) (Bergner et al., 2005), and Pattern-Based Architecture Reviews (PBAR) (Harrison and Avgeriou, 2010).

PBAR is designed for light weight evaluation, primarily performed on small projects with some case studies on projects with at most 10 developers (Harrison and Avgeriou, 2010, 2013). It is not suitable for evaluation of complex safety-critical systems that we assess in this paper (Harrison and Avgeriou, 2010). SCAM and DoSAM are designed for comparing different architectures rather than assessing an individual architecture (Stoermer et al., 2003; Bergner et al., 2005). 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 (Bengtsson and Bosch, 1998) 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 (ISO, 2011, 2018). Moreover, SBAR suggests scenario-based techniques for development quality attributes and simulation based assessment for operational quality attributes (Dobrica and Niemela, 2002). 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 (Bass et al., 2012). 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 (Bass et al., 2012). Note that case studies of ATAM's application to safety critical domains like avionic systems (Barbacci et al., 2003) 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 systems of systems context, 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 cooperative automotive systems.

6.2. 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. (2014) presents a model-based method to define FSRs given safety goals while Abdulkhaleq et al. (2017) uses system theory for safety analysis.

Studies on cooperative driving systems try to replicate the mechanisms from an individual vehicle perspective. For example, Oscarsson et al. (2016) 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 (Stoltz-Sundnes, 2019). Our study closely follows the study by Saberi et al. (2018) in deriving FSRs from cooperative 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 (Cuenot et al., 2014). Martin et al. (2020) uses architecture patterns to incorporate FSRs in the design phase. Sljivo et al. (2020) presents a methodology 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 (Althoff and Dolan, 2014; Bhatti et al., 2016; Mallozzi et al., 2016).

Ensuring fulfillment of FSRs as part of a cooperative system is challenging (Pelliccione et al., 2020). A majority of works on cooperative systems proposes a reference architecture from a system of systems viewpoint (Pelliccione et al., 2020). Some other

solutions look at specific architecture components in specific cooperative scenarios, thereby missing high-level insights (Dajsuren and Loupias, 2019).

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

7. 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 (Zalewski et al., 2017) 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 cooperative 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 (Lee et al., 1985) as the safety analysis technique. The choice of other techniques, like failure mode effect analysis (Stamatis, 2003), 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 (Preschern et al., 2013, 2015). This list of safety tactics is not complete and defines the scope of our case study.

8. Future work

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

Cyber-security alongside functional safety: Cyber-security is a prominent directions to explore in cooperative driving alongside functional safety. The connected nature of cooperative 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 cooperative driving scenarios. Alternative methods to check for unfulfilled FSRs will be an interesting direction to explore.

9. Conclusion

This paper 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 is a combination of methods adapted from the safety engineering and software architecture domains. We show the usability of our method for a cooperative driving scenario, platooning, on a real-life 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.

CRediT authorship contribution statement

Sangeeth Kochanthara: Conceptualization, Methodology, Investigation, Writing - original draft, Writing - review & editing. **Niels Rood:** Conceptualization, Investigation, Validation, Writing - original draft. **Arash Khabbaz Saberi:** Resources, Methodology, Validation. **Loek Cleophas:** Conceptualization, Methodology, Writing - review & editing, Supervision. **Yanja Dajsuren:** Methodology, Writing - review & editing, Supervision, Funding acquisition. **Mark van den Brand:** Conceptualization, Methodology, Writing - review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abdulkhaleq, A., Wagner, S., Lammering, D., Boehmert, H., Blueher, P., 2017. Using STPA in compliance with ISO 26262 for developing a safe architecture for fully automated vehicles. arXiv preprint arXiv:1703.03657.

Althoff, M., Dolan, J.M., 2014. Online verification of automated road vehicles using reachability analysis. IEEE Trans. Robot. 30 (4).

Alvarez, P., Lerga, I., Serrano, A., Faulin, J., 2017. Considering congestion costs and driver behaviour into route optimisation algorithms in smart cities. In: International Conference on Smart Cities. Springer, pp. 39–50.

Babar, M.A., Zhu, L., Jeffery, R., 2004. A framework for classifying and comparing software architecture evaluation methods. In: 2004 Australian Software Engineering Conference. Proceedings. IEEE.

Barbacci, M., Clements, P.C., Lattanze, A., Northrop, L., Wood, W., 2003. Using the Architecture Tradeoff Analysis Method (ATAM) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study. Tech. Rep., Carnegie-mellon Univ Pittsburgh PA Software Engineering Inst.

Bass, L., Clements, P., Kazman, R., 2012. Software Architecture in Practice. Addison-Wesley Professional.

Beckers, K., Côté, I., Frese, T., Hatebur, D., Heisel, M., 2014. Systematic derivation of functional safety requirements for automotive systems. In: (SafeComp). Springer.

Bengtsson, P., Bosch, J., 1998. Scenario-based software architecture reengineering. In: Proceedings. Fifth International Conference on Software Reuse. IEEE.

Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H., 2004. Architecture-level modifiability analysis (ALMA). J. Syst. Softw. 69 (1–2).

Bergner, K., Rausch, A., Sihling, M., Ternité, T., 2005. Dosam–domain-specific software architecture comparison model. In: Quality of Software Architectures and Software Quality. Springer.

Bhatti, Z.E., Roop, P.S., Sinha, R., 2016. Unified functional safety assessment of industrial automation systems. IEEE Trans. Ind. Inf..

Bosch, J., Molin, P., 1999. Software architecture design: evaluation and transformation. In: Proceedings ECBS'99. IEEE Conference and Workshop on Engineering of Computer-Based Systems. IEEE.

Broy, M., Gleirscher, M., Kluge, P., Krenzer, W., Merenda, S., Wild, D., 2009. Automotive architecture framework: Towards a holistic and standardised system architecture description. IEEE Comput. 42 (12).

Bucaioni, A., Pelliccione, P., 2020. Technical architectures for automotive systems. In: (ICSA). IEEE.

Cuenot, P., Ainhauser, C., Adler, N., Otten, S., Meurville, F., 2014. Applying model based techniques for early safety evaluation of an automotive architecture in compliance with the ISO 26262 standard.

Dajsuren, Y., 2015. On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems (Ph.D. thesis). Department of Mathematics and Computer Science, Eindhoven University of Technology.

Dajsuren, Y., van den Brand, M. (Eds.), 2019. Automotive Systems and Software Engineering: State of the Art and Future Trends. Springer International Publishing.

Dajsuren, Y., Loupias, G., 2019. Safety analysis method for cooperative driving systems. In: (ICSA). IEEE.

Davila, A., 2013. Report on fuel consumption. SARTRE, Deliverables.

Dobrica, L., Niemela, E., 2002. A survey on software architecture analysis methods. IEEE Trans. Softw. Eng. 28 (7).

Fu, Y., 2018. Fault Injection Mechanisms for Validating Dependability of Automotive Systems (Master's thesis). Eindhoven University of Technology.

Harrison, N., Avgeriou, P., 2010. Pattern-based architecture reviews. IEEE Softw. 28 (6).

Harrison, N.B., Avgeriou, P., 2013. Using pattern-based architecture reviews to detect quality attribute issues-an exploratory study. In: Transactions on Pattern Languages of Programming III. Springer.

Hasan, S., 2020. Fail-Operational and Fail-Safe Vehicle Platooning in the Presence of Transient Communication Errors (Ph.D. thesis). Mälardalen University.

Hommes, Q.V.E., 2012. Review and Assessment of The ISO 26262 Draft Road Vehicle-Functional Safety. Tech. Rep., SAE Technical Paper.

IEC, 2010. IEC Functional Safety and IEC 61508. Standard, International Electrotechnical Commission.

ISO, 2011. ISO 26262: 2011 - Road Vehicles – Functional Safety. Standard, International Organization for Standardization.

ISO, 2018. ISO 26262: 2018 - Road vehicles – Functional safety. Standard, International Organization for Standardization.

ISO, 2019. ISO/PAS 21448: 2019 - Road vehicles — Safety of the intended functionality. Standard, International Organization for Standardization.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J., 1998. The architecture tradeoff analysis method. In: Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems. IEEE.

Kochanthara, S., 2021. A case study on iso 26262 extension for connected driving. GitHub repository https://github.com/SangeethNila/casestudy_ISO26262_extension_connected_driving.

Kochanthara, S., Rood, N., Cleophas, L., Dajsuren, Y., van den Brand, M., 2020. Semi-automatic architectural suggestions for the functional safety of cooperative driving systems. In: (ICSA-C). IEEE.

Lee, W.-S., Grosh, D.L., Tillman, F.A., Lie, C.H., 1985. Fault tree analysis, methods, and applications a review. IEEE Trans. Reliab. 34 (3).

Liang, K.-Y., Mårtensson, J., Johansson, K.H., 2015. Heavy-duty vehicle platoon formation for fuel efficiency. Trans. Intell. Transp. Syst..

Mallozzi, P., Pelliccione, P., Knauss, A., Berger, C., Mohammadiha, N., 2019. Autonomous vehicles: State of the art, future trends, and challenges. In: Automotive Systems and Software Engineering. Springer.

Mallozzi, P., Sciancalepore, M., Pelliccione, P., 2016. Formal verification of the on-the-fly vehicle platooning protocol. In: (SERENE). Springer.

Martin, H., Ma, Z., Schmittner, C., Winkler, B., Krammer, M., Schneider, D., Amorim, T., Macher, G., Kreiner, C., 2020. Combined automotive safety and security pattern engineering approach. Reliab. Eng. Syst. Saf..

Nilsson, J., Bergenhem, C., Jacobson, J., Johansson, R., Vinter, J., 2013. Functional Safety for Cooperative Systems. Tech. Rep., SAE Technical Paper.

Oscarsson, J., Stolz-Sundnes, M., Mohan, N., Izosimov, V., 2016. Applying systems-theoretic process analysis in the context of cooperative driving. In: (SIES).

Pelliccione, P., Knauss, E., Ågren, S.M., Heldal, R., Bergenhem, C., Vinel, A., Brunnegård, O., 2020. Beyond connected cars: A systems of systems perspective. Sci. Comput. Program. 191.

Ploeg, J., 2014. Analysis and design of controllers for cooperative and automated driving.

Preschern, C., Kajtazovic, N., Kreiner, C., 2015. Building a Safety Architecture Pattern System. In: EuroPLoP '13, ACM.

Preschern, C., Kajtazovic, N., Kreiner, C., et al., 2013. Catalog of safety tactics in the light of the IEC 61508 safety lifecycle. In: Proceedings of VikingPLoP 2013 Conference.

Riel, A., Kreiner, C., Messnarz, R., Much, A., 2018. An architectural approach to the integration of safety and security requirements in smart products and systems design. CIRP Ann. 67 (1), 173–176.

Roy, B., Graham, T.N., 2008. Methods for Evaluating Software Architecture: A Survey. School of Computing TR 545.

Saberi, A.K., Barbier, E., Benders, F., van den Brand, M., 2018. On functional safety methods: A system of systems approach. In: (SysCon). IEEE.

Sawade, O., Schulze, M., Radusch, I., 2018. Robust communication for cooperative driving maneuvers. IEEE Intell. Transp. Syst. Mag. 10 (3), 159–169.

Serban, A., Poll, E., Visser, J., 2018. A standard driven software architecture for fully autonomous vehicles. In: (ICSA-C).

Sljivo, I., Uriagereka, G.J., Puri, S., Gallina, B., 2020. Guiding assurance of architectural design patterns for critical applications. J. Syst. Archit..

Stamatis, D.H., 2003. Failure Mode and Effect Analysis: FMEA from Theory to Execution. Quality Press.

Staron, M., 2017. Automotive software architectures. Automot. Softw. Archit..

Stoermer, C., Bachmann, F., Verhoef, C., 2003. SCAM: The Software Architecture Comparison Analysis Method. Tech. rep., Carnegie-mellon Univ Pittsburgh PA Software Engineering Inst.

Stoltz-Sundnes, M., 2019. Stpa-inspired safety analysis of driver-vehicle interaction in cooperative driving automation.

Trego, T., Murray, D., 2010. An analysis of the operational costs of trucking. In: Transportation Research Board 2010 Annual Meetings CD-ROM, Vol. 18, Washington, DC, p. 20.

Wu, W., Kelly, T., 2004. Safety tactics for software architecture design. In: (COMPSAC). IEEE.

Zalewski, A., Borowa, K., Ratkowski, A., 2017. On cognitive biases in architecture decision making. In: (ECSA). Springer.

Sangeeth Kochanthara is a Ph.D. candidate in the Software Engineering and Technology group at the Eindhoven University of Technology, the Netherlands. His research focuses on functional safety, architecture assessment, and formal verification in the context of automotive systems. He graduated his masters with gold medal from Indraprastha Institute of Information Technology, Delhi (IIITD), India. He has worked at Research Centre in Real-Time and Embedded Computing Systems, Porto, Portugal and Program Analysis group at IIITD.

Niels Rood Master Program in Computer Science and Engineering at Technical University of Eindhoven (TU/e) in The Netherlands. In 2019 he graduated within the Software Engineering and Technology Research Group on Functional Safety Analysis and Safety Pattern Application on i-CAVE. He is currently pursuing a Professional Doctorate in Engineering (PDEng) in Software Technology at TU/e.

Arash Khabbaz Saberi was born on 27-04-1988 in Tehran, Iran. After finishing B.Sc. in 2010 at Shahid Beheshti University in Tehran, Iran, he studied Master Program of Embedded Systems at Technical University of Eindhoven (TU/e) in The Netherlands. In 2013 he graduated within the Systems Control group on Control Relevant MIMO Parametric Identification. In 2015, he completed the Professional Doctorate in Engineering (PDEng) in Automotive System Design at TU/e. From 2015 he started a Ph.D. project at TU/e in collaboration with the Integrated Vehicle Safety (IVS) Department of TNO, of which the results are presented in this dissertation. Since 2015 he is employed at TNO, IVS.

Loek Cleophas is an assistant professor in the Software Engineering Technology (SET) cluster at Eindhoven University of Technology (TU/e) and a research fellow at Stellenbosch University, South Africa. He obtained his doctorate in computer science and engineering at TU/e. He is also managing director of the Dutch research school on programming and algorithmics (IPA).His research has varied from model-driven virtualization of high-tech systems, to generating efficient algorithm toolkits based on algorithm taxonomies, mainly for pattern

matching on text and trees. More recent work focuses on analyzing models and large collections of models and extracting variability and commonality information from them, and on consistency of models. He worked in industry in the Netherlands and the USA, and at universities in South Africa, Sweden, and Germany, on research funded by various national and international projects as well as by industrial partners.

Yanja Dajsuren is a program director of the PDEng Software Technology program and assistant professor at the Software Engineering and Technology (SET) group, Eindhoven University of Technology (TU/e). Prior to her Ph.D. research in the area of automotive software architecture and engineering field, she worked as a scientist and senior scientist for half a decade working on various advanced software development projects at the Philips Research Lab, NXP Semiconductors (former Philips Semiconductors), and Virage Logic. She is currently working on system/software architecture, safety and quality related topics of autonomous and cooperative driving vehicles as well as cooperative-intelligent transport systems.

Mark van den Brand is a full professor of Software Engineering and Technology in the Department of Mathematics and Computer Science, and a visiting professor at Royal Holloway, University of London. His current research activities are on model driven engineering, domain specific languages, meta-modeling, model management, digital twins, and automotive software engineering. His research is industry inspired; he works with most of the high-tech companies in the Eindhoven (The Netherlands) region. He has been an invited lecturer and keynote speaker at various conferences, workshops and doctoral schools. He was and is member of PCs on workshops and conferences related to software engineering, language engineering, rewriting, reverse engineering, and software maintenance. He initiated the special issues of Science of Computer Programming devoted to academic software development (Experimental Software and Toolkits), and since 2007 has been guest editor of six of these. He is on the editorial board of the journals Science of Computer Programming, Open Computer Science, and Computer Languages (COLA). He is Editor-in-Chief of the Journal on Automotive Software Engineering. He is deputy Editor-in-Chief of platinum open access journal JOT.