

A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models

Stefan Wagner*, Bernhard Schätz†, Stefan Puchner‡, and Peter Kock§

**Technische Universität München, Garching, Germany*

Email: wagnerst@in.tum.de

†*fortiss GmbH, Munich, Germany*

Email: schaezt@fortiss.org

‡*Capgemini sd&m AG, Munich, Germany*

Email: stefan.puchner@capgemini.com

§*MAN Nutzfahrzeuge AG, Munich, Germany*

Email: peter.kock@man.eu

Abstract—Driven by market needs and laws, automotive manufacturers develop ever more feature-rich and complex vehicles. This new functionality plays even an active role in driving, what poses many new challenges on assuring the safety of the vehicle. Safety cases constitute a proven technique to systematically use existing information about a system, its environment, and development context to show its safety. We construct the safety case for a cruise control system describe in a case study in the automotive domain with a special consideration of existing domain-specific models. In the case study, we identify generic safety case modules and several reoccurring patterns, which will simplify the development of future automotive safety cases.

Keywords—safety case; model-based development; automotive

I. INTRODUCTION

The market as well as law makers demand more and more features and hence ever more complex automotive vehicles. Manufacturers combine mechanical, electric/electronic, and software parts to implement these features. Especially software plays a major role in the implementation of the functionality, which ranges from embedded control systems to entertainment systems with a rich user interface. The BMW 7 series, for instance, implements about 270 user functions distributed over up to 67 embedded control units, amounting to about 65 megabytes of binary code [1]. In particular software-based functions that interfere with driving gain more importance, because they can (1) decrease the number of crashes and crash effects and (2) increase the driver’s comfort. Examples for such systems are the anti-lock braking system (ABS), electronic stability programme (ESP), automatic parking systems, or systems for collision avoidance.

The many features, their interactions, and especially their complexity pose a challenge to the development and maintenance processes of the automotive industry. Particularly the assurance of the safety of each of these features and finally the complete vehicle is demanding.

A. Problem statement

There are several applicable standards for developing safety-critical electronic systems for the automotive domain such as IEC 61508-3 [2] or the currently developed ISO WD 26262. The suggestions for safety assurance contained therein are not sufficient for software. We identify two major problems: (1) some suggestions are not scientifically investigated to have an influence on safety and (2) they are mostly limited to prescribing activities and techniques that have to be used; they constrain the process. The underlying assumption, however, that a good development process alone produces safe software is questionable [3].

B. Research objective

The overall objective of this research is to establish usable safety assurance methods for automotive systems involving software. In particular, we aim to discover reusable structures, patterns, and processes in safety assurance to support its practical application.

C. Contribution

Safety cases have gained acceptance, for example in the avionics industry, as a systematic means to use all existing information to construct a structured argument for the safety of a system. We use safety cases in a case study in the automotive domain exploiting the already existing models of software functions, the vehicle, the driver, and their environment. The case study analyses the construction of a safety case for a real software-based component in a commercial vehicle at MAN Nutzfahrzeuge AG. The result is a generic safety case architecture and a set of reoccurring patterns involving the models that we use to build a safety case.

D. Context

The case study concentrates on the automotive domain and is performed at MAN Nutzfahrzeuge AG. We construct a safety case for the cruise control unit of a truck.

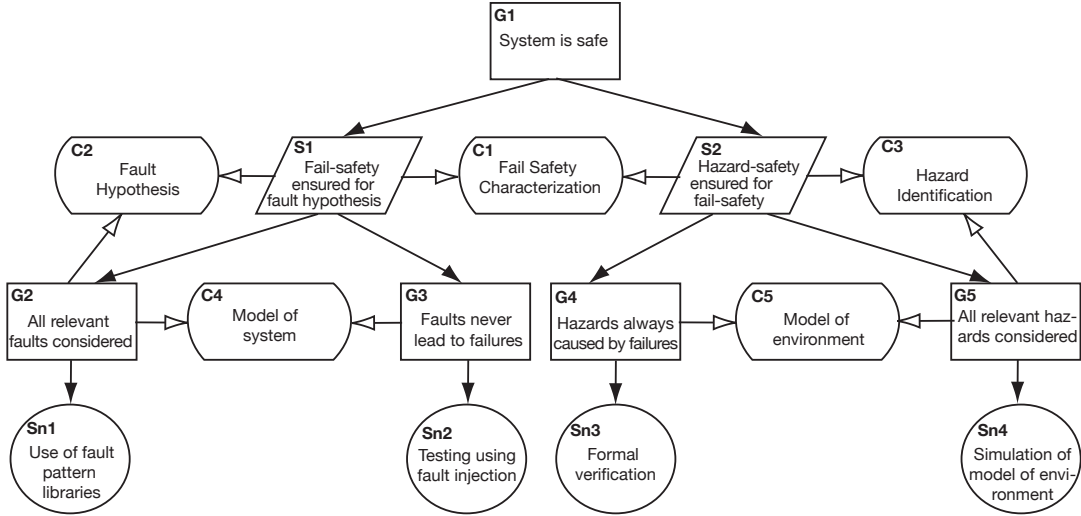


Figure 1. A simplified example of a safety argument using GSN and context information from models

E. Outline

We base the presentation of the case study on the proposal by Runeson and Höst [4]. We start by giving background information (Section II) on safety cases and the models used in the automotive domain. Section III describes existing studies and Section IV explains the study design. The results in Section V are structured in safety argument modules and safety case patterns. Finally, we discuss conclusions in Section VI.

II. BACKGROUND

We give background information on the two main types of artefacts used in the case study: safety cases and models employed in the automotive domain.

A. Safety cases

A safety case is a structured line of arguments that shows that the system under consideration is safe. It addresses the difficulty of combining a large variety of information needed to form this argument. A single safety assurance method is never able to show the complex issue of safety completely. Hence, formal verification, statistical testing, process conformance, and other information must be integrated for a convincing argument.

Bishop and Bloomfield [5] define a safety case as “A documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment”. For this, a safety case comprises three parts: (1) the safety goal that has to be achieved, (2) the available evidence for achieving this goal, and (3) the structured argument, which establishes the systematic relationship between the evidence and the goals. Furthermore, the goal is broken down into smaller, more

detailed goals (*sub-goals*) that can be shown more directly by an argument.

A description technique that has proven to be useful for constructing safety arguments is the *Goal Structuring Notation* (GSN) [6]. It reduces some problems, such as ambiguity, of purely textual descriptions. An example abstract safety argument using GSN is shown in Figure 1. In this example the overall goal **G1** is that the system is safe. This is intended to be achieved by two strategies: **S1** is to ensure there are no failures in terms of deviations from the intended safety-critical functionality even in presence of faults, and **S2** arguments by showing that there are also no hazards in the absence of failures. Hence, defect hypotheses as well as possible hazards must be identified. This is depicted by the two contexts **C2** and **C3**. From the strategies, four refined goals are derived. **G2** expresses that all relevant defects need to be considered, which is shown by using fault pattern libraries in solution **Sn1**. **G3**, **G4**, and **G5** describe further goals for failures and hazards, which are met by the exemplary solutions **Sn2**, **Sn3**, and **Sn4**. Important for the following is also that there are several nodes with context information like **C1**, the characterisations of failures, and models of the behaviour of the system (**C4**) and the environment (**C5**). Not shown here are *justifications*, which would be annotated with **J**.

A variety of information needs to be considered when constructing a safety case. From a methodological point of view, we need to identify safety requirements and hazards. But this is not sufficient. It is also necessary to build models that form the context for more detailed arguments, for example about specific components. Moreover, the models must be able to handle deterministic as well as probabilistic information and need not only to include the system itself but also its environment.

B. Models in automotive

The use of models in the development of automotive embedded software systems has increased substantially over the last years, especially due to tools like ASCET SD¹ and Matlab/Simulink², which support a model-based development process. In such a process, three forms of model application can be distinguished:

Functional models: These models describe the intended functionality provided by the embedded system. In general, the models for this purpose are either models of an ideal function, i.e., independent of a specific realisation in software/hardware and not discretised in the time/value domain; typical examples are Simulink models. Or these are models of a software function, i.e., discretised in the time/value domain; typical examples are TargetLink³ models.

The first class of these models serve in early validation and verification via system simulation, e.g., in model-in-the-loop simulations. The second class takes part in verification as well, e.g., in a software-in-the-loop simulation; more importantly, however, developers implement the system with these models by generating production source code.

Environment models: These models describe characteristics and the behaviour of (a part of) the environment the system is embedded in, or an abstraction thereof [7]. These models of the environment cover aspects of the vehicle, the driver, or even the surroundings of the vehicle including other cars. This form of models supports early validation and verification even before the vehicle itself is available.

Platform models: These models describe the functionality of the technical platform, i.e., the general models of the electric/electronic hardware, which implement the intended functionality of the system under development. They may cover aspects of regular as well as faulty behaviour. This form of models either supports validation and verification before the technical platform is available, or mechanises the deployment of the software to the hardware platform they describe.

The first form of models is in some automotive domains the state of the art. The second form of application is regularly used for in-the-loop tests of specific automotive subdomains like power train, while it is becoming an accepted practice in other domains like chassis electronics and the human-machine interface. The third form of application, especially to systematically investigate platform failures and their effect on the functionality of the overall systems, is currently in use for isolated fault classes in in-the-loop tests.

III. RELATED WORK

A general method for structuring safety cases are safety case modules introduced by Kelly [8]. They encapsulate

¹<http://www.etas.com/>

²<http://www.mathworks.com/>

³Subset of Simulink by dSpace (<http://www.dspace.de/>)

parts of a safety case argument and offer an interface in the form of goals or context nodes, similar to the module concept in software engineering. Modules enable isolation of change, reuse of sub-arguments, and possibly information hiding. We make use of safety case modules in their simplest form in the case study as we describe the safety case architecture analogously to the system decomposition.

Kelly and McDermid [9] propose safety case patterns as a means to reuse common structures in safety cases. We use them to identify reusable parts of our safety case and provide them as building blocks for future automotive safety cases.

Ridderhof, Groß, and Dörr [10] built a safety case for an automotive application, but only to show traceability. They do not discuss how the safety case should be integrated in detail with typical automotive models.

Törner and Öhman [11] performed a case study on safety cases in the automotive domain. They analysed with qualitative methods why and how safety cases should be used. In contrast, our case study investigates one specific application of safety cases in the automotive domain in detail.

Chen et al. [12] describe an approach that is close to what we followed in the case study. They relate safety cases with models on different levels of abstractions. Their approach, however, is limited to their EAST-ADL2 architecture description language. As we use mostly existing models, such as Simulink and Stateflow models, our case study resembles automotive practice more closely.

Habli et al. [13] recently proposed how SysML models influence a safety case. This work is similar to ours in many respects, because their safety cases also use models as a basis. In contrast, our work focuses on the detailed analysis of a case study with the aim to identify a suitable safety case architecture and safety case patterns.

We presented initial ideas on the use of models in safety cases in a position paper [14] that constitutes the basis of this current paper. We extend the position paper by defining concrete relationships to models in the automotive domain, identifying patterns, and using the approach in a comprehensive case study.

IV. CASE STUDY DESIGN

The case study design contains the research questions that drive the case study, how we selected the case and subjects, collected and analysed the data, as well as how we ensure the validity of our results.

A. Research questions

Safety cases for non-trivial systems can become huge and to keep them manageable, a structuring into appropriate parts is necessary. Kelly [8] introduced *safety case* or *argument modules* for this high-level structure. We employ these modules and aim to identify suitable modules in the automotive domain.

RQ 1: What are suitable safety case modules in the automotive domain?

A further technique to make safety cases more manageable is safety case patterns as proposed by Kelly and Weaver [6]. These patterns can reduce the effort in building safety cases as well as the probability to make mistakes. We aim at identifying such reusable blocks especially for automotive safety cases.

RQ 2: What are reoccurring patterns in safety cases in the automotive domain?

Finally, as domain-specific models play a large role in automotive applications as well as the vast amount of information relevant for safety cases contained in these models, we analyse in more detail how we can employ them beneficially in building a safety case.

RQ 3: How can safety cases use existing domain-specific models in the automotive domain?

B. Case and subjects selection

We select one case and several subjects opportunistically. As subjects, all of the authors contribute to the case study in different roles, which makes it partially action research. Stefan Puchner takes the role of the safety case developer, Peter Kock is the expert for the domain and especially the selected case, Bernard Schätz acts as automotive domain and modelling expert as well as method expert for safety cases, Stefan Wagner works in the role of a safety case method and quality assurance expert. Following from this role distribution, we choose a case in which our domain expert has the most expertise and a current interest in its safety assurance. The case should also be representative for an automotive system.

C. Data collection procedure

The study uses a 3-step process for building a safety case for a real automotive component. Specifically, we investigate the structure and application of models to guide the construction of safety cases. The following describes the top-down process for the argumentation about the safety requirements in the case study:

- 1) Identify hazards
- 2) Elicit system requirements to avoid hazards
- 3) Break down requirements and system until actual realisation component is reached by using one of the following argumentations:
 - a) Provide evidence that realisation component fulfils the imposed requirements
 - b) Split component into sub-components, define requirements for those and continue with step 3
 - c) Split requirements into sub-requirements, using appropriate safety case pattern and continue with step 3

Step 3 distinguishes between argumentation cases of elementary requirements for basic components, decomposition

of complex components, and decomposition of complex requirements.

D. Analysis procedure

The safety case developer builds the safety case using the process described in the data collection procedure. All subjects then analyse it qualitatively to find safety argument modules and patterns. This analysis is mostly driven by domain-specific models in the automotive domain as we assume that these capture the important domain knowledge needed for the structuring.

Hence, we first assign different parts of the safety case to different existing models such as the environment model, the user model, or software function models. These assignments provide then the basis for grouping them into suitable argument modules.

Second, all subjects look for reoccurring parts that we could refactor into safety case patterns. We extract them, describe them as patterns, and integrate them into all places where they fit. We then re-check whether the pattern is appropriate at all used locations in the safety case.

Third, the subjects identify all uses of domain-specific models in the safety case. We abstract and combine them to general uses of such models in building a safety case in the automotive domain.

E. Validity procedure

To ensure internal validity, we make extensive use of reviews by the different roles. All modules, patterns, and uses are reviewed by all roles, i.e., domain and method experts. The experts give feedback and improvement suggestions to ensure that it is theoretically correct and practically applicable. Furthermore, we discussed the results with further experts inside and outside the company that provided the case.

External validity can be ensured only to a limited degree as it is a single case from a single company. Nevertheless, we took care to choose an automotive component for the safety case that is part of or in development for most automotive systems.

V. RESULTS

First, we give detailed information on the case that we selected. Then we describe the results for each research question: safety case modules, safety case patterns, and usage in connection with models.

A. Case description

We use a case study from MAN Nutzfahrzeuge AG, a German-based international supplier of commercial vehicles and transport systems, mainly trucks and buses. As the engineers at MAN routinely use models in their automotive software development [15], MAN is an especially interesting case source in our study.

In the case study, we constructed a safety case for the cruise control component. A cruise control is supposed to keep the car at a user defined speed without constantly regulating the speed via the gas pedal. Regarding safety functionality, the driver must be able to switch off this feature at any time. Different shut-off paths have to be available, e.g., brake pedal activation, clutch activation, off switch, or park brake switch.

B. Safety Case Construction

For the cruise control, the safety case construction is instantiated as follows:

- 1) *Identify hazards:* The hazard we analysed was that the speed of the vehicle is higher than the speed set by the driver. It is the most relevant hazard for the cruise control, because excessive speed may have legal consequences as well as lead to harm of persons and equipment.
- 2) *Elicit system requirements to avoid hazards:* With the help of the conceptual model of the car and its environment, we elicit several requirements the car needs to fulfil in order to avoid the hazard. We concentrate in the following on the requirement that the car does not further increase speed if it reaches the target speed.
- 3) *Break down system requirements:* We consider functional models on the system, subsystem, and function level. Here, the system and subsystem models help to (1) differentiate between acceleration provided by the car and by the environment (e.g., by a declining road), (2) identify the abstract engine functionality as the single point for acceleration provided by the car, (3) identify sub-requirements split between the subsystems, and (4) split between mechanics, electric, and software functions of the cruise control and identify sub-requirements. Even on the level of the cruise control software function, corresponding to a single functional model in form of the actual realisation of the software function, this model is too complex to be covered by a single argumentation, requiring to divide it into smaller pieces by iteratively refining the safety case.

While this form of safety case construction already provides relevant feedback for research questions RQ 1 and RQ 2, the investigation of research question RQ 3 requires a more detailed application of models. One use of models is their application to *guide the construction of the safety case along the architecture of the system* when splitting components into sub-components. Another form of use is their application to *provide evidence for a realisation component*.

Typical for that case is the application of *fault models* of the hardware platform. In the case study, the correct functionality of the cruise control depends on the validity of

the input signals current speed and target speed. To provide a safety argument for the case that those signals are not correct, we employ a fault model in the safety case, using standard failure modes for the relevant signals. This results in several sub-requirements for different input signal failure modes, providing argumentation for safe activation of a fail-safe mode. The builders of these fault models constructed them also in a pattern-based fashion by extending functional models with typical faults (e.g., early, late, lost, or false signals).

While fault models are not explicitly used in all automotive projects, *functional (software) models* are commonplace in a model-based development process. They establish a specific property of a component on the implementation level, e.g., in form of a formal proof. In the case study, we applied the verification tool EmbeddedValidator⁴ to a sub-system of the cruise control TargetLink model, which calculates the torque requirement based on the current speed and the target speed desired by the driver. The property proved that it does not reach an unsafe state where the actual speed is higher than the required speed and the torque requirement towards the engine is not zero. In the proof of this property, we include assumptions established in the process of constructing the safety case; here, specifically, we apply such assumptions in the form of restricted inputs ensuring an activated cruise control mode. For this type of argumentation, the safety case provides the requirement to be checked and the relevant TargetLink (sub-) model.

The modular construction of models can immediately be exploited for a modular construction of safety cases. To demonstrate modular safety cases, the case study includes the scenario of a supplied sensor with a corresponding supplied safety case. For the sensor, we constructed a safety case corresponding to its structure of a custom electronic circuit and a software function that pre-processes the raw input for actual use in further software components (Figure 2), which is based on fault model patterns for the hardware components.

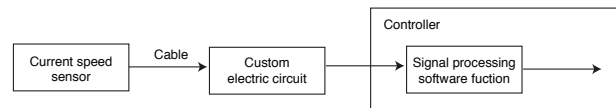


Figure 2. Schematic illustration of signal generation

C. Safety cases modules

Software-intensive safety-critical systems, like automotive vehicles, use a combination of mechanic, electric/electronic, and software components to implement their overall functionality. Consequently, as demanded, e.g., in IEC 61508 [2]

⁴Model checker for Simulink/Stateflow and TargetLink (<http://www.btc-es.de/>)

“all the safety-related systems making up the total combination of safety-related systems” from all these domains have to be considered in a safety case to support a suitable safety argumentation. To structure these different components in the safety case, we built a safety case architecture consisting of *safety case modules* [8].

Certification standards like [2] take both the product under development and the development process into account. Therefore, as shown in Figure 3, a safety case should cover both areas. In the case study, however, we concentrated on the product part. For the cruise control, we found it useful to structure the product-related safety case into arguments about the system itself in different abstraction levels and arguments about the environment and the user of the system. We discuss the different modules in more detail in the following:

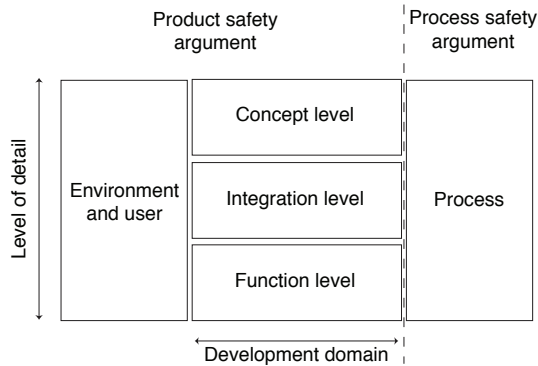


Figure 3. Safety case modules

Environment and user: We can only argue about a system’s safety, if we take people and systems in its environment into account. This includes the driver of the car as well as passengers and maintenance personnel. This module contains assumptions about the users and their behaviour. For the cruise control, the reaction time of the driver is important, for example. Furthermore, the module describes environmental situations, like inclining and declining roads, wet surface, or wind from different directions, which all have an influence on the acceleration and deceleration of the car. As we used partly existing user and environment models, the content of this module is likely to be reusable for other car functions.

Concept level: On the highest level of abstraction, we used the *concept* module, which contains the argumentation about the logical functions of the car in combination with its users and environment. Here, we have not yet distinguished between mechanical parts, wires, or software. We argue on the conceptual level how the car should react to certain user inputs or environmental situations. For the cruise control, a relevant environmental situation that does not already need more information about the implementation, is that the driver

reduces the target speed. We made the experience that this conceptual considerations help in identifying hazards as we were not distracted by technical details. For example, we used the user expectations on the cruise control function as source for potential hazard. If the driver reduces the target speed, the expectation is that the car does not decelerate instantly, but slowly. An abrupt deceleration could be a hazard as the driver would be surprised as well as vehicles behind our car might rear-end it.

Integration level: On this level, we need to decide how to partition the logical functions into mechanics and electronics. If we had a safety case before the implementation of a car function, we could make this decision also based on safety argumentations. For the cruise control, we took the existing partitioning. We split the safety requirements from the concept module into sub-requirements regarding the more specific design of the car. While the models on this level already specify high-level components, most rather specify functionality than its realisation. For the cruise control, we distinguish between the *cruise control software function* and the *cruise control controls*, which are the buttons and signal cables that the driver uses to control the cruise control. This enabled us to split the responsibility for the safety goal that the cruise control does not request acceleration if the current speed is greater or equal the target speed and the driver does not change it. The cruise control controls have to make sure that they do not change the target speed if the driver is inactive, the cruise control software function ensures the correct comparison of target and current speed.

Function level: On the function level, we separated mechanics, software, and electric circuits. Furthermore, for software we found that it is useful to analyse its deployment separately. These different realisations have fundamentally different properties and therefore different ways of arguing their safety. Finally, as supplied parts are possible on all levels, we considered them as separate. For the cruise control, we included a requirement for the supplied speed sensor.

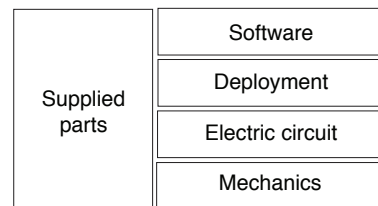


Figure 4. Safety argument modules in the function level module

- 1) *Mechanics:* This safety case module would contain argumentations like the location of a mechanical part or a boundary of temperature. We could check these requirements using CAD models and corresponding

evaluation methods. We did not further analyse mechanics for the cruise control, however.

- 2) *Electric circuit*: We used the electric circuit module to argue about the correctness and failure resistance of electric circuits. There could be dependencies to the mechanical design module, for instance, if it comes to temperature or vibration exposure of the electric circuit parts. We analysed the validity of signals and their transmission from speed sensors to the cruise control function.
- 3) *Deployment*: The deployment module acts as the mediator between the virtual software world and the physical world. Here we argue about the influence of failures in the execution platform (controllers, middleware, or operating system). We did not analyse these aspects in detail for the cruise control.
- 4) *Software*: Because of the deployment module, we can argue about the correctness of software in the software module, while we abstract from the real runtime environment. It contains the functions executed by controllers as well as those executed by FPGAs, since both are designed in software. For the cruise control, we had Simulink/TargetLink models for the analysed software parts. These enabled us to check for certain properties either by inspection or by formal proof using EmbeddedValidator. For example, for the model of the cruise control we proved that a change in acceleration does not exceed the given limits.
- 5) *Supplied parts*: This module covers parts that the manufacturer of the vehicle does not develop and is therefore not responsible for safety issues caused by those parts. It is necessary, however, to integrate those parts with their requirements and guarantees into the safety case for a comprehensive argumentation. Such requirements could contain temperature, physical stress, or maximum time of operation before the part has to be exchanged. Black-box safety case modules are a good way to exchange the information necessary for safety analysis in a ready-to-use form, while at the same time sensitive information is concealed from the vehicle manufacturer.

D. Safety case patterns

Besides using an overall module structure, we employed safety case patterns as basic building blocks. In this section, we give an overview of the patterns and how often we utilised them. Finally, we present brief examples from the patterns identified in our case study to illustrate how we solved common questions in creating the safety case.

The *strategy* node is the core element of the actual argumentation in a GSN safety case, supporting the justification of goals by subgoals. By applying strategy patterns, we built argumentations using only accepted justifications. This way confidence in the correctness of the argument can be

increased. Still the appropriateness and correct use of a pattern has to be evaluated before trusting a safety case, but the question of the fundamental validity of each single argumentation step itself need not be argued. On the long run, it is desirable to establish general and domain-specific patterns as a pattern library for a faster and easier creation of safety arguments. This case study is a first step in this direction.

Overall we identified 13 potential patterns stated in Table I. As we used several of them only once or twice, we cannot be sure that they qualify as patterns, but they seem generic enough to be used for other systems. We employed five of the patterns three or four times, which makes us confident that they are useful patterns. We describe the three most used patterns in the following.

Table I
THE EXTRACTED PATTERNS AND HOW OFTEN THEY WERE USED

Pattern	No. Usages
Use fault model	4
Logical transformation	4
Split by architecture	4
Formal elicitation on model	3
Split by items	3
Identification pattern	2
Split by category pattern	1
Property probability calculation	1
Detect & contain fault	1
Failed expectations	1
Fail-safe	1
Redundant signals	1
Independence of sensor signals	1

Pattern: Use fault model: For many mechanics and electric/electronic parts of the system, there are standard fault models that describe in which ways the part can fail. For the cruise control, we used a fault model for the signal cables that transport the information about the current speed. Hence, using context-specific fault patterns (e.g., for sensors, cables, etc.) we differentiated situations such as disruptions of the signal or overload on the cable and their consequences. The application in the safety case in GSN is shown in Figure 5. We utilised it also when considering the speed sensor failures or the common failure modes of a cruise control.

Pattern: Logical transformation: The goals are often logical combinations of situations that we want or do not want to occur. By combining different situations, these combinations can get long and overcomplicated. In that case, we checked whether we can apply the logical transformation pattern, which resolves the logical combinations to make the goals simpler if that is possible by using rules of predicate calculus. The part of the safety case in Figure 6 applies the pattern to the goal that demands to avoid a situation in which the constraint is violated to keep acceleration larger or equal than a lower bound. We simplified this by transforming it to

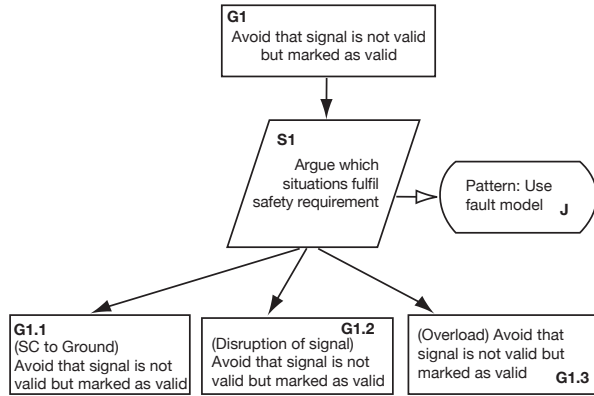


Figure 5. An example usage of the pattern: Use fault model

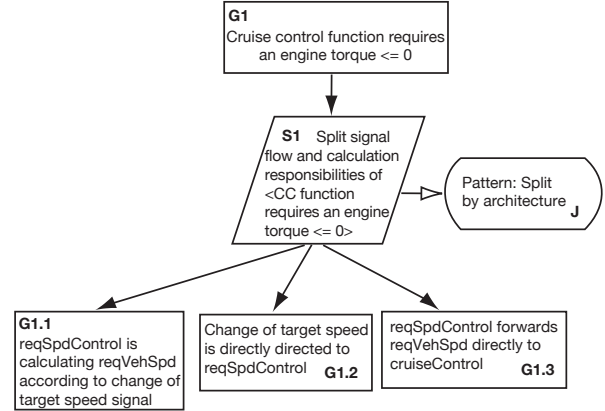


Figure 7. An example usage of the pattern: Split by architecture

the goal to avoid an acceleration which is smaller than the lower bound.

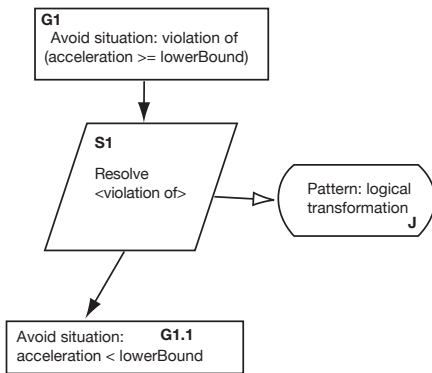


Figure 6. An example usage of the pattern: Logical transformation

Pattern: Split by architecture: If there is an architecture prescribed by existing specifications or models, we used these constraints to detail the safety case. We mostly used this pattern for the software part, which was modelled with Matlab Simulink/TargetLink. The hierarchical decomposition of the software functions were useful to refine goals into sub-goals. The (partial) application of this pattern in Figure 7 analyses the decomposition of the cruise control function and how it requires a specific engine torque. In this case, we split it into the two parts *reqSpdControl*, which is responsible for determining the target speed and *cruiseControl*, which is responsible for controlling the current speed. In addition to the responsibilities of the two components, their interaction needs to be considered. We used this pattern on all levels of the software architecture.

Despite that the patterns seem to cover only simple cases, they proved useful in providing us with a common and systematic way to build and structure the safety case.

E. Usage of safety cases with models

Using the automotive safety case architecture introduced in section V-C, we illustrate how we linked the safety case with the models used in the model-based development process at MAN. By using contexts in the safety case to link it to these models, we have a sound basis for the argumentation in the safety case by making assumptions explicit and by justifying them. To that end, we structure the safety case according to the structure of the system under development. Furthermore, we use the development models as information and requirements source for the safety case as well as a sink for assumptions posed by the safety case.

1) *Accordance of model and safety case structure:* The product-based safety case built was requirements-driven in the respect that the goal of building up the safety case is ultimately to fulfil safety requirements. Since the safety case is product-based, we assigned those requirements to a specific element of the system under development by providing a link between the artefact in the development model and the corresponding context element in the safety case. To provide the same structure in the safety case argument and in the system model, as a result the safety case splits the safety requirements according to the sub-components in the design, requiring reason that those sub-components are connected properly. Thus, to support this form of structuring, we employed the pattern *split by architecture*. In the case study this pattern is used extensively on all different abstraction levels, especially based on the Matlab Simulink/TargetLink models.

2) *Linking safety cases and models:* Besides guiding the argumentation structure of a model-based safety case, we utilised development models to ground the argumentation. As introduced in section II-B in a model-based development process, there are models of the environment, the system under consideration, and of the platform. Models of the environment and the platform *describe* reality by explicating,

e.g., assumptions about physical laws of the environment or available computational resources, while models of the system under consideration *prescribe* by specifying, e.g., the functionality of a software component or the task and bus schedules. While in a qualified model-based development process prescriptive models directly correspond to the implementation, descriptive models capture assumptions and need additional validation, e.g., by use of defect pattern libraries, as illustrated in Figure 1.

In case of the establishment of a functional safety requirement, e.g., the pattern consists of the corresponding goal to be proven, the solution to be applied, and the context defining the (part of) the functional model to be used in the solution. An example for the application of this pattern is a solution in form of the formal proof of the goal via model checking, as mentioned in Subsection V-B. Other solution techniques are, e.g., simulation or inspection.

More complex argumentations involve different models or different parts of a model. Figure 8 shows a requirement posed by a controller component in the deployment model. It does require the safety case to provide an argument that sufficient calculation capacity is available for the software modules deployed on this controller. Here the deployment model acts as the source context for the capacity demands.

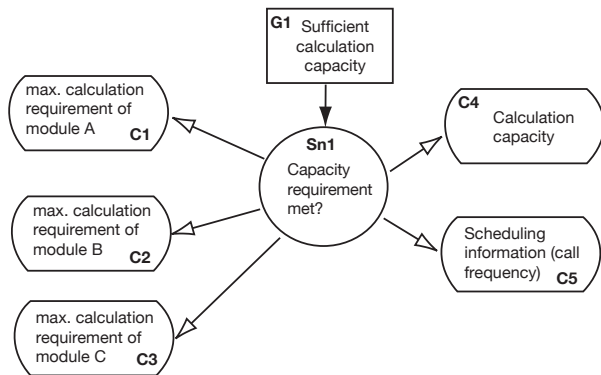


Figure 8. Platform requirement & provided properties

The solution for this kind of requirement uses properties that are contained in the deployment and software model:

- In the deployment model, the applied solution references the available calculation capacity as defined by the specification of the control unit.
- In the software model, the applied solution references the maximum calculation cost of the corresponding software modules deployed to this controller.
- In the deployment model, the applied solution references the scheduling frequency of the software modules.

Based on this information, the solution can establish that the requirement is fulfilled.

VI. CONCLUSIONS AND FUTURE WORK

First, we summarise the conclusions from the case study, set them into relation with existing evidence, discuss possible implications and limitations, and describe directions for future research.

A. Summary of conclusions

Combined hardware and software components become more and more important in safety-critical parts of automotive systems. In their development, models play an important role and contain relevant information to build a structured argumentation about their safety. In this case study, we followed a model-based approach for building a safety case for a real function in the automotive domain.

The first main contribution is a high-level safety case architecture that can be reused, as well as fine-grained patterns that describe reoccurring situations that can be handled in a similar way. The second contribution is a description of how the safety case is built and used.

The results have been developed and validated in an exploratory case study at the MAN Nutzfahrzeuge AG with the example system of a cruise control in a truck. It helped to create the overall structure and to identify patterns. Furthermore, it showed the practical applicability of the approach.

B. Relation to existing evidence

We have not made extensive use of safety case *modules* [8] as we followed mostly the decomposition of the system itself. Nevertheless, we found these explicit modules to be a useful concept for separation of concerns and to simplify the comprehension of the safety case.

Furthermore, we added several new safety case *patterns* to the collection of patterns available that are partly specific for the automotive environment, but several of the patterns could be useful in other contexts as well. We can support Kelly and McDermid [9] that there is a reuse potential in safety cases and that safety case patterns are a suitable medium for that.

Our findings show in accordance with [10]–[13] that safety cases can be usefully applied in the automotive domain. Similarly, as in [12], [13], we found the usage of models in a safety case beneficial. As an addition to Habli et al. [13], we investigated in particular Simulink/Stateflow/TargetLink models that are used to generate the final production code. Here the incorporation of these models proved especially valuable.

C. Impact/implications

First, the case study is an additional piece of evidence that safety cases can be usefully applied in the automotive domain. In combination with the existing evidence, we advise automotive companies to incorporate them in their normal development processes for safety-critical functions.

Second, the usage of domain-specific models seems to be especially helpful in the development of the safety case. Hence, the combination of models and safety cases should be further investigated.

D. Limitations

As it is often the case in case study research, the external validity cannot be assured, because we investigated only a single safety case development for a single automotive function at a single company. We used, however, a function that is likely to be part of a large share of new cars and trucks to make these results transferable. Furthermore, to some degree, the domain-specific models tend to be similar over different companies.

E. Future work

At present, we conduct a further case study with the BMW AG and a different automotive function to analyse the applicability of our safety case architecture and the found patterns. In addition, we consider most of the patterns we have found not to be specific for the automotive domain and therefore, we would like to test them in other domains as well.

Moreover, we plan to integrate our experiences with product quality models [16], [17] as source for safety case structuring, argumentation as well as quantitative evaluation results.

ACKNOWLEDGEMENTS

This work has partially been supported by the German Federal Ministry of Education and Research (BMBF) in the project Quamoco (01 IS 08023B).

REFERENCES

- [1] A. Pretschner, C. Salzmann, B. Schätz, and T. Stauner, “4th international ICSE workshop on software engineering for automotive systems,” in *ICSE Companion*. IEEE Computer Society, 2007, p. 146.
- [2] *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems*, International Electrotechnical Commission, 1998.
- [3] B. Kitchenham and S. L. Pfleeger, “Software quality: The elusive target,” *IEEE Software*, vol. 13, no. 1, pp. 12–21, 1996.
- [4] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.
- [5] P. Bishop and R. Bloomfield, “A methodology for safety case development,” in *Proc. Sixth Safety-Critical Systems Symposium (SSS’98)*. Springer-Verlag, 1998.
- [6] T. Kelly and R. Weaver, “The goal structuring notation – a safety argument notation,” in *Proc. DSN 2004 Workshop on Assurance Cases*, 2004.
- [7] *Using Simulink and Stateflow in Automotive Applications*, The Mathworks, 1998.
- [8] T. Kelly, “Managing complex safety cases,” in *Proc. 11th Safety Critical System Symposium (SSS’03)*. Springer-Verlag, 2003, pp. 99–115.
- [9] T. P. Kelly and J. A. McDermid, “Safety case construction and reuse using patterns,” in *Proc. 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP’97)*. Springer-Verlag, 1997.
- [10] W. Ridderhof, H.-G. Groß, and H. Dörr, “Establishing evidence for safety cases in automotive systems - a case study,” in *Proc. 26th International Conference on Computer Safety, Reliability, and Security (SAFECOMP’07)*, ser. LNCS, vol. 4680. Springer-Verlag, 2007.
- [11] F. Törner and P. Öhman, “Automotive safety cases – a qualitative case study of drivers, usages, and issues,” in *Proc. 11th IEEE High Assurance Systems Engineering Symposium*. IEEE Computer Society, 2008, pp. 313–322.
- [12] D. Chen, R. Johansson, H. Lönn, Y. Papadopoulos, A. Sandberg, F. Törner, and M. Törngren, “Modelling support for design of safety-critical automotive embedded systems,” in *Proc. 27th International Conference on Computer Safety, Reliability, and Security (SAFECOMP’08)*, ser. LNCS, vol. 5219. Springer-Verlag, 2008, pp. 72–85.
- [13] I. Habli, I. Ibarra, R. Rivett, and T. Kelly, “Model-based assurance for justifying automotive functional safety,” in *Proc. 2010 SAE World Congress*, 2010.
- [14] P. Braun, J. Philipps, B. Schätz, and S. Wagner, “Model-based safety cases for software-intensive systems,” *Electronic Notes in Theoretical Computer Science*, vol. 238, no. 4, pp. 71–77, 2009.
- [15] F. Deissenboeck, B. Hummel, E. Juergens, B. Schaetz, S. Wagner, J.-F. Girard, and S. Teuchert, “Clone detection in automotive model-based development,” in *Proc. 30th International Conference on Software Engineering (ICSE ’08)*. ACM Press, 2008, pp. 603–612.
- [16] F. Deißböck, S. Wagner, M. Pizka, S. Teuchert, and J.-F. Girard, “An activity-based quality model for maintainability,” in *Proc. 23rd International Conference on Software Maintenance (ICSM ’07)*. IEEE Computer Society Press, 2007, pp. 184–193.
- [17] S. Wagner, “A Bayesian network approach to assess and predict software quality using activity-based quality models,” *Information and Software Technology*, vol. 52, no. 11, pp. 1230–1241, 2010.