Requirements Engineering for Automotive Embedded Systems

Miroslaw Staron

Abstract Requirements engineering is both a phase of software development lifecycle and a subdomain of software engineering. In general, “requirements" is defined as the description of the functionality of software under design and its properties (functional and nonfunctional requirements). Requirements are often perceived as textual documentation. However, in automotive software engineering, requirements can have multiple forms—starting from the short textual descriptions of functionality to fully executable model-based specifications.

In this chapter, we overview the notion of a requirement in general, and describe the types of requirements used when designing automotive software systems. We use the V-model, prescribed by the ISO 26262 safety standard, which describes the way in which software is designed in the automotive domain. We consider the different types of requirements used in these phases.

1 Introduction

Contemporary cars, trucks, buses, and even bikes have software—some as much as 1 GB of onboard binary code excluding maps, music, and other downloadable data. As the history of software dates back to the 1970s with the first onboard Electronic Control Units (ECUs) in an engine, we could observe an enormous growth of software. Up until the end of the 1990s, the amount of onboard code was measured in megabytes, and only a few ECUs were present in the car. However, in the last decade, this amount has grown to over 130 ECUs per car and as much as the aforementioned 1 GB of code.

Moreover, software is included in more safety-critical areas, such as collision avoidance by breaking, automatic parking, or autonomous driving. Therefore, we need to enhance our expertise in working with software as one of the primary

M. Staron (✉)
Computer Science and Engineering, University of Gothenburg, Gothenburg, Sweden
e-mail: Miroslaw.Staron@cse.gu.se

Y. Dajsuren, M. van den Brand (eds.), *Automotive Systems and Software Engineering*, https://doi.org/10.1007/978-3-030-12157-0_2

development entities alongside mechanics and electronics. In this chapter, we focus on one of these areas—requirements engineering. We contribute by providing practical examples of how to efficiently utilize requirements engineering for automotive systems.

The area of requirements engineering is one of the disciplines in vehicle development on the one hand, and, on the other hand, it is a subdomain of software engineering and one of the initial phases of software development lifecycle. It deals with the methods, tools, and techniques for eliciting, specifying, documenting, prioritizing, and quality assuring the requirements. The requirements themselves are very important in enhancing the quality of software in various ways as quality is defined as *"The degree to which software fulfills the user requirements, implicit expectations and professional standards."* [16].

Requirements is often defined as *(1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents; and (3) a documented representation of a condition or capability as in (1) or (2)* [16]. This definition stresses the link between the user of the system and the system itself, which is important for a number of reasons:

- Testability of the system—it should be clear how a requirement should be tested, for example, what is the usage scenario realized by the requirement.
- Traceability of the functionality to design—it should be possible to trace which parts of the software construction realize the requirement in order to provide safety argumentation and enable impact/change management.
- Traceability of the project progress—it should be possible to get an overview of which requirements have already been implemented and which are still to be implemented in the project.

It is a very technical definition for something that is intuitively well known—a requirement is a way of communicating what we, the users, want in our dream car. In this sense, it seems that the discipline of requirements engineering is simple. In practice, working with requirements is very complex as the ideas that we, users, have need to be translated to one of the millions of components of the car and its software. So, let us see how the automotive companies work with our requirements or dreams.

We discuss software requirements engineering because the automotive industry has recognized the need to shift its innovation from the mechanical parts of the car to its electronics and software. The majority of us, the customers, buy cars today because they are fast (sporty), safe, or comfortable. In many cases, these properties are realized by adjusting the software that steers parts of the modern cars. For example, we can have the same car with a software package that makes it extremely sporty—look at the Tesla's "Insane" acceleration package or the Volvo's Polestar performance package. These are just the two challenges which lead to two very important trends in automotive software requirements engineering:

1. Growing amount of software in contemporary cars—as the innovation is driven by software, the amount of software and its complexity grows exponentially. For example, the amount of software in the 1990s was a few megabytes of binary code (e.g., Volvo S80) and today reaches over one gigabyte excluding maps and other user data (e.g., Volvo XC90 of 2016).
2. Safety requirements posed by standards such as ISO 26262—as the software steers more parts of the car, there is a larger probability that it can interfere with our driving and cause accidents, and therefore, it has to be safety-assured just like the software in airplanes or trains. The contemporary standard for functional safety (ISO/IEC 26262, Road Vehicles—Functional Safety) prescribes methods and processes to specify, design, and verify/validate the software.

Automotive software requirements engineering needs rigid processes for handling the construction of the software in the car and therefore is much different from the other types of software requirements engineering, such as telecom or web design.

This chapter explores the theory of requirements engineering in automotive development by examining two types of requirements—textual specifications and models used as requirements. It also helps us to explore the evolution of requirements engineering in automotive software development to finally draw on the current trends and challenges for the future.

2 Requirements and Requirements Engineering

Requirements engineering in the automotive sector is increasingly about the software, since software is the source of innovations in the dramatically increasing tempo of changes. According to Houdek [15] and the report on the innovation in the car industry [8], the number of functions in an average car grows much faster than the number of devices, with the number of systematic innovations growing faster than the individual innovations. Systematic innovations are systems of software functions rather than individual functions. Therefore, the discipline of requirements engineering is more about engineering than it is about innovation.

The volume of an automotive requirement specification is in the range of 100000 pages for a new car model according to Houdek based on his study at Mercedes-Benz [15], with ca. 400 documents of 250 pages each at the lowest specification level (component specifications), which are sent over to a large number of suppliers (usually over 100 suppliers, one for each ECU in the car).

Weber and Weisbrod [42] expounded the complexity and size of requirement specifications in the automotive domain based on their experiences from DaimlerChrysler. Their large software development projects can have as many as 160 engineers working on a single requirement specification and producing over 3 GB of requirements data. Weber and Weisbrod describe the process of requirements engineering in the following way: “Textual requirements are only part of the

game—automotive development is too complex for text alone to manage." This quote reflects the -state-of-the-art practice of requirements engineering—that the requirements form only one part of the construction database. However, let us look at how the requirements are specified in the automotive domain. Similar challenges of linking requirements to other parts of the construction database can be also found in our previous studies in [23].

3 Types of Requirements in Automotive Software Development

When designing software for a car, designers (who are often referred to as constructors) gradually break down the requirements from the car level to the component level. They also gradually refine them from textual requirements to models of behavior of the software. This gradual refinement is caused by the fact that the requirements have to be sent to Tier 1 suppliers for development and therefore should be as detailed as possible to enable their validation. Figure 1 presents the main phases of software development for automotive systems, roughly based on the software development process model prescribed by ISO/IEC 26262 Systems and Software Safety—Functional Safety standard [17].

In the figure, we also make a distinction between the responsibilities of original equipment manufacturers (OEMs) (vehicle manufactures) and their suppliers. This distinction is important as it is often the phase where the handshaking between the suppliers and the OEMs takes place, and therefore the requirements are used during the contract negotiations. In this context, a detailed, unambiguous, and correct requirement specification prevents potentially unnecessary costs related to

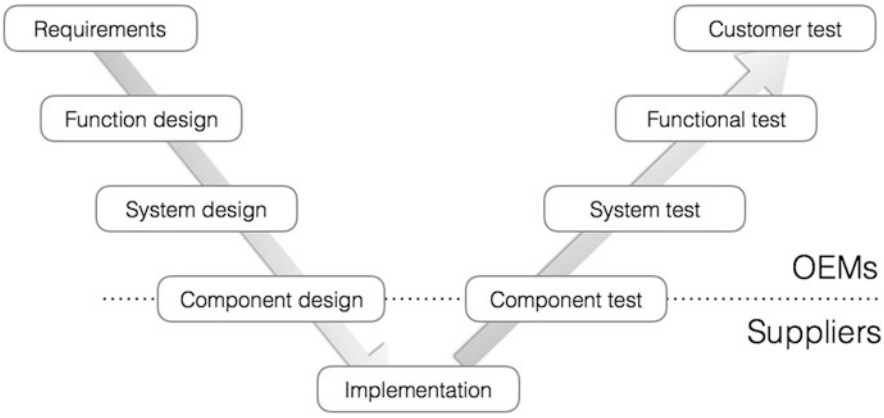

Fig. 1 V-shaped model of software development process in automotive software development

the changes in requirements caused by miscommunication between the OEMs and suppliers.

In the automotive domain, we have a number of tiers of suppliers:

- Tier 1—suppliers working directly with OEMs, usually delivering complete software and hardware subsystems and ECUs to the OEMs.
- Tier 2—suppliers working with Tier 1 suppliers, delivering parts of the sub-products that are then delivered by Tier 1 suppliers to the OEMs; Tier 2 suppliers usually do not work directly with OEMs, which makes it even more important for the requirements to be detailed so that they can be correctly broken down by Tier 1 suppliers for Tier 2.
- Tier 3—suppliers working with Tier 2 suppliers, similar to Tier 2 suppliers working with Tier 1 suppliers.

In this section, we describe these different types of requirements, which can be found in these phases.

3.1 Textual Requirements

AUTOSAR is a great source of inspiration for research in automotive software development, and therefore let us look at the requirements in this standard—it appears that they are mostly textual. An example of a requirement specified in this format, for a feature of keyless entry, is presented in Fig. 2.

The structure of the requirement is quite typical for requirements in general—it contains the description, rationale, and use cases. So far, we do not see anything specific. Nevertheless, if we look at the sheer size of such a specification—over

REQ-1: Keyless vehicle entry

Type	Valid
Description	It should be able to open the car with an RFID key or a mobile phone
Rationale	The majority of our competitors have a RFID sensors in the car that open and start the car based on the proximity of the designated driver who has the RFID sender (e.g. a card). To stay ahead of the competition, we need to provide the key as a mobile phone app for iOS and Android phones.
Use case	Keyless start-up
Dependencies	REQ-11: RFID implementation
Supporting material	---

Fig. 2 An example of a textual requirement, specified in a format used by AUTOSAR requirements

1000 pages—we can see that we might be at loggerheads, so let us discuss the kind of issues we can discover.

Why: The textual requirements are used when describing high-level properties of cars. These types of requirements are mostly used in two phases—the requirements phase when the specification of the car's functionality at a high level takes place and at the component design phase where large software requirement specification documents are sent to suppliers for development (although the textual requirements are often complemented by model-based requirements).

How: Specifying this kind of requirements rarely happens from scratch. Textual requirements are often specified based on models (e.g., UML domain models) and are intended to describe details of the innerworking of software systems. They are often linked to verification methods describing how the requirement should be verified—for example, describing the test procedure for validation that the requirement is implemented correctly. Quite often, it is the suppliers who do the verification as many requirements demand specific test equipment to test their implementation. If this is the case, the OEMs choose a subset of requirements and verify them to check the correctness of the verification procedure from their side.

What: The text for the requirement is specified in the format given in Fig. 2—tables with text. This format is effective for specific details, but ineffective when we want to communicate overviews and provide the context for the requirements. For this, we need other types of formats—use cases or models.

3.2 Use Cases

In software engineering, the golden standard to specify requirements is to adopt the use cases as defined by Jacobson together with this objectory methodology in the 1990s [18]. The use cases describe a course of interaction between an actor and the system under specification, for example, as shown in Fig. 3 where the actor interacts with the car in the use case "Keyless start/up." The corresponding diagram (called the use case diagram in UML) is used to present which interactions (use cases) exist and how many actors are included in these interactions.

In the automotive industry, this kind of requirements specification is the most common when describing functions of vehicles and their dependency. It is used to describe how the actors (drivers or other cars) interact with the designed vehicle (the system) in order to realize a specific use case. This kind of specification is often described using the sequence diagrams of UML, and we can see an example of such a specification in Fig. 4.

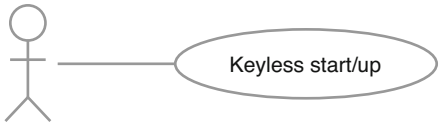

Fig. 3 An example use case specification with one use case

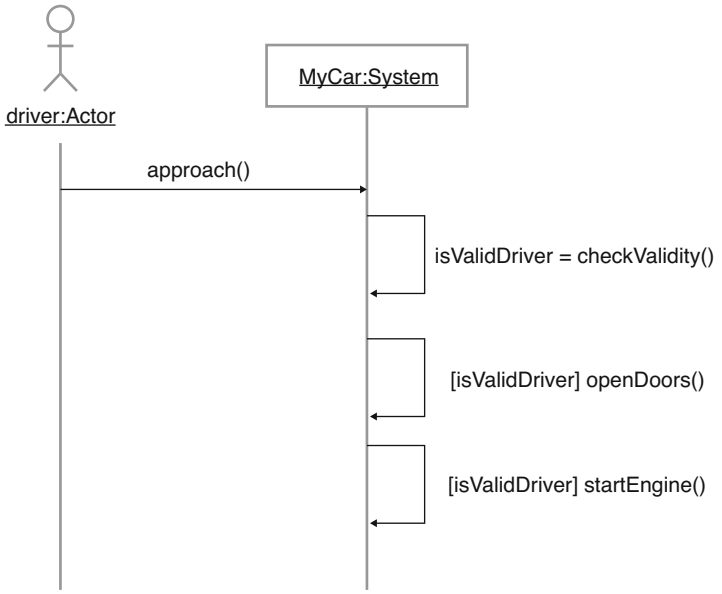

Fig. 4 An example specification of a use case utilizing the message sequence charts/sequence diagrams

Why: The use case specifications provide a high-level overview of the functionality of the designed system, such as a car, and therefore are very useful in the early phases of vehicle development. Usually, these early phases are functional design (use case diagrams) and the beginning of system design (use case specifications).

How: Using high-level descriptions of product properties, functional designers break down these properties into usage scenarios. These usage scenarios provide a possibility to identify which of the functions (use cases) are of value to the customers and which are too cumbersome.

What: These kinds of specifications consist of three parts—(1) the use case diagram, (2) the use case specification utilizing the sequence diagram, and (3) the textual specification of a use case detailing the steps of the interaction applying a somewhat structured natural language.

3.3 Model-Based Requirements

One method to provide more context to the requirements is to express them as models. This kind of representation can be done in two types of formalisms—UML-like models and Simulink models. In Fig. 5, we present an excerpt of a Simulink model for an ABS system from [32, 33, 37].

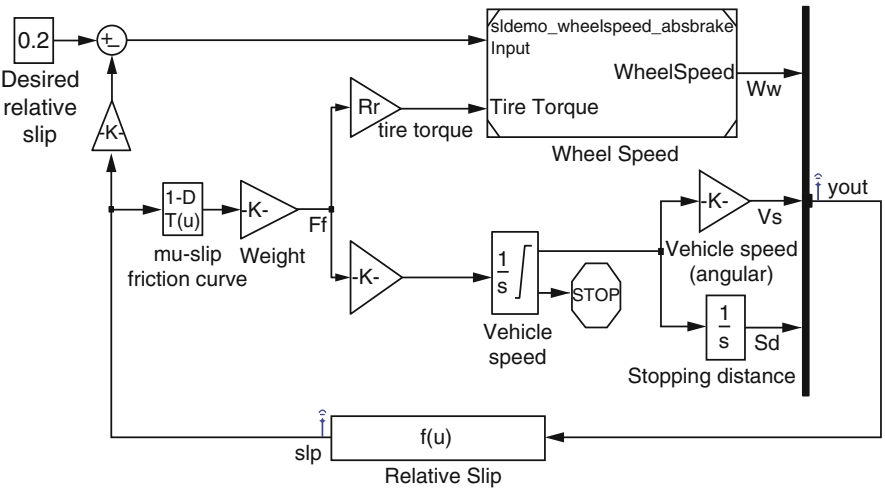

Fig. 5 An example Simulink model that can be used as a requirement to describe how to implement the ABS system

The model shows how to implement the ABS, but the most important property is that the model shows how the algorithm should behave and therefore how it should be verified.

Why: Using models as requirements has been recognized by practitioners, and in an automotive software project, up to 23% of the models are used as requirements according to our previous studies [26] and [25]. According to the same study, up to 13% of the effort is spent in the software project to design these kinds of requirements.

How: The simulation models used for requirements engineering are often used as part of the process of system design and function design where the software and system designers develop algorithms that describe how functions in modern cars are to be realized. These models can be automatically translated to C/C++ code using code generation, but it is rather uncommon. Hence, these models describe the entire functions that are often partitioned into different domains and spread over multiple components. Quite often, these kinds of requirements are translated into textual specifications as shown in the previous subsection.

What: The models are expressed using Simulink or a variation of statechart such as Statemate or Petri nets. These simulation models detail the functions described in the use cases by adding the system view of the interaction—the blocks and signals. The blocks and signals represent the realization of the functionality in a car and are focused on one function only. These models are often used as specifications, which are then detailed and often used to generate the source code automatically.

3.4 Requirements as Models

With the introduction of SysML, the models became more expressive than they were when modeled with UML. SysML introduced the notion of requirements diagram, as shown in Fig. 6.

Why: Considering the requirements as first-class entities in models provides the possibility to link them to construction elements of the design [39]. These links provide the possibility to trace requirements to implementation details and therefore speed up modifications.

How: The requirements and their rationale are modeled boxes and lines, just like any other modeling element in SysML. The requirements diagram is one of the most flexible diagrams in SysML, where we can place all kinds of structural elements.

What: The requirements capture the functions and properties of the products. They are linked to rationales and design intentions to increase awareness of the design and implementation constructs in the context.

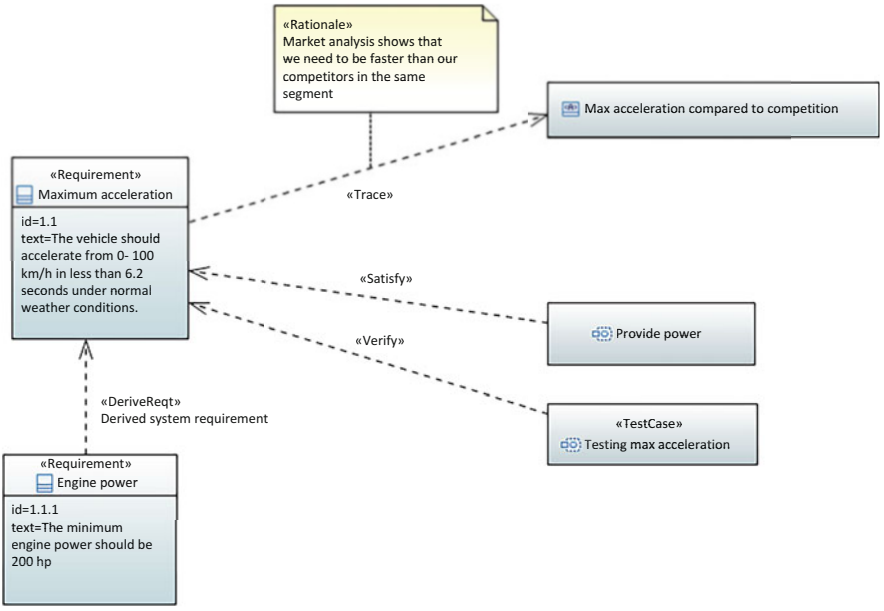

Fig. 6 An example SysML requirement model

4 Measuring Requirements and Requirement Specifications

Industry grade requirement specifications are significantly large—tens of thousands of requirements. Therefore, software engineers use quantitative assessments to understand the complexity and quality of software requirements.

Honig [14] provides a number of rudimentary measures for requirement specifications[1]:

- Requirement correctness—Is the individual requirement properly defining a genuine system function and need? In some cases, the measure may be determined by a formal system requirement verification process.
- Requirement unambiguity—Is the requirement clear and understandable to the expected users of the document? Are multiple, different interpretations of the requirement by different readers unlikely?
- Requirement completeness—Does this single atomic requirement include everything necessary to fully understand the desired function? Are all realizable types of input data, events, system environment covered? Are all terms used understandable or included in the glossary?
- Requirement verifiability—How adequately can this requirement be tested? Is it perfectly clear what test(s) are needed to confirm the requirement is met? Is it clear what should be considered a failure of a test of this requirement?
- Requirement modifiability—Is the individual requirement written so as to be easy to update, change, and eliminate in the future as system needs evolve?
- Requirement atomicity—Is the requirement all one, individual, atomic requirement, including limits, constraints, and all details of the functionality?
- Requirements completeness—Is the set of atomic requirements complete and providing a full definition of all necessary functionality for the entire system (or the current portion being reviewed)?
- Requirements consistency—Is the set of atomic requirements internally consistent, with no contradictions, no duplication between individual requirements?
- Requirements importance ranking—The set of atomic requirements are individually assigned to suitable importance categories (e.g., Essential, Desirable, Optional/Frill) and the assignment of values is appropriate.
- Requirements traceability—Are the individual atomic requirements uniquely identified with unchanging numbers? Are other existing documents or deliverables linked to individual requirements appropriately (e.g., use cases related to atomic requirements)?
- Requirements purity—Is the document free from system design and project schedule, staffing, etc.?
- Requirements count—Current number of individually identified and numbered atomic requirements.

[1]The definitions of the measures are quoted directly from the paper.

The abovementioned set of measures shows the major shortcoming of the requirement assessment practices—they are based on manual assessments. However, some studies show that requirements can be quantified automatically in a meaningful way. This quantification can be done based on the semantical analysis of the meaning of requirements (majority of the research), but it can also be approximated with the search-based techniques.

For example, Antinyan and Staron [2, 3] identified the following measures to be significant for assessing the complexity of requirements:

- Number of conjunctions
- Number of vague phrases
- Number of references
- Number of referenced documents
- Number of words

These measures can be combined into a requirement quality index. The index provides designers with the possibility to rank their requirements and improve their quality.

5 How All These Requirements Come Together

All these types of requirements need to come together somehow; hence, we have the process and the infrastructure for requirements engineering. Let us start with the infrastructure—usually named the design or construction database. In the light of the work of Weber and Weisbrod [42], it is called the common information model. Figure 7 presents the way in which this design database is used. The construction database contains all elements of the design of the electrical system of the vehicle—components, electronic control units, systems, controllers, etc. The structure of such a database is hierarchical and reflects the structure of the vehicle. Each of the elements in the database has a set of requirements linked to them. The requirements are also linked to one another to show how they are broken down. Such a database grows over time and is version controlled as different versions of the same elements can be used in different vehicles (e.g., different year models of the same car or different cars).

An example of such a system is described by Chen et al. [6] and has been developed by the company Systemite, which specializes in the databases for designs. Such a database structures all the elements of the construction of the integrated electronics of the vehicle and links all artifacts to the construction elements. An example of a construction element is the engine's electronic control unit, and all the functions that use this control unit are linked to it.

Such a database usually has a number of views that show the required set of details—functional view, architectural view, topological view, and software components' view. Each view provides the corresponding entry point and shows the relevant elements, but the database is always in a consistent state where all the links are valid.

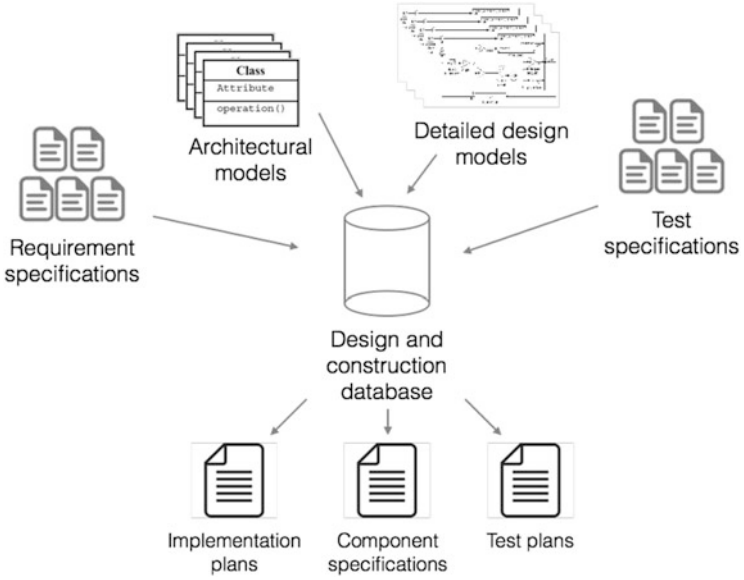

Fig. 7 Design database

The database is used to generate construction specifications for different actors. For each supplier who delivers an ECU, the database generates the set of all requirements that are linked to the ECU and all models that describe the behavior of the ECU. Sometimes, depending on the situation, the documentation contains even the simulation models for the functions that are to be included in the ECU.

6 Current Trends of Software Requirements Engineering in the Automotive Domain

Based on the observations of the evolution of the automotive embedded software, we could observe a number of trends in requirements engineering. In this section, we describe these trends.

Agility in Specification Development Agile software development has been used in many domains outside the automotive domain, and now there is evidence that it is used increasingly in the automotive domain. In particular at the lower part of the V-model, the suppliers work more agile with their requirements engineering and software development [22]. We can also observe these trends scaling up to the complete vehicle development [11] and [20]. With this increased adoption of Agile principles, we can foresee increased ability to specify requirements along software development, especially as the trends in automotive electronics are that we

use increasingly more commodity (or off-the-shelf) components. AUTOSAR also prescribes standardized approach to development, which eases the use of iterative development principles as the development of electronics/hardware is decoupled from the development of functions/software.

Increased Focus on Traceability Increased amount of software in cars and their increased presence in safety systems lead to stricter processes for keeping track of requirements for safety-critical systems. ISO 26262 (Road Vehicles—Functional Safety) is one example of this. In the automotive domain, this means that the increased complexity of software modules [34] leads to more fine-grained traceability management. One of the enablers of this increased traceability is the increased integration between tools and tool chaining [5] and [4].

Increased Focus on Non-functional Properties The increased use of software for active safety systems calls for the increased focus on non-functional properties of software. The increased traffic on communication buses within the car and the increased capacity of the communication buses call for more synchronization and verification. The safety analyses such as control path monitoring, safety bits, and data complexity control are just a few examples [38]. As the focus of requirements engineering research in the automotive domain was mainly (or implicitly) on the functional requirements, we foresee increased growth of research and emphasis on the non-functional requirements.

Increased Focus on Security Requirements A dedicated group of requirements is the security requirements. As our cars are increasingly connected, they are prone to hacker attacks [35] and [43]. The recent demonstration of the possibility to steer a Jeep Wrangler vehicle off-road showed that the threat is real and related to the safety of cars and transport systems. We therefore perceive that the ability to prevent attacks will be of focus for the automotive software development increasingly more in the coming decade.

7 Further Reading

This chapter provides an overview of the techniques used for requirements engineering in the automotive domain, and interested readers are encouraged to dive deeper into the topic. We provide a number of interesting entry points to more research in requirements engineering for automotive software systems.

Ott et al. [28] and [29] present a study on requirements engineering at Mercedes-Benz where they classified over 5800 requirement review protocols to their quality model. Their results showed that textual requirements (or natural language requirements as they are called in the publication) are prone to problems such as inconsistency, incompleteness, or ambiguity—with about 70% of the defects in requirements falling into these categories. In the light of this article, we can see the need for complementing the textual requirements with more context provided by use case models, user stories, and use cases.

Törner et al. [40] presented a similar study but of the requirements at Volvo Cars Group. In contrast to the study of Ott et al. [28], these authors studied the use case specifications and not the textual requirements. The results, however, are similar as the main types of defects are missing elements (correctness in Ott et al.'s model) and incorrect linguistics (ambiguity in Ott et al.'s model).

Eliasson et al. [12] described further experiences from Volvo Cars Group where they explored challenges with requirements engineering at large in a mechatronics development organization. Their findings showed that there is a lot of communication in parallel to the requirement specification. The stakeholders in the requirement specification frequently mentioned the need to have a good network in order to specify the requirements correctly. This indicates the challenges described previously in this chapter that the requirements need more context than it is usually provided in just the specification (especially the textual specification).

Mahally et al. [20] identified requirements to be the main barriers and enablers of moving toward Agile mechatronics organizations. Although today OEMs try to move toward fast development of mechatronics and reduce the cycle time by using Agile software development approaches, the challenges are that we do not know upfront whether a requirement needs the development of electronics or it is only a software requirement. According to Mahally et al., this kind of problem needs to be solved, and based on the prediction of Houdek [15], issues of this kind might be coming to an end as device development flattens out and most of the requirements will be software requirements. Similar challenges were presented by Pernstål et al. [31] who found that requirements engineering is one of the top improvement areas for the automotive OEMs. The ability to communicate via requirements was also an important part.

At Audi, Allmann et al. [1] presented the challenges in the requirements communication on the boundary between the OEMs and their suppliers. They have identified the needs for better communication and the challenges of communicating through textual representations. They recognized the needs for tighter partnerships as there is an inherent deficiency in communicating through requirements—transferring knowledge through an intermediate medium. Therefore, they recommend to integrate systems to minimize the knowledge loss via transfer of documents.

Siegl et al. [36] presented a method for formalizing requirement specifications using Time Usage Model and applied it successfully to a requirement specification from one of the German OEMs. The evaluation study showed an increased test coverage and increased quality of the requirement specification.

At BMW, Hardt et al. [13] demonstrated the use of formalized domain engineering models in order to reason about the dependencies between requirements in the presence of variants. Their approach provided a simplistic, yet powerful, formalism, and its strength was the industrial applicability.

A study of the functional architecture of a car project at BMW and the requirements linked to the functions by Vogelsanag and Fuhrmann [41] showed that 85% of the functions are dependent on one another and that these dependencies caused a significant amount of problems in software projects. This study shows

the complexity of the functional decomposition of the vehicle's design and the complexity of its description.

At Bosch, Langenfeld et al. [19] the longitudinal study of a 5-year project showed that 61% of the defects in requirements come from the incompleteness or incorrectness of the requirement specifications.

One of interesting trends in requirements engineering is the automatization of tasks of requirement engineers. One of such tasks is the discovery of non-functional requirements. This task is based on reading the specifications of functional requirements and identifying phrases that should be transformed into non-functional requirements. A study on the automation of this task has been conducted by Cleland-Huang et al. [7]. The study showed that the automated classification of requirements could be as good as 90%, but at this stage it cannot replace the manual classifiers.

7.1 Requirements Specification Languages

A model for requirements traceability [10] DARWIN4Req has been proposed to address the challenges related to the ability to follow the requirements' lifecycle. The model allows to link requirements expressed in different formalities (e.g., UML, SysML) and link them to one another. However, to the best of our knowledge, the model and the tool have not been adopted on a wider scale yet.

EAST-ADL [9] is an architecture specification language, which contains the elements to capture requirements and link them to the architectural design. The approach is similar to SysML but with the difference that there is no dedicated requirement specification diagram. EAST-ADL has been demonstrated to work in industry; however, it is not a standard for automotive OEMs yet. Mahmud [21] presented a language ReSA that complements the EAST-ADL modeling language with the possibility to analyze and validate requirements (e.g., basic consistency checks).

For the non-functional requirements in the domain of safety, Peraldi-Frati and Albinet [30] have proposed another extension of the EAST-ADL language that allows for increased traceability of requirements and their linking to the non-functional properties of the designed embedded software (e.g., safety).

Mellegård and Staron [24] and [27] conducted an empirical study on the impact of using hierarchical graphical requirement specification on the quality of change impact assessment. For the purpose, they designed a requirements' specification language based on the existing formalism—Requirements Abstraction Model. The results showed that the graphical overview of the dependencies between requirements introduces a significant improvement.

8 Conclusions

Correct, unambiguous, and consistent requirement specifications are the foundations of high-quality software, in general, and in automotive embedded systems, in particular. In this chapter, we introduced the most common types of requirements used in this domain and provided their main strengths.

Based on the current state of evolution of the automotive software, we could observe three trends in the requirements engineering for the automotive embedded systems—(1) agility in requirement specification, (2) increased focus on non-functional requirements, and (3) increased focus on security as a domain for requirements. Toward the end of this chapter, we also provided an overview of the requirements practices in some of the vehicle manufacturers (Mercedes-Benz, Audi, BMW, and Volvo) based on the published experiences from these companies. We have also pointed out a number of directions for further reading for the interested.

In our future work, we plan to make a review of the requirements engineering practices in the main automotive OEMs and identify their commonalities and differences.

References

1. Allmann C, Winkler L, Kölzow T, et al (2006) The requirements engineering gap in the oem-supplier relationship. J Univers Knowl Manag 1(2):103–111
2. Antinyan V, Staron M (2017) Proactive reviews of textual requirements. In: IEEE 24th international conference on Software Analysis, Evolution and Reengineering (SANER), 2017. IEEE, Piscataway, pp 541–545
3. Antinyan V, Staron M (2017) Rendex: a method for automated reviews of textual requirements. J Syst Softw 131:63–77
4. Armengaud E, Biehl M, Bourrouilh Q, Breunig M, Farfeleder S, Hein C, Oertel M, Wallner A, Zoier M (2012) Integrated tool chain for improving traceability during the development of automotive systems. In: Proceedings of the 2012 embedded real time software and systems conference
5. Biehl M, DeJiu C, Törngren M (2010) Integrating safety analysis into the model-based development toolchain of automotive embedded systems. In: ACM sigplan notices, vol 45. ACM, New York, pp 125–132
6. Chen D, Törngren M, Shi J, Gerard S, Lönn H, Servat D, Strömberg M, Årzen KE (2006) Model integration in the development of embedded control systems-a characterization of current research efforts. In: 2006 IEEE international conference on control applications, computer aided control system design. IEEE, Piscataway, pp 1187–1193
7. Cleland-Huang J, Settimi R, Zou X, Solc P (2007) Automated classification of non-functional requirements. Requir Eng 12(2):103–120
8. Dannenberg J, Burgard J (2015) Car innovation: a comprehensive study on innovation in the automotive industry. Oliver Wyman Automotive, New York
9. Debruyne V, Simonot-Lion F, Trinquet Y (2005) East-adlan architecture description language. In: Architecture description languages. Springer, New York, pp 181–195

10. Dubois H, Peraldi-Frati MA, Lakhal F (2010) A model for requirements traceability in a heterogeneous model-based design process: application to automotive embedded systems. In: 2010 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE, Piscataway, pp 233–242
11. Eliasson U, Heldal R, Lantz J, Berger C (2014) Agile model-driven engineering in mechatronic systems-an industrial case study. In: Model-driven engineering languages and systems. Springer, Cham, pp 433–449
12. Eliasson U, Heldal R, Knauss E, Pelliccione P (2015) The need of complementing plan-driven requirements engineering with emerging communication: experiences from volvo car group. In: 2015 IEEE international Requirements Engineering conference (RE). IEEE, Piscataway, pp 372–381
13. Hardt M, Mackenthun R, Bielefeld J (2002) Integrating ECUs in vehicles-requirements engineering in series development. In: 2002 IEEE international Requirements Engineering conference (RE). IEEE, Piscataway, pp 227–236
14. Honig WL (2016) Requirements metrics - definitions of a working list of possible metrics for requirements quality. Retrieved from Loyola eCommons, Computer Science: Faculty Publications and Other Works
15. Houdek F (2013) Managing large scale specification projects. In: 19th international working conference on Requirements Engineering Foundations for Software Quality, REFSQ 2013, Essen, Germany, 8–11 April 2013
16. IEEE (1990) IEEE standard glossary of software engineering terminology (IEEE std 610.12-1990). IEEE Computer Society, Los Alamitos
17. ISO I (2011) 26262–road vehicles-functional safety. International Standard ISO/FDIS 26262
18. Jacobson I, Booch G, Rumbaugh J (1997) The objectory software development process. Addison Wesley, Boston. ISBN: 0-201-57169-2
19. Langenfeld V, Post A, Podelski A (2016) Requirements defects over a project lifetime: an empirical analysis of defect data from a 5-year automotive project at Bosch. In: Requirements engineering: foundation for software quality. Springer, Cham, pp 145–160
20. Mahally MM, Staron M, Bosch J (2015) Barriers and enablers for shortening software development lead-time in mechatronics organizations: a case study. In: Proceedings of the 2015 10th joint meeting on foundations of software engineering. ACM, New York, pp 1006–1009
21. Mahmud N, Seceleanu C, Ljungkrantz O (2015) Resa: an ontology-based requirement specification language tailored to automotive systems. In: 10th IEEE international Symposium on Industrial Embedded Systems (SIES), 2015. IEEE, Piscataway, pp 1–10
22. Manhart P, Schneider K (2004) Breaking the ice for agile development of embedded software: an industry experience report. In: Proceedings of the 26th international conference on software engineering. IEEE Computer Society, Washington, pp 378–386
23. Mellegård N, Staron M (2008) Methodology for requirements engineering in model-based projects for reactive automotive software. In: 18th ECOOP doctoral symposium and PhD student workshop, p 23
24. Mellegård N, Staron M (2009) A domain specific modelling language for specifying and visualizing requirements. In: The first international workshop on domain engineering, DE@ CAiSE, Amsterdam
25. Mellegård N, Staron M (2010) Characterizing model usage in embedded software engineering: a case study. In: Proceedings of the fourth European conference on software architecture: companion volume. ACM, New York, pp 245–252
26. Mellegård N, Staron M (2010) Distribution of effort among software development artefacts: an initial case study. In: Enterprise, business-process and information systems modeling. Springer, Berlin, pp 234–246
27. Mellegård N, Staron M (2010) Improving efficiency of change impact assessment using graphical requirement specifications: an experiment. In: Product-focused software process improvement. Springer, Berlin, pp 336–350

28. Ott D (2012) Defects in natural language requirement specifications at mercedes-benz: an investigation using a combination of legacy data and expert opinion. In: 2012 20th IEEE international Requirements Engineering conference (RE). IEEE, Piscataway, pp 291–296
29. Ott D (2013) Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements. In: Requirements engineering: foundation for software quality. Springer, Berlin, pp 50–64
30. Peraldi-Frati MA, Albinet A (2010) Requirement traceability in safety critical systems. In: Proceedings of the 1st workshop on critical automotive applications: robustness & safety. ACM, New York, pp 11–14
31. Pernstål J, Gorschek T, Feldt R, Florén D (2013) Software process improvement in inter-departmental development of software-intensive automotive systems–a case study. In: Product-focused software process improvement. Springer, Berlin, pp 93–107
32. Rana R, Staron M, Berger C, Hansson J, Nilsson M, Törner F (2013) Improving fault injection in automotive model based development using fault bypass modeling. In: GI-Jahrestagung. Chalmers University of Technology, Gothenburg, pp 2577–2591
33. Rana R, Staron M, Mellegård N, Berger C, Hansson J, Nilsson M, Törner F (2013) Evaluation of standard reliability growth models in the context of automotive software systems. In: Product-focused software process improvement. Springer, Berlin, pp 324–329
34. Rana R, Staron M, Berger C, Hansson J, Nilsson M, Törner F (2013) Increasing efficiency of ISO 26262 verification and validation by combining fault injection and mutation testing with model based development. In: ICSOFT 2013, pp 251–257
35. Sagstetter F, Lukasiewycz M, Steinhorst S, Wolf M, Bouard A, Harris WR, Jha S, Peyrin T, Poschmann A, Chakraborty S (2013) Security challenges in automotive hardware/software architecture design. In: Proceedings of the conference on design, automation and test in Europe, EDA consortium, pp 458–463
36. Siegl S, Russer M, Hielscher KS (2015) Partitioning the requirements of embedded systems by input/output dependency analysis for compositional creation of parallel test models. In: 9th annual IEEE international Systems Conference (SysCon), 2015. IEEE, Piscataway, pp 96–102
37. SimulinkDemo (2012) Modeling an anti-lock braking system. The MathWorks, Inc, Natick. Copyright 2005-2010
38. Sinha P (2011) Architectural design and reliability analysis of a fail-operational brake-by-wire system from iso 26262 perspectives. Reliab Eng Syst Saf **96**(10), 1349–1359
39. Staron M (2017) Automotive software architectures: an introduction. Springer, Cham
40. Törner F, Ivarsson M, Pettersson F, Öhman P (2006) Defects in automotive use cases. In: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering. ACM, New York, pp 115–123
41. Vogelsanag A, Fuhrmann S (2013) Why feature dependencies challenge the requirements engineering of automotive systems: an empirical study. In: 2013 21st IEEE international Requirements Engineering conference (RE). IEEE, Piscataway, pp 267–272
42. Weber M, Weisbrod J (2002) Requirements engineering in automotive development-experiences and challenges. In: 2002 IEEE international Requirements Engineering conference (RE). IEEE, Piscataway, pp 331–340
43. Wright A (2011) Hacking cars. Commun ACM 54(11):18–19