CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Analysing the Evolution of System Requirements

A case study of AUTOSAR at Volvo Car Group

Master of Science thesis in Software Engineering and technology

CORRADO MOTTA

Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden June 2016

Analysing the evolution of system requirements – A Case Study of AUTOSAR at Volvo Car Group

CORRADO MOTTA

Examiner: AGNETA NILSSON
Supervisor: MIROSLAW STARON
Industrial Supervisor: DARKO DURISIC

Chalmers University of Technology | University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden June 2016

Analysing the Evolution of System Requirements

Master thesis at Volvo Car Group

Corrado Motta
Department of Computer Science and Engineering
Chalmers University of technology | University of Gothenburg

Abstract

BACKGROUND: *The evolution of system requirements is an important and inevitable aspect of software development and maintenance. Being aware of the amount of changes as well as understanding how to measure them is advantageous not only for software engineering purposes but also in industrial contexts.*

OBJECTIVES: *This thesis aims to efficiently analyse the evolution of system requirements, by means of quantitative analysis based on a number of software metrics. Our goal is to facilitate the updates of large software systems with new features.*

METHOD: *In this paper we perform quantitative analysis of the evolution of system requirements across multiple versions of large software systems. We rely on the design research methodology and we evaluate the results of our study in a case study conducted in collaboration with Volvo Car Group.*

RESULTS: *We defined a set of metrics based on the existing studies, such as the Requirement maturity index, and we refined their input data by building the taxonomy of changes. Furthermore, we designed a metric, named Accuracy, for testing the reliability of the requirements. The empirical results assess the quality of the metrics and offer a way to monitor the requirements evolution considering the history of changes.*

CONCLUSION: *We concluded that quantitative analysis of requirements evolution using the proposed metrics can help different industrial organisations in managing software evolution by facilitating the adoption of new features in large software systems. This is achieved by visualizing the extent of the requirements evolution and indicating which requirements are mostly unstable.*

Acknowledgments

First, I would like to thank Dr. Miroslaw Staron for the support and the precious advises he provided throughout the thesis.

Second, I am deeply grateful to my industrial supervisor, Darko Durisic, who was always available to finely discuss my findings and to patiently teach me so many things. The acknowledgments are extended to all the persons who have contributed to this thesis, directly and indirectly, and to Volvo Car Group for the great opportunity.

Finally, I would like to express my deepest gratitude to my family. Without them, none of this would have been possible.

Corrado Motta, June 2016

Contents

1. Introduction

The impact of requirements evolution on large software systems is one of the major problems which nowadays affects software engineering processes and products [1]. It can affect projects in multiple aspects, such as cost, effort, time, and its final quality and reliability. This makes the competition in the global market more arduous, especially for large companies. Therefore, dealing with the requirements evolution is one of the primary objectives of large companies in order to be able to update their systems faster with less cost. Although several solutions have been provided in the last two decades mostly originating from the academia, requirements evolution is still considered one of the most challenging problems in the development of large software systems [2]. Researchers and engineers are induced to focus more on the specific context by studying the evolution of a single software product and considering it as an issue to be solved thereby losing the general nature and behaviour of the entire system.

If we look at a real industrial domain such as automotive, we can see that many automotive companies today base the development of their systems on AUTOSAR (Automotive Open System Architecture), a worldwide standard which specifies the general architecture for the system and its development methodology using a number of different types of requirements [3]. The aim of AUTOSAR is to standardize the communication between OEMs (Original Equipment Manufacturer), who are usually responsible for designing automotive software systems, and suppliers, who develop embedded software deployed to a number of ECUs (Electronic Control Unit). In order to reduce the effort of updating the automotive systems in terms of time and costs, AUTOSAR provides a set of requirements that describe the ECU middleware layer (referred to as the "basic software") and ECU application software layer. Basic software layer does not have any functional task itself and it just specifies a number of services necessary to run the application layer. On the other hand, the application layer consists of a number of software components that are responsible for executing different vehicle functions such as automated cruise control. These software components are designed by OEMs who specific their behaviour by a number of OEM specific system requirements.

The main advantage of using standards, such as AUTOSAR, is to make use of a number of reliable requirements for the parts of the systems that do not create competitive advantage, e.g., requirements for the development of middleware that are not OEM specific (AUTOSAR counts more than 21.000 basic software requirements). On the other hand, adopting a standard brings new challenges such as dealing with requirements not owned by OEMs that are constantly evolving requiring updates in the entire system, according to new versions of the standard. For example, AUTOSAR counts more than 150 partners, including both OEMs and suppliers, and the amount of software in a car is continuously increasing due to a major complexity of the electronic components [9]. Hence, it becomes necessary to also have a clear knowledge about how AUTOSAR evolves and how to adopt new AUTOSAR features based on their requirements from the new releases in a faster way. These features are needed for updating the automotive software systems with new functionalities such as autonomous drive or car-to-car communication that are heavily discussed today among car OEMs.

Generally speaking, there are at least two types of requirements evolution: Requirements evolution through different phases of a software development project and requirements evolution through different releases of a software system. The objective of this work is to define a set of suitable metrics for quantitative analyses of impact of the requirements evolution on new releases of large software systems. Hence, we define our main research question as follows:

How can the evolution of system requirements be efficiently measured in order to facilitate the adoption of new features during updates of large software systems?

In order to provide the answer to this research question, we first conducted a literature review on the existing metrics for analysing the evolution of requirements. We were interested in quantitative approach because it gives us the possibility to automatize the analysis and thereby provide an early help to the organizations who manage software evolution. We adopted the RMI (Requirements Maturity Index) metric defined by the IEEE Standard [4], a refinement of RMI, which considers the total amount of changes, defined by Anderson and Felici [5], and a series of metrics defined by Shi L. et al. [6] based on the studies of the requirements history. Additionally, we relied on a set of change metrics for the system requirements. As the number of changes is not always presented in the same way, we had to define the taxonomy of changes which specifies the types of changes that are considered in our research. Finally, we complemented these metrics by designing another metric, named *Accuracy*, for measuring the reliability of requirements and their disposition to change.

In order to define and evaluate the metrics, this paper provides an investigation in the automotive domain in a case study at Volvo Cars Group using AUTOSAR standard and its requirements as unit of analysis. Because of the amount of AUTOSAR specifications that contain requirements, the analysis of the AUTOSAR requirements evolution was executed by developing and using a software tool. Furthermore, the tool represents a mean for the engineers at Volvo Cars Groups to analyse the changes between different requirements in more details, i.e., the tool provides reports and measures about changes in different releases for different specifications. By its usage, the engineers can have practical information about the amount of changes, the content of the requirements changed, the values or statements modified for each requirement, etc. Furthermore, they can avoid reading hundreds of pages of specifications and learn what is changed in a faster way.

The results obtained in our study point out the quality of the proposed approach and increase our knowledge and awareness of the requirements evolution. Furthermore, they validate the metrics used, including the *accuracy.* We showed that using quantitative and automated analysis we can support engineers in assessing the impact of changes and thereby making the process of adopting new versions faster.

The next sections are organised as follows: Section 2 talks about the theoretical concept as the basis of this research; Section 3 provides a summary of the related work; Section 4 shows the research methodology we used during this project; Section 5 provides descriptions of the proposed metrics; Section 6 presents the application and validation of the metrics on the case study; Section 7 provides the discussion of the results; finally, Section 8 concludes our work and describes our plans for future work.

2. Background

In this section we describe the general terms and concepts used in this paper, namely the definitions, behaviour, and importance of requirements in software engineering.

2.1. Requirements engineering

In this thesis we analysing *software requirements*. The term is known in software engineering from the beginning. For this reason several definitions and interpretation are used for describing it. A general but precise definition was defined by Sommerville et al. [7]:

Requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.

This definition highlights the importance of establishing, understanding and documenting the requirements during the whole software development process. These important actions, considered together, form the process named *Requirements Engineering.* This process involves a large number of participants. We can identify two macro categories: The customers, or, in a more general view, the stakeholders (i.e., all the entities that have an interest or a role in the system), who request features to the system, and the requirements engineers, who communicate with the stakeholders in order to understand, elicit and write down the requirements.

The term *requirement* is not always used with the same meaning. There are different types of requirements with different levels of abstraction. The lack of separations during the requirement elicitation leads to serious errors during the subsequent steps of the process [8]. There are two different levels of descriptions for the same requirement:

- *User requirements***.** High-level of abstraction. A natural and comprehensible language is used, together with diagrams and figures in order to explain the purpose of the system and its constraints. The aim of user requirements is to present the information to the customers as clear as possible.
- *System requirements.* Low-level description. The system's function and general features are explained in detail. System requirements documents are used by all stakeholders that need to know the system accurately (e.g., software engineers who are designing the system). They are usually written with a natural language notation because it does not requires any further knowledge for understanding them. However, with a natural language, there are several ways for expressing the same concept and it is not easy to modularise the requirements; this may lead to misunderstandings. Furthermore, analysing and extracting data from the document is fairly complicated. System requirements can be also written with more specific notations, such as *Design Description languages, Graphical notations* or *Mathematical Specifications* (e.g., finite state machine)[8].

Then the requirements can be grouped in two types:

- *Functional Requirements.* Typically they specify the behaviour or a function that the system shall be able to perform. How the system should react to particular inputs or how the system should behave in particular situations.
- *Non-Functional Requirements.* They define the "qualities" or attributes of the system. These requirements add internal restrictions on the services offered by the system, or external constraints that the product shall meet. An example of them is showed in Table 2.1. Non-functional requirements may be even more critical than functional requirements. If these are not met, the system could be useless.

In some cases, non-functional requirements are used to specify how to *design* a specific project, product, and system. For instance, in software engineering, the use of models and meta-models, that define languages for the models, is usually recommended instead of a direct code development because it raises the level of abstraction thereby increasing the understanding of the system and facilitating its testing. To specify the language for the models and to use it as an input for developing modelling tools for creating the models and generating code from them, requirements are needed. We can grouped these requirements in two further groups:

- *Design Requirements:* All the definitions and the properties that the meta-model shall meet.
- *Constraints:* A constraint is a requirement that does not add any new functionalities to the model. Instead, it inserts controls or restrictions to one or more features that a tool-based implementation of the meta-model shall meet.

Table 2-1 Types of requirements and examples

Types of requirements	*Examples*
Functional Requirement	The user shall be able to log in to the System by providing his username and password
Non Functional Requirement	If the credentials are correct, the user shall be redirected in his personal page within 10 Seconds.
Design Requirement	The attribute *username* of the class *User* defines the name inserted by the user during the login.
Constraint	The value of the attribute *user* shall be in the range [0, 64] (bytes).

Generally, writing requirement is not an easy task and requires experience. Any mistake or misunderstandings during this phase can lead to errors, often difficult to detect, that can infect the subsequent steps of the process or, in the worst case, the release of the system. A good requirement, according to Manfred & Simmons [8], should be *feasible* (its implementation shall be possible), *valid* (the requirement is one that the system shall meet), *unambiguous*, *modifiable* (changes are possible and easy), *consistent* (not in conflict with other requirements or external documents, etc.), *complete*, *verifiable* and *traceable* (it is possible to view the life of a requirement through the whole process, included the tests). In order to respect all these points, the requirement are written with a structured language specifications, where the requirement's form is standardise and should follows some constraint and rules. It depends strictly on the context and on the type of requirements.

2.2. Software requirements specification

Software requirements specification (SRS), sometimes called software requirements document, is the document in which all requirements, functional and non-functional, are reported. The main characteristic of this document should be its *completeness*. A requirements specification is complete if it is understandable by all the stakeholders who need to consult it (from the manager who required the system to the developers) and if it includes all the requirements and the responses of the system. Furthermore, a complete requirements document should also provide some extra contents as tables, context diagrams, use cases, figures and all it is needed for a full understanding of the system. For fulfilling these requests, a common structure template is used. The most well-known is the IEEE standard that provide a general guideline for writing a complete requirements document. According to IEEE [9], the main sections are:

I. *Introduction.* Explain the purpose of the document, provide an overview of the structure (i.e., a Table of contents), and list the references and the glossary.
II. *General description.* This section is more focused on the product. The functionalities of the product explained in the document, the user features and the general constraints.
III. *Specific Requirements.* List all the functional, non-functional and interface requirements. This chapter is usually long and it strictly depends on the system. For this reason IEEE does not provide any further sub-sections template in this section.
IV. *Appendices.* Further information about the system. For instance, a report about its evolution or the hardware description.
V. *Index.* Several indexes could be provided (e.g., index of tables, requirements, figures, etc.).

2.3. Requirements hierarchy and traceability

During software development processes, requirements are usually elicited by following structured processes. For having a clear knowledge about how high-level requirements, such as objectives, main goals, needs, etc., are turned into low-level requirements, traceability is needed [10]. For example, in a business context, the interest points to the business objectives and visions whilst in an engineering context the interest may be focused on the system requirements. It is necessary to keep all the requirements traceable in order to maintain a full understanding of the system as well as the ability of evolve, manage or reuse it [10]. The requirements are usually connected with many to many relationships and there are different ways to represent them. For example, by using a matrix in appendix to relevant document or by using hyper-linked documents, where it is possible to freely move through the statements. Otherwise, a hierarchy of requirements, grouped in relevant documents, as showed in Figure 2.1, could be provided, with different level of abstraction and objectives.

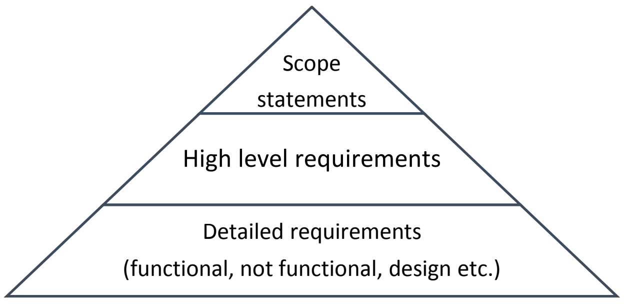

Scope statements
High level requirements
Detailed requirements (functional, not functional, design etc.)

Figure 2-1 Example of requirements hierarchy

2.4. Requirements evolution management

Since most of the systems today are not stable products but need to be changed, improved or upgraded during their life-cycle [11], stakeholders should be aware of the occurrence of changes. The change management could be subtle in some situations, for example when a requirement is in relationship with many others and a change may lead to a chain of changes. For this reason, when a problem is identified or a change proposal is raised on a requirement, it is very important to analyse it carefully in order to check its validity and the cost associated with the change (also in terms of modifications and feasibility). Then, if the reply is positive, it is possible to proceed with the *change modification.*

A change can be required between different steps of the software development process or between different versions of the software released. Both of them could be problematic and have a negative impact on either the development plan, in the first case, and the maintenance one, in the second case. As a consequence big changes may cause re-planning and cost fluctuation. This first aspect is most known as *changes management* since the engineers need to check the impact, the motivations, and the efficacy of each change proposed.

If we consider the use of standards, the changes are represented by new releases of a standard, where the requirements are already changed and should be understood, selected (sometimes modified again, not considered etc.), and adopted in new version of the system. This second aspect is mentioned as *evolution management* because the companies and the engineers involved in this process need to deal with a great number of changes that come from an external product.

3. Related work

In this section we briefly explain all the papers and methods which had a role in the choice and design of the metrics used in this study.

Several studies about requirements evolution can be considered for the purpose of this thesis. Wang, H et al. provide in [12] a general method for studying the requirements change with a quantitative analysis. Their approach, based on consider each modification, addition, and deletion, has been considered as a starting point for this thesis work. Similar but more exhaustive study has been conducted in the avionics context by Anderson S. and Felici M. in several papers [2, 5]. Their approach is made of two different layers named Empirical Analysis (EA) and Product Oriented Refinement (OOR). The first one is focused on collecting information from the Avionics case study by analysis of data and relies on the *Requirement Maturity index (RMI)* metric. The second one aim to refine the information gathered by focusing on the product. Hence, in these articles they show how to conduct an Empirical Analysis starting from a general point and moving to a product-oriented one. From the first layer, we identified two further interesting metrics based on *RMI*: the *Requirement Stability Index* (*RSI*) which considers the cumulative number of requirement changes and the *Historical Requirement Maturity Index* (*HRMI*), which is also similar to *RMI* but takes into account the average of all the changes made so far. We decided to adopt *RMI* and *HRMI* since they efficiently show the stability of the versions and how the stability changes in relation with the past releases. *RSI* is not considered since great number of changes can lead to negative number or too meaningless results for our scope.

For measuring the requirements behaviour, we considered the study of Shi l. et al. [6] which aims to predict requirements that are possible to be changed in the future, basing on historical information (i.e., previous versions of the software). Although prediction is outside of the scope of this thesis, several metrics are considered significant for measuring requirements and for analysing trends between changes in different releases. They constructed six metrics for measuring the history of changes, the pace of change, and the volatility of each requirement. These measurements are useful since they give results both for a single requirement or for groups of them using average values, hence we adopted four of them. We added the *accuracy* for taking in consideration the reliable aspect of the requirements. In order to study the evolution, Nurmuliani et al. [13] have provided a taxonomy of changes for categorizing the change types, their reason and their origin. We took inspiration from it for making our taxonomy of changes.

Lastly, Stark g. et al. [14] proposed a method for analysing the requirements evolution based on two different steps. After a general overview of the evolution behaviour of the system, they proceed with a *micro* view of requirement changes, on one release. Then they constructed a *macro model* for foreseeing the effect of requirements change on all the releases. The models are based on data from historical releases. We adopted this approach for having a preliminary view of the requirements architecture, the documents structure and for a first application of the metrics.

In order to understand the field of the case study, several papers have been considered from the automotive context. For the AUTOSAR perspective, two studies of Durisic et Al. [15, 16] were useful for improving our knowledge of AUTOSAR, its architecture, methodology and complexity. More in general terms, Broy et Al. [17] point out the current role of automotive software and how it is growing but do not mention how to measure this evolution.

Although there is a significant number of methods related to the analysis of requirement evolution, only few of them are applied in a real industrial context in which large software systems are developed. This thesis aim to combine existing studies with personal considerations and metrics for providing an efficient way to measure requirements evolution.

4. Research methodology

The aim of this section is to provide general information about the approach and the procedures used during this thesis project. We adopted the design science research [18] as methodology for this thesis. The content is organised in two subsections. The first describes our research questions and the second talks about our research method.

4.1. Research questions

The objective of this study was to efficiently analyse and measure the evolution of requirements and to facilitate the update of large software systems by reducing time and costs related to it. We proposed a set of metrics for this purpose and we applied them on AUTOSAR requirements. In order to fulfil this objective we decided to have a main research question and a series of sub-research questions. The main research question, as presented in the introduction, is expressed as follow:

How can the evolution of system requirements be efficiently measured in order to facilitate the adoption of new features during updates of large software systems?

We want to measure the system in an efficient way. In order to do that we divided the analysis of requirements evolution in smaller objectives that correspond to a number of research questions. We aim to analyse each of these sub aspects by selecting or choosing the best metrics and we believe that focusing on smaller goals can bring advantages in order to make the most efficient measurements. Furthermore, we provide the answer to the main research question by proceeding step by step in a structured way. Hence, in this study, *efficiently measure the evolution of system requirements* is considered the result of choosing or designing metrics for each aspect of the evolution, testing their effectiveness and gathering all together. The sub research questions are defined both according to the literature review and our case study context, i.e., they are inspired by the problems identified in the automotive domain where companies develop their systems based on the AUTOSAR standard and its requirements. However we decided to describe the research questions in a general form without talking about the case study context as they are applicable to other contexts that make use of large software systems with similar characteristics to the automotive domain. The sub *RQs,* named *SQs,* are defined as follow.

SQ1: Which releases of a software system are most stable and which one change most?

This question is considered useful for pointing out the roles of the releases and their behaviour. Firstly, the evolution is observed from the releases point of view for being aware about the amount of changes to a number of requirements and to identify the trends in their evolution. For doing this, RMI and HRMI metrics shall be applied.

SQ2: Which types of changes are more common and how they are connected to the releases?

This second research question is a deepening of the first one. After overview the impact of evolution we start to look within each release in a more specific way trying to find common pattern among the types of change. For example we do this by checking if the number of requirements increase for each releases or if the number of removed requirements is significant.

SQ3: Which requirements specifications are most unstable?

SQ3 is the last question about evolution at the releases level and it aims to understand the evolution of requirements for a specific context. The requirements specifications group requirements in different sets depending on their topics and tasks. For example, in the AUTOSAR context, *AUTOSAR_SWS_SAEJ1939DiagnosticCommunicationManager* defines the message structures and behaviour of so-called ‘Diagnostic messages’ (DMs) which are used for diagnostic communication in J1939 networks. We analyse the stability of each of them and we investigate about their behaviour.

SQ4: Which categories of requirements are most stable and which ones are mostly affected by the evolution of the system?

From this research question we move to study single requirements behaviours. Through a series of metrics we measure the stability of the requirements, their accuracy, their frequency of changes and the “age”, i.e., the moment in which they are introduced in the system. By investigating this data we can answer this research question and the following one. Here we wonder about the stability of requirements categories such as functional and design.

SQ5: Which types of requirements are more characterized by subsequent changes and which one are more reliable?

The objective of this last research question is similar to SQ4 with the difference of considering the relation between multiple changes. We introduced this question for testing our *accuracy* metric and for investigating about the behaviours of requirements that have been already affected by changes. For example, by measuring if there is a relation between the requirements added in a release and the requirements changed in the subsequent version. These types of trend could be used as an alarm signal to the engineers that need to know when update the system.

4.2. Research Method

The methodology used for this thesis is the design science research methodology, which is summarized in Figure 4-1, in the white boxes.

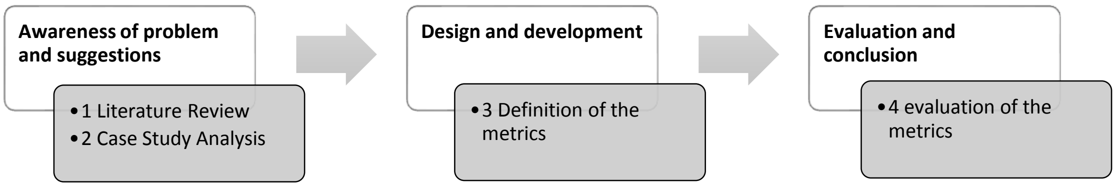

Awareness of problem and suggestions
•1 Literature Review
•2 Case Study Analysis
Design and development
•3 Definition of the metrics
Evaluation and conclusion
•4 evaluation of the metrics

Figure 4-1 Design science research methodology

We followed five steps directly associated to the design research methodology, listed in the Figure inside the black boxes and described in the next subsections.

- Awareness of problem and suggestions
  - I. *Literature review* – A literature study on the existent metrics and approaches.
  - II. *AUTOSAR analysis* – A general study of AUTOSAR.
- Design and development
  - III. *Definition of the metrics* – The selection of the metrics used in the case study.
- Evaluation and conclusion
  - IV. *Evaluation of the metrics on AUTOSAR*
    - *a. Application of the metrics* – Apply the metrics to the case study
    - *b. Validation of the results* – Validate the metrics from the case study results.

4.3. Literature review

In order to choose and define our metrics, we conducted a literature review on the already existent approaches. The literature review is considered of central importance in order to be aware about the actual "state of the art" or general development achieved in this specific field. The literature review has been conducted using the key words "*requirements evolution", "requirements change",* and "*requirements volatility".* These three terms look very similar but they have consistent differences. *Requirement evolution* concerns the actual evolution of requirements through different version or releases of a specific software, so it is considered the closest term to this research. *Requirement change* is a more general term which can be associated to changes either during the *elicitation* of requirement (i.e., during one of the steps in the software development process) or through the releases. *Requirement volatility* is a branch of requirement evolution which tries to deal with the unpredictability behaviour of the requirement. The aim of studying the *requirements volatility* is to provide a method for foreseeing the state of requirements in future releases. For instance, try to anticipate what requirements, modules or functions are going to change or remain stable in the next version. We considered these keywords because all of them have in common the idea to measure the evolution of requirements and to investigate the effects of such evolution. Other terms, such as *managing changes* were not considered both because more related to the aspect of change management rather than evolution management and too generic.

The literature review has been conducted using the *snowball method* [19]. This method present some guidelines for writing an efficient and reliable literature review. We chose a *start set* of three papers using the key words, and looking for the most relevant, namely the ones cited by several other articles and also more close to our research. Then, we proceed backward and forward using respectively the references and the citations. Google scholar and the servers “IEEE Xplore: digital library” and “Scopus” provide features as “cited by”, “abstract” and “references” which can be useful for analysing all the documents. In the *snowballing method* there are different iteration as well. The first one is about a cycle of backward and forward analysis from the *start set*. Then the second cycle iterates on the outcomes of the first one and so on. A general advice is to perform at most three or four iterations, both for having a good set of outcomes and for not going off topic.

4.4. AUTOSAR analysis

After the literature review we analysed requirements evolution in the AUTOSAR context. We used three different ways for gathering information from AUTOSAR:

I. By using some general information sources as [3] or by reading general papers written on AUTOSAR, such as [15, 16].

II. Through several study sessions with an AUTOSAR expert from VCC.

III. By performing a *MICRO* and a *MACRO* analysis on some AUTOSAR requirements documents.

The study sessions were performed through the first three months and were organised either as lectures or as unstructured interviews. These meetings have had a strong relevance for the project. Firstly, they elucidated the complete structure of AUTOSAR. Secondly, they highlighted the relevance aspects of AUTOSAR for this work. Lastly they pointed out the reference points for conducting the research. Then, we conduct a preliminary semi-automatic analysis using two requirements specifications in order to improve our knowledge about the structure, the form and the features of AUTOSAR. We called it *micro* and *macro* analysis. Micro analysis is a comparison between two entire requirements specifications. This comparison aims to point out the types of change, the structure of the document, and the semantic used in the requirements. In order to perform the comparison we made use of an open source tool named *WinMerge* [20], which compares two PDFs and highlights the differences. On the other hand, the *macro* analysis is the study of a small set of requirements (e.g., ten in total) through all their lifecycle in the considered releases. The objective of the *macro* study is to understand conventions, structural, and semantic changes of AUTOSAR during the considered releases in order to develop an efficient tool. Both approaches are considered as *suggestions* because they give the opportunity to suggest a number of metrics, to test them and to choose the relevant ones.

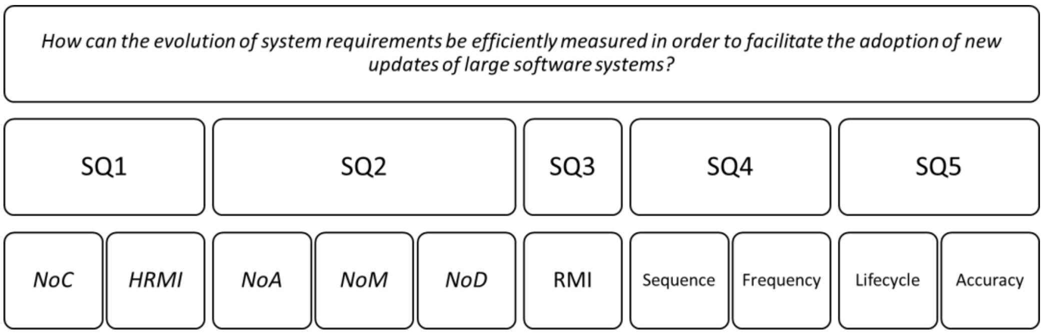
For being aware of the structure and the semantic of all the specifications we selected both an *SWS* and a *TPS* document. From the study sessions at Volvo Car Group we agreed to consider the following two requirements specifications:

- *AUTOSAR_SWS_COM*
- *AUTOSAR_TPS_SystemTemplate*

Both of them have a relevant role in the AUTOSAR architecture. The first one acts as an interface for the application (automotive software components) layer, by packing or unpacking signals and providing a communication transmission control. The second one is based on the AUTOSAR *meta-model* and defines the relationship between the pure software components view on the system and a physical system architecture with ECU instances.

4.5. Definition of the metrics

The metrics were defined by using the Goal Question Metric Approach (GQM) [21]. Figure 4-2 shows the GQM model applied to this study. The metrics are described in details in section 5. This model relies on the assumption that for choosing or designing metrics in an accurate way, the researcher shall follow a goal based workflow. Firstly, they need to specify the objectives, then, to link them to the data that are considered significant, and finally, to *provide a framework for interpreting the data with respect to the stated goals* [21].


How can the evolution of system requirements be efficiently measured in order to facilitate the adoption of new updates of large software systems?
SQ1
SQ2
SQ3
SQ4
SQ5
NoC
HRMI
NoA
NoM
NoD
RMI
Sequence
Frequency
Lifecycle
Accuracy
Figure 4-2 Goal Question Metric Approach

The GQM model is composed by three levels:

I. *Conceptual level* – One or more goals defined for the study
II. *Operational level* – A set of questions for characterizing how to achieve the goals.
III. *Quantitative level* – A number of metrics associated to each question in order to quantitatively measure the data and answer the questions.

The GQM approach is used for finding the most efficient metrics in order to answer to the main research question.

4.6. Evaluation of the metrics on AUTOSAR

The evaluation of the metrics is composed by two steps. First we applied the metrics on the case study and then we validated them by supplying a survey to AUTOSAR experts.

4.6.1. Application of the metrics

The metrics are then applied and tested on AUTOSAR in collaboration with Volvo Car Group. AUTOSAR counts more than 21.000 basic software requirements in its last version, hence we needed to automatize the data collection. We built a configurable tool for gathering data and applying the measurements to the requirements specifications. The tool compares different versions of specifications in PDF and makes report on their changes. These types of data are generally called *second and third degree* [22] because come from sources not originated for studies but for business purposes. The data are collected according to their characteristics and to our research questions in the following way:

- Types of change through all the releases.
- *Changes history* for each requirement.
- Number of requirements, cumulative number of requirements, number of changes and cumulative number of changes for each release.

The same values are collected also for the constraints. From these raw data we built a requirements map for effectively applying all the measurements. The details are showed in Section 5.

The data collected concern all the releases from the last major version (R4). Table 4.1 shows the considered releases. The decision to study only these releases relies on two aspects. Firstly, Volvo Cars Group does not have interest in the previous major releases, since are not used today. Secondly, AUTOSAR had a lot of significant changes in the last major release. A great number of requirements has been added in one of the last eight versions from which we can collect enough data for obtaining significant results.

According to AUTOSAR terminology, within a major release we have two types of updates:

I. *Minor release,* for which the central digit changes.

II. *Revision,* for which the last digit change.

Minor releases are showed in the first column of the Table and usually bring new features and functionalities inside the major release. The Major releases are usually big clean-ups that are backwards incompatible. The *revisions* are listed in the corresponding rows and usually are less significant releases. *Minor releases* and *revisions* are backwards compatible.

Table 4-1 Considered releases of AUTOSAR

Release	*12/2009*	*04/2011*	*12/2011*	*03/2013*	*08/2013*	*03/2014*	*10/2014*	*07/2015*
4.0	R4.0.1	R4.0.2	R4.0.3	↴				
4.1				R4.1.1	R4.1.2	R4.1.3	↴	
4.2							R4.2.1	R4.2.2

For each release, the requirements specifications studied are the entire set of basic software and design requirements, namely all the standard documents named *SWS* and *TPS* that are described in detail in Section 6.1. Although we did not analyse the entire hierarchy, which may be regarded a further work, the results are considered complete for the auxiliary nature of the upper levels of the hierarchy. By studying the lowest one, it is possible to collect and analyse data that concern all the objectives and features of AUTOSAR.

Finally, in the AUTOSAR lowest level, the requirements are mentioned as *specifications items*, however the meaning for the purpose of this research remains the same. Therefore in the results we use indifferently the two terms, requirements and specifications items, with the same standard meaning.

4.6.2. Validation of the results

The results are validated through a survey supplied to six AUTOSAR experts, from Volvo Cars Groups, who are involved in AUTOSAR software management process or other AUTOSAR projects. These surveys aim to assess the reliability and the quality of the measurements by assessing the results obtained. Since the results are related to our metrics with the GQM model, their validation is directly associated to the measurements. We designed a process, in Figure 4-3, to perform the validation.

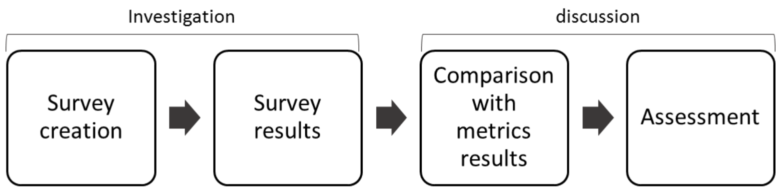


Figure 4-3 Validation workflow.

The first phase is called *investigation* and is made of two steps:

I. *Survey creation.* We decided to make questions only with multiple answers (appendix A for details) in order to study the results by percentages of correct answers. The surveys are composed by ten questions, five about general characteristics of AUTOSAR evolution and five about specific behaviours of two software requirements documents. We talked with each expert for understanding which specifications are most suitable for them.

II. *Survey results.* We supplied the survey and waited for answers. The experts were not informed about the measured results, so they had to answer with their *expectations.* Since they work in different context, the five specific questions concerned different requirements specifications.

We called the second phase *discussion.* Here, we interpret the results in the following two steps:

I. *Comparison with metrics results***.** The questions are related to the measurements made during this study. We compared the measured results with the expected results. The five specific questions are evaluated individually and used for assessing also the utility of the tool.

II. *Assessment.* In this step we discussed both results and we point out correspondences as well as discontinuities between the expectations and the measurements. Based on the outcomes of this step we can consider the results, and thus the metrics, validate or not.

5. Metrics definition

In this section, we describe our interpretation of requirement change, i.e. the taxonomy of changes, and the metrics we used in the analysis. We chose 5 existing metrics and we designed one new metric.

5.1. Taxonomy of changes

The usage of the term *change* in the context of our study is related to three possible types of requirement changes: addition, modification and deletion. Table 5-1 shows our interpretation of change-related metrics *NoA*, *NoD*, *NoM* and *NoC*.

Table 5-1 Types of change considered

Name	*Description*	*Equation*
NoA	Number of added and eventually split requirements	-
NoD	Number of deleted and eventually merged requirements	-
NoM	Number of modified requirements	-
NoC	Total number of requirements changed	*NoC = NoA + NoD + NoM*

It is very important to clearly specify the changes considered in order to correctly understand the measurements. In this study, the total number of changes is considered as the sum of all added, modified and deleted requirements. We did not apply weights to different types of changes since we believe that the effort needed for updating the system according to the added, modified and deleted requirements depends more on the requirements content rather than the type of change. A requirement added could be critical or have just a minor impact on the system, or a requirement modified could be an easy property to fix or a complete change in its functionality.

Additionally, there are two other types of changes that are less likely: split and merged requirements, i.e., when a requirement is divided into two new requirements or vice versa. Although splitting and merging are two effective types of change, we consider them as parts of *NoA* and *NoD*, respectively, because of the low probability of appearance and the difficulty in detecting and counting them. Furthermore, a requirement split or merged with another one is usually a consequence of a planned change in the content of the same requirement.

One of the more important things that should be taken in consideration, when the types of change are defined, concern the *modification* concept. We can accurately identify which specifications are added and deleted whilst defining whether a requirement is modified needs our judgement. One requirement can have different types of modification among different versions. Not all of them should be considered meaningful and there are no general rules for this purpose. Since the modifications can be valued in a different way by different researchers, the total number of changes could be affected in a sensitive way and conduct to complete different results. Table 5-2 shows our taxonomy of the main types of modification that usually occur during an analysis of requirements modifications. We made this taxonomy by both referring to the literature review and studying the AUTOSAR context and discussing our finding with the practitioners.

Furthermore, in the last column, we reported our judgment. The sum of the "considered" rows establish the number of modifications (*NoM*).

Table 5-2 Taxonomy of modifications

Types of modification	*Description*	*judgment*
Grammar and form corrections	Form and grammar mistakes fixed in a new release.	Not considered.
Encoding modification	The document can be encoded in different ways and the output could slightly change.	Not considered.
Form modification	The form can change. For example the structure of the requirement ID or how the requirements are structured (e.g., in tables or just text requirements)	Not considered.
Object's name modification	For instance the change in the name of a function, a class or a meta-class etc.	Considered.
Title modification	The title of a requirement (if it has a title)	Considered.
Content modification	The actual modification in the content of the requirement.	Considered.
Reference modification	The modification in the reference for the traceability of requirements.	Not considered.

5.2. Metrics

Six metrics for interpreting the data are used in this study. The purpose of our measurements is firstly to have general outcomes about the evolution of requirements and then to conduct a deeper analysis. We chose and designed metrics based on our objectives, by following the GQM model.

SQ1, *SQ2*, and *SQ3* aim to study the evolution from the releases point of view. In addition to the *NoC,* we selected two metrics for overviewing the evolution. The first one comes from the IEEE standard [4] and is named *Requirement Maturity Index.*

$$RMI = \frac{Rt - NoC}{Rt}$$

This metric studies the stability of each requirement. *Rt* is the total number of requirements for a specific version. Since *RMI* is calculated for each version, it does not take count of the historical information about changes but only the *NoC* among two following releases. For this reason, Anderson & Felici [5] considered this metric too pessimistic. So they defined a refinement of *RMI* which concerns not only the *NoC* for each release, but also the average *NoC* (*ANoC*), which takes count of all the previous versions. They called this refinement *Historical Requirements Maturity Index.*

$$HRMI = \frac{Rt - ANoC}{Rt}$$

The average of changes, *ANoC,* is calculated as the cumulative number of changes divided by the number of previous versions considered.

For *SQ4* and *SQ5* we need to observe the evolution by analysing the "life" of each requirement. For achieving the questions we make use of 4 metrics. In order to efficiently perform these measurements we made a table representing the life of each requirement. An example of how the table looks like is showed below in Table 5-3. Each row is a requirement and each column represents a past version of the system. In the boxes there are different values which correspond to requirements behaviours between that release and the one before: “1” means added, “2” modified, “3” deleted, “0” unchanged and “-“ not present/considered. In this example, version 1 is the first release considered, thus all the requirements do not have a value (i.e., “-”), whilst version 8 is the last one. *Req_n* is the requirement id. For example, requirement 1 was added in version 3, modified in version 5 and deleted in version 6.

Table 5-3 Requirements table example.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Req_1	-	-	1	0	2	3	-	-
.								
.								
.								
Req_n	-	1	0	2	2	0	0	2

Three of the following four metrics are selected by the studies of Shi & Al. [6] and are considered important measurements for the requirements study. The first one is named *sequence*, which measures the biggest sequence of consecutive changes (i.e., added, modified, deleted) for a requirement. We found this measure interesting for analysing the unstability of requirements which are affected by changes.

$$Sequence: \quad r = \max\big(N(r,i)\big), (2 \leq i \leq n)$$

N(r,i) denotes the number of versions that *r* continuously changed from its added version to the version *i*. For example, taking the values showed in Table 5-3, *Req_n* has a sequence *r=2.*

The second metric adopted from [6] is the *frequency.* Frequency measures the total times that a requirement has been changed in all of its historic versions. The frequency is an interesting metric for checking the general unstability of the system or for one subset of requirement specifications. Further we can compare its outcomes with *RMI* ones.

$$Frequency: \quad r = \frac{1}{n-1}\sum_{i=2}^{n} isChanged(r,i)$$

The result is divided by (*n-1*) for scaling the result in the range [0-1]. For Example, Req_n has a frequency of change *r=4/(8-1)=0,57* . These two metrics are used for *SQ4.*

The last metric adopted is the *Lifecycle*, and aims to check the hypothesis whereby *requirements with short lifetimes are unstable.*

$$LifeCycle: \quad r = n - VA + 1$$

N is the current number of version, *VA* the version number of the introduction of the requirement r. Then the result is increased by one to consider the last version. For example *Req_n* has a lifecycle *r= (8-2+1) =7.* The lifecycle of a requirement inserted before the considered releases and still used is labelled as 9.

Lastly, we built a further requirement named *accuracy*. We designed this metric for measuring the reliability of changes. Is based on the idea that a requirement change is *accurate* if in the immediate next release we do not have again a change for the same requirement. This metric, together with *lifecycle,* are used for answering *SQ5*.

$$Accuracy: \quad r = \frac{\sum_{i=2}^{n-1}\left(Rc(i) \rightarrow \ Rs(i+1)\right)}{\sum_{i=2}^{n-1} Rc(i)}$$

R is the requirement and i is the corresponding release. Rc(i) denotes a requirement that have been added or modified in version i, whilst Rs(i+1) denotes a requirements that is unchanged in version i+1. For example, Req_n has an accuracy r = *(2/3) = 0.66*

6. Application of the metrics

The metrics have been applied and validated on AUTOSAR case study with the collaboration of Volvo Car Group. This Section starts with the presentation of the study context. Then it describes the evaluation of the metrics by calculating the results of our metrics and presents the validation results for the metrics.

6.1. Case study context

6.1.1. Automotive system development process and the role of AUTOSAR

The objective of system engineering is to generate a system that represents a design solution and meet certain needs or requirements. Therefore system engineering transforms the requirements to a system definition. This process is not unique and one-way but it is iterative or recursive. There are different models used for this purpose. The Automotive sector has adopted the *V-Model* [23] for the car electrical systems. The V-Model guarantees the cooperation of different entities in order to design and develop software products. In the automotive domain the software and hardware products are developed by a number of suppliers. The OEMs just design the components by using specific models and provide a set of requirements specifications to the suppliers. The communication between OEMs and suppliers is usually critical since the final product shall meet the model provided by OEMs. For facilitating this communication and for standardizing the middleware requirements, AUTOSAR architecture has been introduced in 2003.

AUTOSAR *is a worldwide development partnership of vehicle manufacturers, suppliers and other companies from the electronics, semiconductor and software industry* [3]. As said above, the purpose of AUTOSAR is to standardise and facilitate the communication between OEMs (i.e., original equipment manufacturer) and automotive suppliers, through the creation of an open industry software architecture. This standard is used in the context of the automotive ECUs. In order to achieve these goals, AUTOSAR provides an *ECU* architecture made of three layers (Figure 6-1):

I. *Application software layer*

II. *Runtime Environment*

III. *Basic Software*

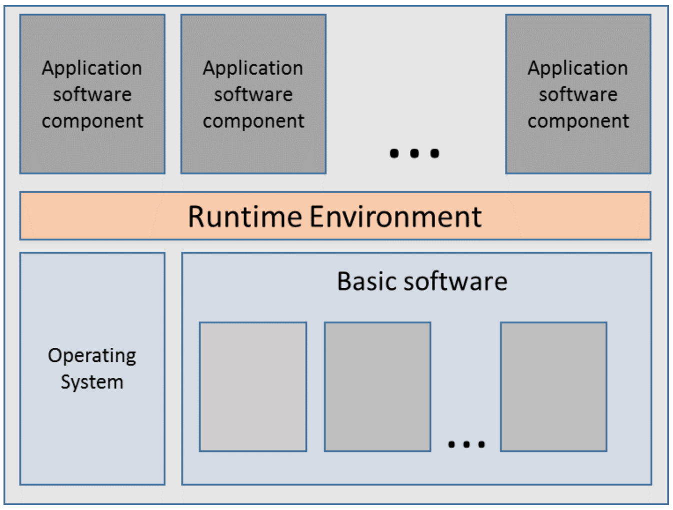

Application software component
Application software component
Application software component
Runtime Environment
Operating System
Basic software

Figure 6-1 AUTOSAR ECU architecture [4]

The Basic Software contains several modules which are essentials to define the communication between different ECUs, while the Runtime Environment is an abstraction level for the communication inter and intra ECUs. We can consider both of them, together with the operating system, as *middleware*. The middleware provides services to the AUTOSAR application layer and run the functional part of the software. It does not have any functional job. For this reason, the middleware is completely standardize by AUTOSAR, which provides a set of requirements specifications, and adopted by OEMs. The application software layer is composed by multiple entities called software components. The functionality of an ECU or multiple ECUs is driven by these entities. For maintaining the competition between OEMs, the internal structure and the connection points of the software components are not standardized by AUTOSAR, but are designed by OEMs and delivered to suppliers. However, AUTOSAR provides also the domain model and meta-model used for designing these components.

The usage of this development process requires three different types of requirements:

I. *Functional requirements* - requirements specified by OEMs for the software components.

II. *Design requirements* – requirements and constraints standardized by AUTOSAR for modelling the components and setting the tools model-based.

III. *Basic software requirements* – requirements standardized by AUTOSAR for the middleware.

Design requirements are grouped in document and packages called *TPS* specifications, i.e., AUTOSAR meta-model and templates, whilst the Basic software requirements are grouped in *SWS* specifications, i.e., standard basic software. Both of them are standardized by AUTOSAR. On the other hand, functional requirements are specified by OEMs since they concern functional jobs and introduce competition in the automotive market. In this study we focus on AUTOSAR, therefore we do not analyse the OEM requirements.

One of the AUTOSAR objectives is to guarantees a full traceability for the requirements by providing a hierarchy made of different levels. Figure 6-2 shows the overall structure of the requirements specifications architecture in AUTOSAR. The arrow starts from the most specific document and points to the most abstract. The aim is trace a requirement from its effective description to the top-level group of belonging. These top level groups are described in the *project objectives*. Then the *Main requirements* package is still dedicated to general requirements, whilst all the *features* of AUTOSAR (including Basic Software (BSW) and the Run Time Environment (RTE)) are discussed in the homonym document. Below these general guidelines the requirements become more detailed. A first significant distinction is below the *Features*. *SRS* and *SWS* contain the requirements for the basic software and the *RTE*. *RS* and *TPS* specify the requirement templates. The two following subsections clarify this distinction.

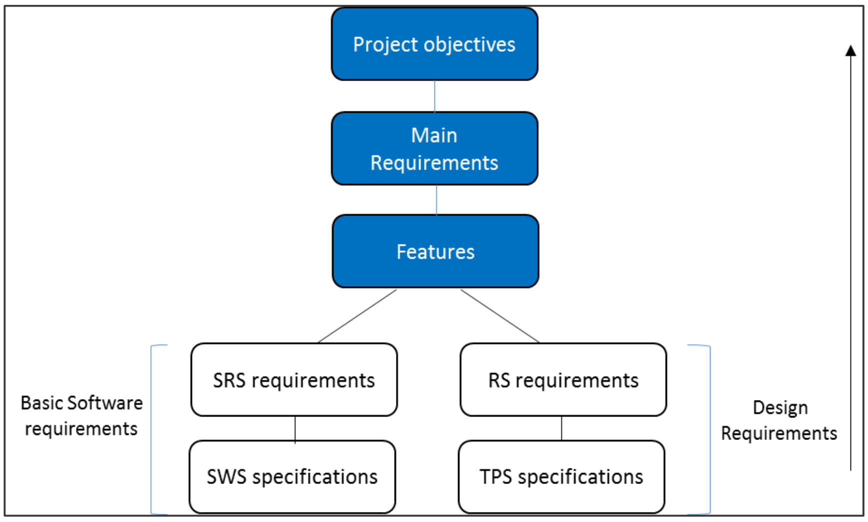


Figure 6-2 The AUTOSAR specifications hierarchy

Basic software requirements

SRS and SWS are divided into a number of requirements specifications. AUTOSAR distinguishes between SRS "requirements" and SWS "specifications" in a different manner than traditional software engineering.

- *SRS Requirements.* They are generally considered "auxiliary" documents and are used to specify the higher level requirements for a group of Basic software Modules. It is usually not necessary to read these requirements for implementing the functionality of different basic software modules and they are mostly used for traceability of low level requirements to the high level AUTOSAR features. For example, the document *AUTOSAR SRS CAN* contains the higher level requirements for the following Basic Software Modules: *CAN Driver, CAN Interface, CAN State manager, CAN transport layer, and CAN Bus Transceiver Driver*.
- *SWS specifications.* They describe the functionality, API and configuration of one AUTOSAR module. For the OEMs perspective these packages are usually most interesting and are considered "requirements" as well.

Design requirements

Since the software component layer in the ECU is not standardised by AUTOSAR but it is implemented directly by OEMs, there is the need of a common exchange format between OEMs and Suppliers. AUTOSAR has achieved this objective by introducing a domain specific *meta-model*, namely requirements and constraints which specify a shared language for modelling the automotive electrical components [17]. The semantics of the elements of the AUTOSAR meta-model is described in several documents and packages named *templates.* For example the *ECU Configuration* describes the configuration process for each single module of AUTOSAR. Because of the complexity of AUTOSAR architecture, modules, communication, and dependencies, OEMs are recommended to use an adequate *tool* support. The tool strategy and details are out of scope of AUTOSAR. However AUTOSAR provides the input for the tool and some *constraints*, listed in the templates as well as the requirements. The constraints are meant to be checked by the tools based on the AUTOSAR meta-model. In this thesis we mention these types of requirements as *TPS.*

According to the AUTOSAR terminology, *SWS* and *TPS* contain "specification items" and "constraints" whilst RS and SRS contain requirements. However, for the purpose of this study, we consider the specification items as requirements as well, since, according to the theory, they have the same meaning. Furthermore, AUTOSAR also says that every text in *SWS* and *TPS* specifications is mandatory to be fulfilled, so semantically, the entire text within a specification could be considered a requirement, even though AUTOSAR does not call it in that way.

For overviewing and fully understanding the AUTOSAR requirement specifications structure and hierarchy we present an example of the requirements traceability through different levels of abstraction on AUTOSAR. Figure 6-3 shows the hierarchy traced from requirements belonging to the basic software module named *CAN Driver.* CAN (Controller Area Network) is a well-known bus standard and message-based protocol used mainly within the automobiles (but also in many other contexts). AUTOSAR propose a set of basic software requirements wrapped in the *AUTOSAR_SWS_CANDriver* specification. In the example, we selected four requirements from the lowest level and we point out the traceability until the project objectives.

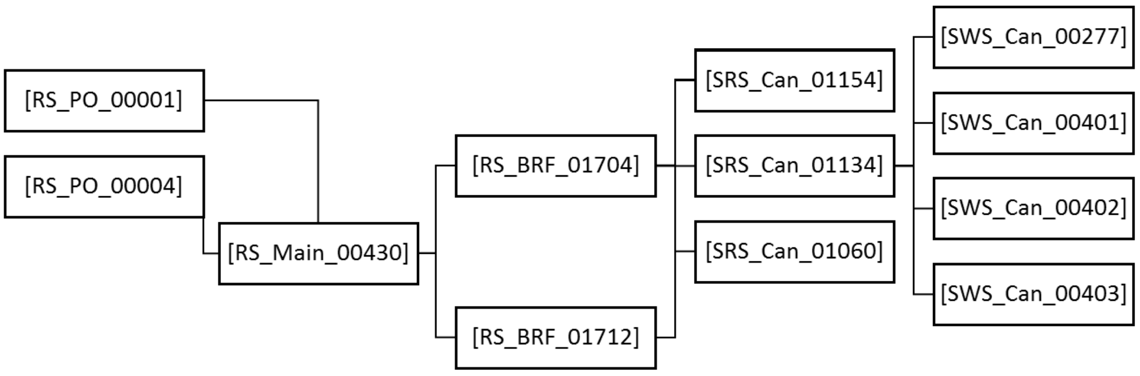


Figure 6-3 Example of AUTOSAR hierarchy and traceability for four SWS_Can specification items

Although here we have mostly one direction with multiple relationships, in AUTOSAR is common to have multiple to multiple relationships between different levels. In the example, this happens from main requirements (*RS_Main*) to the projects objectives (*RS_PO*) where *RS_Main_00430* refer to two different project objectives, and from features (*RS_BRF*) to auxiliary requirements (*SRS*) where *SRS_Can_01154* refers to two features, *RS_BRF_01704* and *RS_BRF_01712.* An extraction of the requirements selected for this example is showed in Table 6-1, with their contents, roles, and references. From the table is clear the abstraction of AUTOSAR and its traceability. From features AUTOSAR starts to generally talk about CAN. In auxiliary requirements AUTOSAR specifies that the support and usage of "multiplexed transmission" is handled by CAN Driver module. Then, in the lowest level, several requirements finally set up this functionality.

Table 6-1 Example of requirements traceability.

Requirement Id	*Hierarchy role*	*Description*	*References*
[SWS_Can_00277]	SWS specifications	The Can module shall allow that the functionality "Multiplexed Transmission" is statically configurable (ON | OFF) at pre-compile time	SRS_Can_01134
[SRS_Can_01134]	Auxiliary requirements	The CAN Driver shall support multiplexed transmission	RS_BRF_01704
[RS_BRF_01704]	Features	AUTOSAR communication shall support the CAN communication bus	RS_Main_00430
[RS_Main_00430]	Main requirements	AUTOSAR shall support established automotive communication standards	RS_PO_00004 RS_PO_00009
[RS_PO_00004]	Project objectives	AUTOSAR shall define an open architecture for automotive software	-

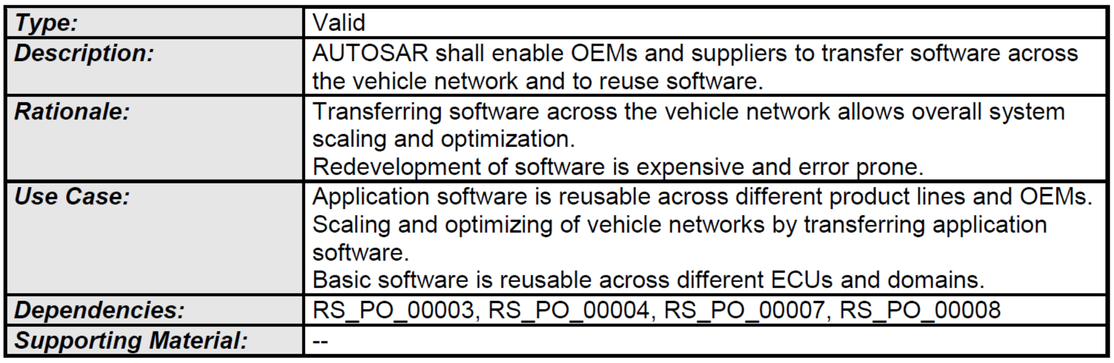
6.1.2. AUTOSAR requirement structure and conventions

For Basic software specifications and design specifications, each requirement has its own, unique headline made of three part:

[DocID_Module_ReqId]

Where *DocID* is the name of the hierarchy role, *Module* is the module of belonging and *ReqId* is the unique number of the requirement. For example *[SWS_Can_00385]* is the requirement with the id equals to 385 in the *SWS_CanDriver module* whilst *[SWS_CanSM_00447]* is the requirement with the id equals to 447 in the *SWS CANStateManager*.

The requirements may have different structures in AUTOSAR. Project objectives, main, features, SRS, and RS requirements are showed as tables with a common structure. Usually the following fields are generally provided: Type, Description, Rationale, Use Case, Dependencies, Supporting Material (figure 6-4). For SWS and TPS documents the requirements exist with different flavours. They may be composed just by text, in a human readable way, or represented by table or even by context diagram, if needed.



Type:	Valid
Description:	AUTOSAR shall enable OEMs and suppliers to transfer software across the vehicle network and to reuse software.
Rationale:	Transferring software across the vehicle network allows overall system scaling and optimization. Redevelopment of software is expensive and error prone.
Use Case:	Application software is reusable across different product lines and OEMs. Scaling and optimizing of vehicle networks by transferring application software. Basic software is reusable across different ECUs and domains.
Dependencies:	RS_PO_00003, RS_PO_00004, RS_PO_00007, RS_PO_00008
Supporting Material:	--

Figure 6-4 Structure of RS_PO_00001 in the Project Objectives

6.2. Results

6.2.1. *Micro* and *macro* analysis

In order to provide suggestions for defining the metrics and the taxonomy of changes needed for performing the measurements (i.e., obtaining preliminary results), we did the *micro* and *macro* analysis.

We proceed first with the *micro* analysis. Figure 6-5 shows a very short extract of the results for *SWS_COM* specifications (see appendix A for details). The table is composed by an entry for each requirement which has been changed. Furthermore, all the columns represent the attributes of the changes, namely the types of change encountered between the two specific versions.

	Mod	Dele	Add
[SWS_Com_00736]		▨	
[SWS_Com_00789]	▨		
[SWS_Com_00393]		▨	
[SWS_com_00845]	▨		
[SWS_com_00863]			▨

Figure 6-5 Extract of the micro analysis

On the other hand, the *macro* analysis is about the history of a small set of specific requirements through all their life in the last major release. Figure 6-6 shows an extract of the outcomes for *SWS_COM specifications*.

	4.0.1	4.0.2	4.0.3	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2
COM696	/	MODIFIED	/	/	/	SPLITTED	/	/
COM260	/	MODIFIED	/	/	/	/	/	DELETED(MERGED)
COM734	/	ADDED	/	MODIFIED	/	MODIFIED	MODIFIED	MODIFIED

Figure 6-6 Extract of the macro analysis.

The complete tables are showed in appendix A. We selected 7 requirements in total and we applied a number of metrics. We use this preliminary study for proposing, suggesting, and testing the metrics in order to understand the most relevant to our purposes.

6.2.2. General Results (*SQ1*, *SQ2*, *SQ3*)

The general results show an overview of the trend and the characteristics of AUTOSAR and they consider both TPS and SWS specifications, which are showed individually in the next Section. The chart in Figure 6-7 points out the size of AUTOSAR.

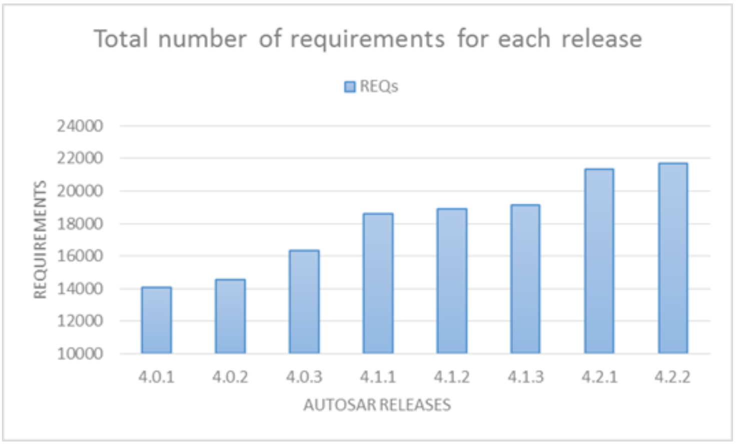


Figure 6-7 Number of requirements for each release in AUTOSAR.

This diagram shows a clear trend of increase in the number of requirements where each new release has more requirements than the previous one. Release 4.2.2 counts about 22.000 requirements whilst 4.0.1 has about 14.000. In the last minor release AUTOSAR counts more than 20.000 requirements.

The Heat map in Table 6-2 gives an idea of the amount of changes in AUTOSAR and its strong evolution. The table shows the total *NoC* by comparing all the releases considered. From the first *minor* release to the last *revision* (i.e., 4.0.1 to 4.2.2) AUTOSAR counts more than 17.000 changes, which means that only 35% of AUTOSAR standard requirements have remained unchanged.

Table 6-2 Heat map of NoC in AUTOSAR

Column1	4.0.2	4.0.3	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2
4.0.1	2399	6278	12543	13151	13699	16503	17195
4.0.2	0	4504	11413	12116	12715	15596	16363
4.0.3	0	0	8302	9227	9902	13025	13908
4.1.1	0	0	0	1626	2803	6533	7820
4.1.2	0	0	0	0	1478	5382	6750
4.1.3	0	0	0	0	0	4262	5786
4.2.1	0	0	0	0	0	0	2002
4.2.2	0	0	0	0	0	0	0

In order to answer *SQ1*, we show two graphs. Figure 6-8 provides an overviews of the *NoC* in AUTOSAR in major release 4.x.

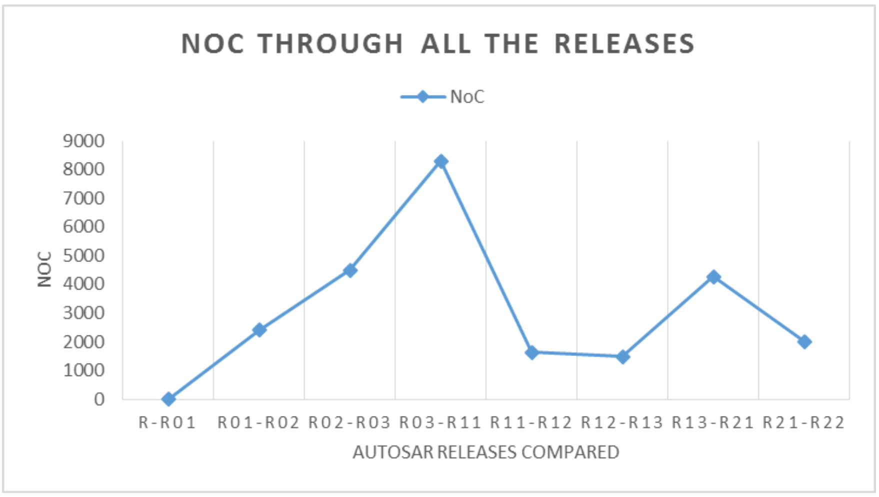
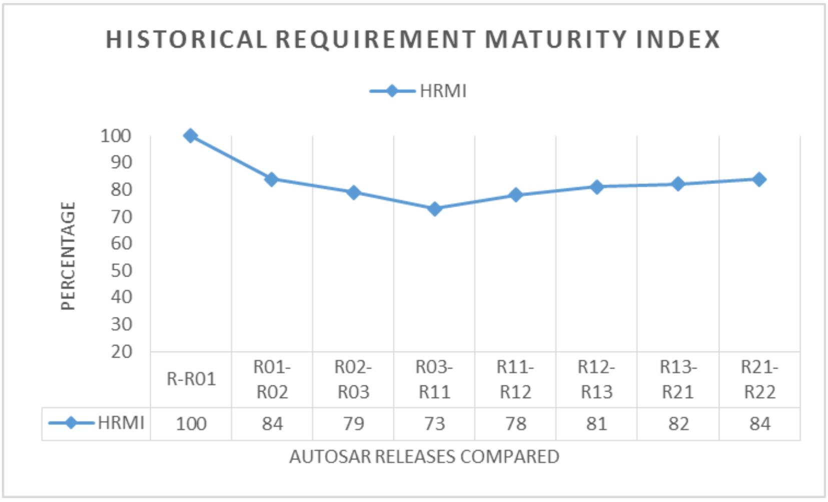


Figure 6-8 NoC in AUTOSAR

R-R01 does not show changes since R01 is the first considered version. In this chart (and in all the following ones) the *NoC* is calculated between following releases, e.g., R01-R02 means the *NoC* between release 4.0.1 and release 4.0.2. Furthermore, when we talk about the behaviour of a version, is always referred to the last release of the pair (e.g. , "R11 has an high *NoC*" means "between R03 and R11 there is an high *NoC*"). In Figure *6-8* we have a general line chart for pointing out where the peaks of *NoC* are placed. The main peak is between R03 and R11. Then, other two significant vertices corresponding to R02-R03 and R13-R21. These three versions are the only ones that register more than 4.000 changes. Furthermore, we notice that two peaks over three are placed on the *minor releases* whilst only one is on a *revision*.

For a deeper analysis on the stability of the releases we applied one of the metric elicited with the GMQ approach. In Figure 6-9 we show the results of the HRMI.



	R-R01	R01-R02	R02-R03	R03-R11	R11-R12	R12-R13	R13-R21	R21-R22
HRMI	100	84	79	73	78	81	82	84

Figure 6-9 Historical requirement maturity index

Note that in the chart, for a better visualization of the results, we calculated the percentage for each *HRMI*, i.e., greater is the value (up to 100%), more the release is considered stable. *HRMI* tries to highlight the historical behaviour of the software, by checking whether the evolution moves to a better stability or not. The chart shows that after a first flexion until the lowest peak in R03-R11, the stability starts to increase linearly, about 2 percentage points for each new release.

The next results concern *SQ2* which is about the most common types of change and their relationships with the releases. With the data collected, we can see that AUTOSAR is not only continuously *changing* but also *growing*. Figure 6-10 presents this behaviour with a pie chart divided in cumulative *NoM*, *NoA* and *NoD*.

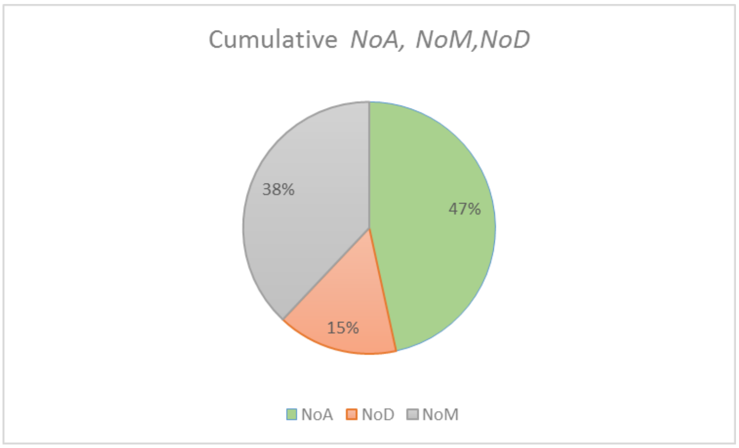


Figure 6-10 Cumulative NoA, NoM, NoD

Cumulative *NoM* (38%) and cumulative *NoA* (47%) are the main types of change while the *NoD* is less significant.

For addressing the types of changes with the respective releases, we provide the chart in figure 6-11. This column chart is used for completing the answer to *SQ2*.

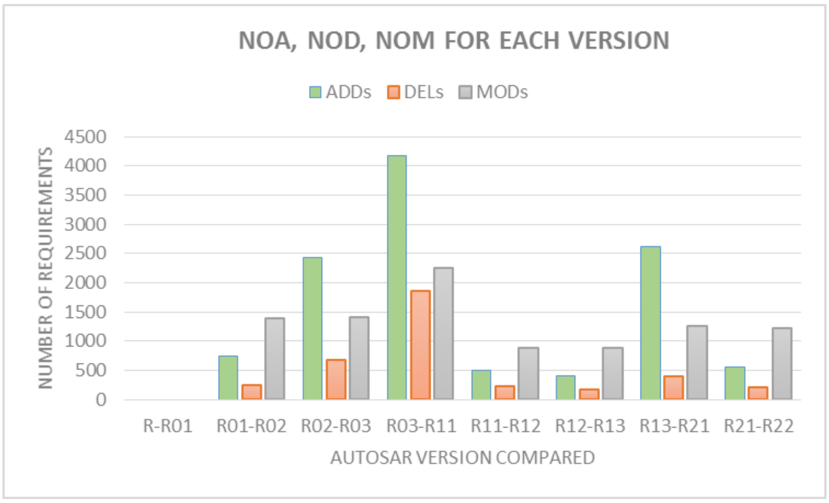


Figure 6-11 NoA, NoD, NoM for each version

The *NoD*, represented by the red column, is always the lowest of the set of three bars for each version. Furthermore, it is usually small (i.e., less than 2-3% of changes over all the requirements), with the only exception for R03-R11. On the other hand, the *NoM*, represented by the grey column, has two trends. A first one, until the first *minor* release, where the line is equal to or greater than 1500 changes, and a second one, where the line is always less than 1500 and quite straightforward on its evolution. Lastly, the number of additions (green columns) is always less than modifications, except for the three peaks discussed above.

In order to answer to the last "general question", i.e., *SQ3*, we measured the evolution by applying the IEEE metric *RMI.* As already described in Section 0, *RMI* aims to measure the stability of each release independently from the history of changes. We applied *RMI* not only to releases but also to AUTOSAR requirement specifications in order to see which ones are most unstable. Figure 6.12 shows the outcomes of *RMI* measured between the releases.

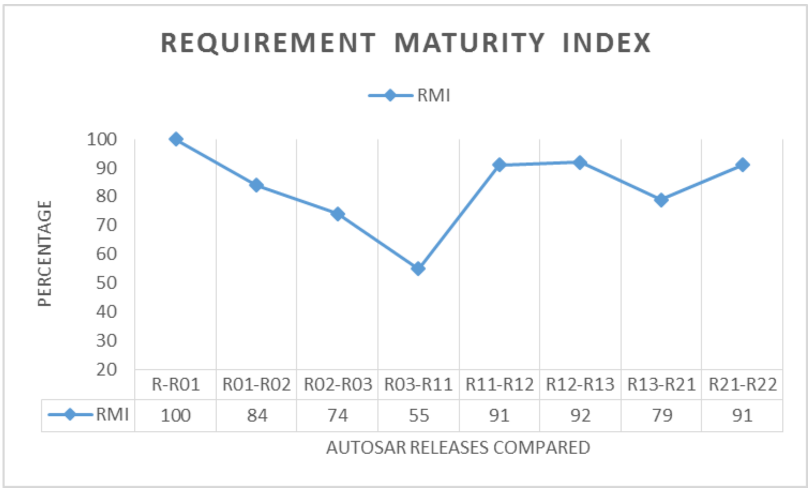


Figure 6-12 Requirement maturity index

We measured *RMI* between releases for both assessing our perceptions about the peaks described for *SQ1* and choosing between which releases checking the unstability of the specifications. From this chart (we show the percentage as done before for *HRMI*), we recognize the three same peaks already mentioned: the releases with more changes are also the ones more unstable. On the other hand, AUTOSAR has a significant stability for releases R11-R12, R12-R13, and R21-R22. All of them have *RMI* greater than 90%. Therefore the *revisions*, with the only exception of R02-R03, are generally less affected by changes. However, also in the *revisions* we observe two trends. The values of *RMI* between R01 and R03 are lower than the latest.

For answering to *SQ3*, we collected the requirements specifications and we ranked them by comparing their *RMI*. By checking the trend showed in Figure 6-12, we chose to measure the unstability for the two *minor* releases (i.e., the changes from R03 to R11,

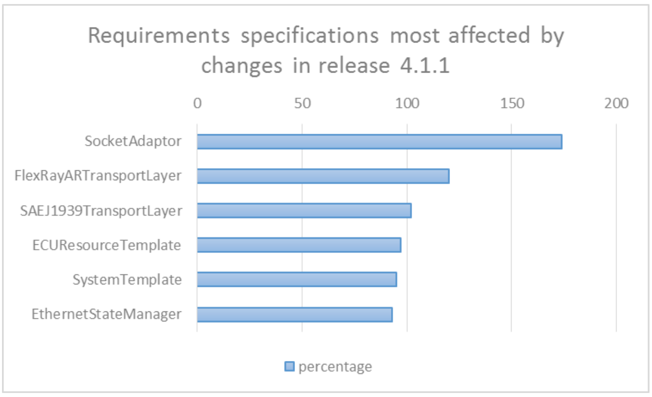


Figure 6-13 Most unstable specifications from R03 to R11

and R13 to R21). Figure 6-13 shows the outcomes for R03-R11.

Note that three of the bars exceeded 100%. This happens because the RMI takes count of all the changes, i.e., modifications, additions and deletions, and the latter can lead the percentage to these values. For example, *SocketAdaptor* counts 191 *NoD*, 165 *NoA*, and 39 *NoM*, thus it has 395 *NoC*. In version 4.1.1 *SocketAdaptor* has 252 requirements, hence the *NoC* is greater than the actual number of requirements. In this *minor* release the top changed documents are composed by two design and four basic software specifications. All of them have a very high *RMI*, close, or greater, to 100%.

Figure 6-14 shows the *RMI* for the most unstable specifications from R13 to R21.

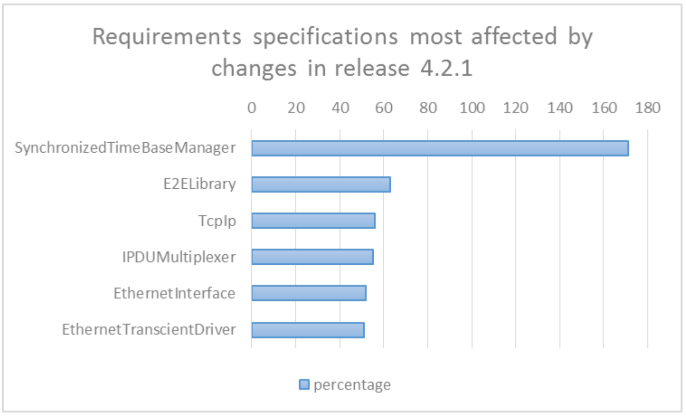


Figure 6-14 Most unstable specifications from R13 to R21

The chart confirm the improving of AUTOSAR specifications in terms of maturity. In the second *minor* release the specifications most unstable are all basic software specifications and for all of them the *RMI* is less than 65%, with the only exception of *SynchronizedTimeBaseManager*. Furthermore, there are no documents in common in the two charts. However, we observe that the modules of AUTOSAR mostly recurrent in the charts concern the communication intra or iter ECUs: *SocketAdaptor*, *Flex Ray*, *Ethernet*, *TcpIp*, *SAE1939* are specifications, or protocols, related to the transport layer.

6.2.3. *Template* and *Basic Software Requirements* results (*SQ4*, *SQ5*)

Using several charts and the requirements table, we looked for trends and relations among consecutive releases. The analysis in this section starts with a general view of the evolution, individually for *TPS* and *SWS* and then moves down to the requirements level. The results maintain the distinction between *TPS* and *SWS* for showing their different scopes and behaviours as well as for answering to the last two research questions, i.e., *SQ4* and *SQ5*.

In order to provide an answer to *SQ4* which concern the behaviour of categories of requirements, we consider *TPS* and *SWS* as two categories: basic software requirements and design requirements. The constraints are included by the latter. Figure 6-15 shows the lines chart with the two main types of change for *TPS* in AUTOSAR.

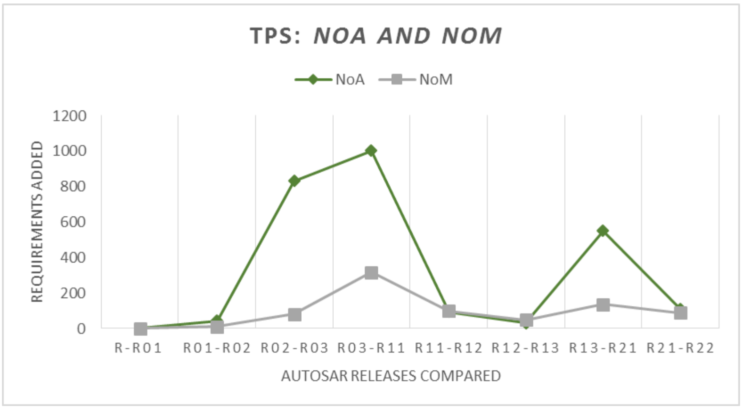


Figure 6-15 NoA and NoM in design requirements

TPS has always a low *NoM* even though it presents many additions. However, the *NoA* is more significant than the *NoM* only for the two *minor* releases and for *revision* R03. On the other hand, *SWS* documents contains both a large number of *NoA* and *NoM*, as showed in Figure 6-16.

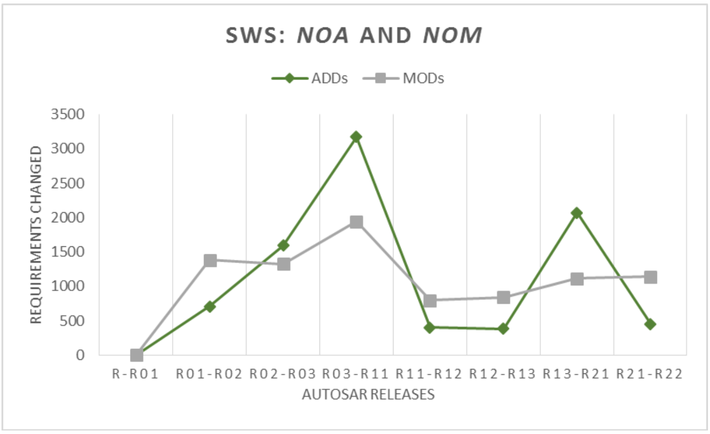


Figure 6-16 NoA and NoM in basic software requirements

A shared behaviour for the two categories is the distribution of the *NoA*. Also for *SWS* the number of additions is more significant than the number of modifications only for the two minor releases and for R03. However the impact of *NoA* is different in the two cases. *TPS* in the previous major releases does not have almost any requirements, but just a small set of constraints, whilst *SWS* had already a large set of requirements. This is clarified in figure 6-17.

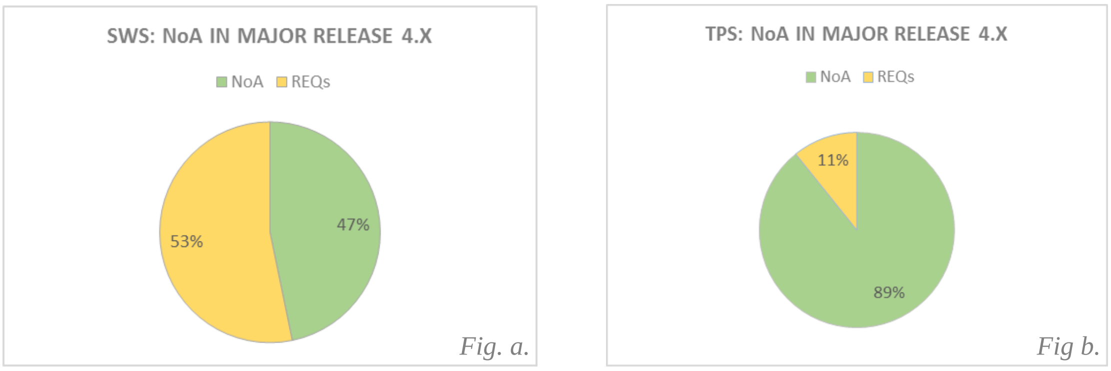


Figure 6-17 Cumulative NoA in the last AUTOSAR major release for SWS and TPS

Fig. *a* shows that although SWS counts a great number of additions, their impact is less significant if compared with the number of already existent requirements. Fig. *b* tells the opposite for *TPS* and highlights the importance of this major release for the elicitation of design requirements.

We continue our investigation on the most stable and unstable categories by using two other metrics, namely *lifecycle* and *frequency*. Since these metrics are calculated on the same data of the last two, i.e., *sequence* and *accuracy*, we made a Table with the outcomes of all the measurements, divided by categories, for answering to both *SQ4* and *SQ5*.

Table 6-3 shows the evolution from the requirements point of view, using the proposed metrics. We gathered several values as averages between all the requirements, and we represented these averages with a column in the table. The last column, named *effective sequence* is the average of all the sequences which have at least one change in the last major release.

Table 6-3 Metrics applied to design requirements (TPS) and software requirements (SWS)

Specifications	*Frequency*	*Sequence*	*Lifecycle*	*Accuracy*	*Sequence (effective)*
GENERAL	0.26	0.83	6.90	0.84	1.15
TPS	0.30	1.03	5.15	0.90	1.11
SWS	0.25	0.80	7.14	0.82	1.18

By observing the table, we make the following considerations:

- The *frequency* of changes is very similar between *design* requirements and *software* requirements.
- The average of the *sequence* of changes, namely the maximum number of changes that happens consequently for a requirement through its history, is higher in *TPS*, about one change, than in *SWS*, less than one change.

- The *SWS* specifications have a “longer” life than *TPS.* The majority of them was introduced in a major release before the last one whilst *TPS* specifications are mostly added in R402 and R403
- The *accuracy* points out that the most reliable specifications are within the *TPS* documents.
- The *effective sequence* contradicts the *sequence* of changes, which suggested a bigger number of changes in *SWS* specifications, and agrees with the *accuracy*, by giving the lowest value to *TPS.*

The *sequence* is calculated for each requirement, even for those that do not change through all the releases. This may affect the results and lead them to a value less than one. If we consider the *effective sequence*, the results change significantly and *SWS* present now a bigger value. Furthermore, the *sequence* takes count of all the types of change, hence, also the additions are considered. In *TPS*, as described above, almost all the requirements are added in the last major release, and all of them are counted as changes in the sequence, which means a further increment of a unit.

The last research question *SQ5* relies on the changes analysis. We looked for relations between changes and for behaviours of requirements just changed in the immediate next release by using the metrics *lifecycle* and *accuracy*. We found a relation between these measurements in the AUTOSAR requirements and we calculated two different *accuracies*, one for the requirements with a *lifecycle* equal to 9 (i.e., the ones that exist from another major release) and the requirements with a lifecycle less than 9 and greater than 2 (i.e., until release 4.1.3.). The results are presented in the following Table.

Table 6-4 Accuracy measured with different Lifecycle (LC)

Specifica-tions	*Accuracy (LC==9)*	*Accuracy (3<LC<9)*
SWS	0.78	0.85
TPS	0.80	0.91

This Table points out the reliable nature of additions (LC<9) both for *SWS* and *TPS*, furthermore, it shows that modifications usually brings other changes to the same requirements.

For assessing this observation, we described a further analysis of additions and modifications. According to our general results, the main peak of change, both for *SWS* and *TPS*, is placed on the first *minor* release. Hence, we conducted a deeper research on the behaviour of the requirements on that version, and we look for any trend or relations. The pie chart in Figure 6-18 shows the percentage of requirements which are either added or modified in the *minor* release considered and immediately changed in the following *revisions* (i.e., R12 and R13).

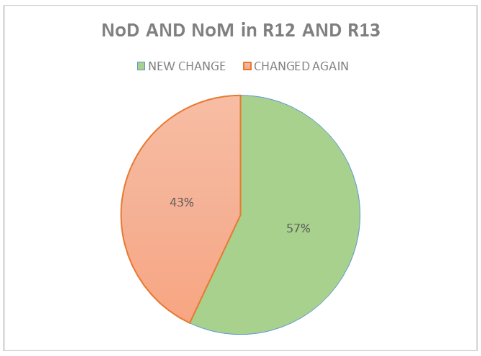


Figure 6-18 consequently changes in R12 and R13 after R11

The Figure presents the cumulative number of changes registered in *revisions* R12 and R13. These changes are divided in requirements already changed in the *minor* release, the red side, and novel changes, the green side. Here, we are considering only *NoD* and *NoM*. The 43% of requirements deleted or modified were already changed in the previous *minor* release.

Although this can be read as a weakness of AUTOSAR or as poor accuracy in modifying or adding requirements, we need to take into account the amount of changes in the *minor* release, around 8.000, and in the *revisions*, around 1500 for each. Therefore, we have a significant number of requirements changed multiple times, but a small number of changes in general. In order to clarify this concept, in the next two charts, in Figure 6-19, we looked for the same types of change, but we compared them with the total number of requirements changed in *minor* release R11.

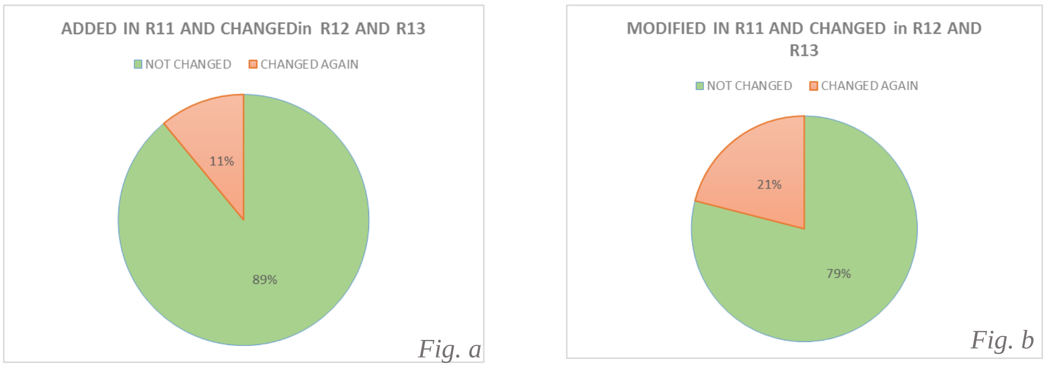


Figure 6-19 Behaviour of changes in the minor release and in the subsequent revisions

Figure *a* shows the percentage of requirements added between R03 and R11 and immediately changed in the next two *revisions*. Figure *b* does the same with modifications instead of additions. We observe that requirements added or modified between R03 and R11 are then modified or deleted in the next two releases for respectively the 11% and the 21%. The majority of requirements is not changed again, this can be read as an index of *accuracy*. Furthermore it is important to notice one time again that the requirement added are usually more stable (only 11% of subsequently changes) than the one modified (21%) in the *minor* release.

6.3. Validation

This section provides the results of the validation of the metrics and assesses the quality of the measurements performed in the previous section.

The validation has been conducted by following the schema described in the research methodology. We involved six experts of AUTOSAR at Volvo Car Group, and we provide personalized surveys with ten questions, four generals and six specifics. All the questions are attached in appendix B. Since the experts could not have an accurate knowledge about the general evolution of the entire set of basic software requirements and templates, we gave the possibility to answer with the option “I do not know”. The pie chart in figure 6-20 shows the results of the survey.

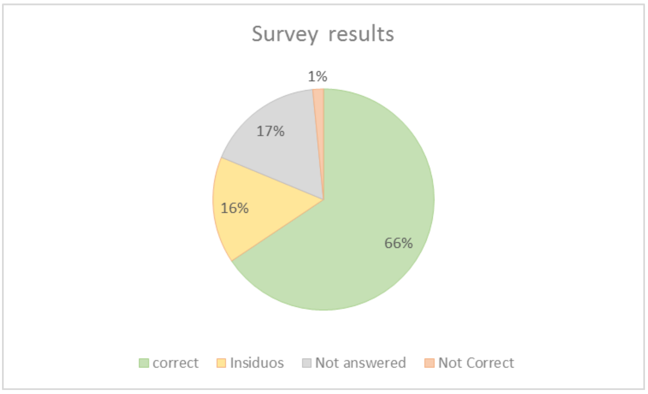


Figure 6-20 Survey results

All the questions were composed by multiple options with one correct, one considered “insidious” i.e., close to be correct, and one considered wrong according to our results. We follow this way in order to quantitative analyse the results and to easily address their expectations with our results. Only one answer was allowed and only one option was actually considered correct, except for question n.4 and n.7 where the possible choices were multiple.

The experts answered to the majority of the questions and all the answers that did not meet our results, except for one, came from the “insidious” option. Two thirds of the answers meet our measurements.

The first four questions were generals and relied on our *SQ1*, *SQ2*, and *SQ3*. In figure 6-21, the graph shows the answers of the experts for these questions.

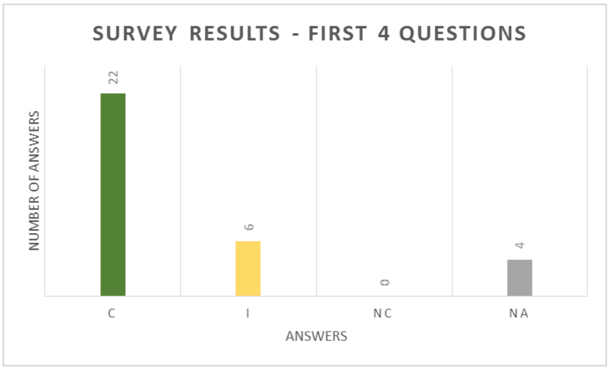


Figure 6-21 Survey results about the first four questions

C means correct (with the green column), I insidious (orange column), NC not corresponding to our measurements (red column), NA not answered (grey column).

This chart indicates a good awareness of the expert about the general evolution of AUTOSAR. Their expectations meet our findings for the majority of the topics such as the most unstable releases, the general behaviours of AUTOSAR and the most common types of changes. Furthermore, question 4, which was directly associated with *SQ3* assess the efficiency of the *NoC* considered and the reliability of our results. We asked to choose the AUTOSAR requirement specifications most affected by the evolution from version 4.0.3 to the last one. We provided a set of answers composed by seven AUTOSAR documents: three with a significant *NoC*, one with a medium *NoC* and three with a low *NoC*. Figure 6.22 shows the results.

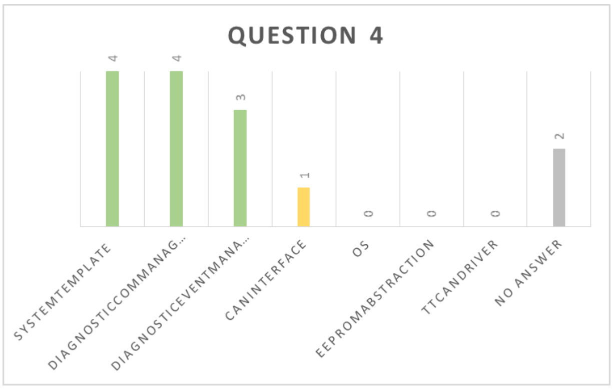


Figure 6-22 Results for question 4

Also in this case the expectations of the experts were aligned with our findings.

In the second part of the survey, we focused on two requirements specifications for each participant. We personalized the questions by maintaining a common structure between all the surveys, in order to analyse the data. Figure 6.23 shows the results from question 4 to question 10.

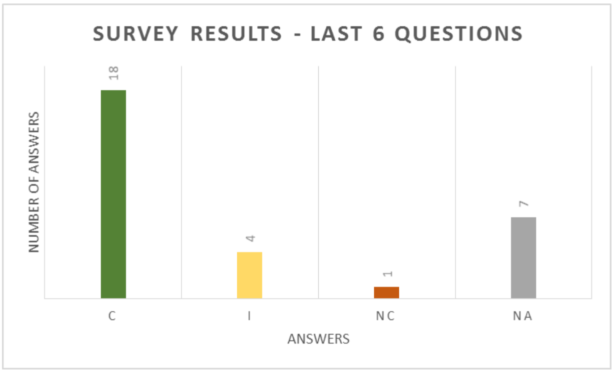


Figure 6-23 Survey results about the last 6 questions

Again the answers meet the results and assess the precision of metrics used for analysing the requirements within a single set. The questions were based on requirements specifications about both *SWS* and *TPS*, and concerned the most common types of change in a specific document, the behaviours of set of requirements across all the releases and the stability of requirements categories such as basic software and design.

The documents used for the validation are *TPS_ECUConfiguration*, *TPS_SystemTemplate*, *SWS_Com*, *SWS_ComManager*, *SWS_FLexRayNetworkManagement*, and *SWS_CanManage*, see appendix A for details.

The validation represents an assessment of the findings obtained with the proposed metrics. The outcomes show a correspondence between the experts' expectations and our findings. This means that our set of metrics can indeed be used for measuring the evolution of system requirements and are therefore applicable for monitoring the changes and for facilitating their understanding.

7. Discussion

In this section we discuss the results we obtained. Firstly, we describe interesting findings observed during the analysis and the measurements of AUTOSAR such as inconsistencies and considerations on the results. Then, we answer the research questions described in the methodology and we discuss about recommendations and validations.

7.1. Issues and inconsistencies in the AUTOSAR specifications document

We came across several problems during the application of the metrics on the AUTOSAR specifications due to its inconsistencies, especially for the first releases. The high number of changes in the form, the structure, and the conventions used in AUTOSAR makes it hard to point out if a requirement has been affected by an actual change or not. In AUTOSAR the *ReqId* changed its form either for three or four times in the last 8 releases, depending on the considered document. For example, the *Autosar_SWS_ADCDriver* specifications 432 was named *ADC432* in releases 4.0.1 and 4.0.2, then it was identified by *[ADC432]* in release 4.0.3 and finally stabilized in the following releases with the form *[SWS_Adc_00432]*. Furthermore not for all the specifications the *ReqId* is changed in the same way and even inside the same AUTOSAR documents there were different versions of the *reqId*.

Another example is showed in figure 7-1. In this case we use the requirements specification named *AUTOSAR_SWS_IFXLibrary* which specifies the functionality, API and the configuration of the AUTOSAR libraries. Here the AUTOSAR inconsistencies are clearly visible. In release 4.0.1 (Fig. a), we can see two requirement: *IFX002* represented by a table and *IFX005* which is inside the description of the previous one. In release 4.0.3 (Fig. b) the *reqId* form of *IFX002* is changed with the addition of the square brackets. Further, there is also the special character "[", used for wrapping the content of a requirement. However the changes are applied only to that requirement, not to the internal one, which has changed the name to *IFX003* instead. Lastly in release 4.2.1(Fig. c) the two requirements are completely separated: one with table form and one with text form. The name is modified again and the special character is used for both of them. In all three figures, the effective content of requirement *[SWS_Ifx_00003]* has not been changed.

As we can see from the presented findings, even though AUTOSAR is one of the most known standard for automotive systems, there is still a number of inconsistencies in the last major release. However, we noticed a significant improvement in the last *minor* versions, were all the specifications started to follow a common structure. Since in AUTOSAR there are several thousands of specifications and it was not feasible to proceed manually, a tool has been developed with a particular attention for finding and interpreting these inconsistencies.

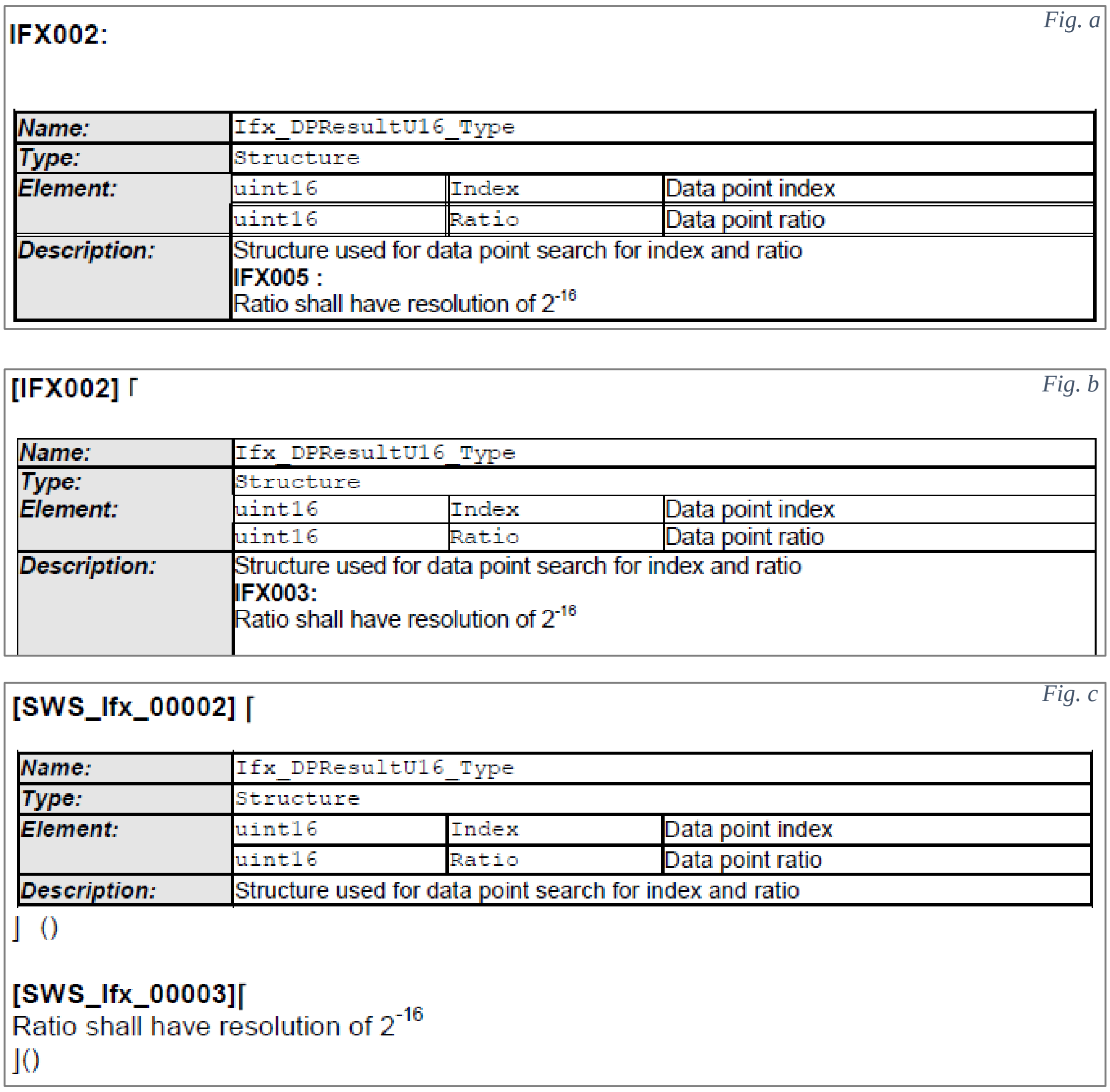


Fig. a

IFX002:

Name:	`Ifx_DPResultU16_Type`		
Type:	`Structure`		
Element:	`uint16`	`Index`	Data point index
	`uint16`	`Ratio`	Data point ratio
Description:	Structure used for data point search for index and ratio **IFX005 :** Ratio shall have resolution of 2^{-16}		

Fig. b

[IFX002] ⌈

Name:	`Ifx_DPResultU16_Type`		
Type:	`Structure`		
Element:	`uint16`	`Index`	Data point index
	`uint16`	`Ratio`	Data point ratio
Description:	Structure used for data point search for index and ratio **IFX003:** Ratio shall have resolution of 2^{-16}		

Fig. c

[SWS_Ifx_00002] ⌈

Name:	`Ifx_DPResultU16_Type`		
Type:	`Structure`		
Element:	`uint16`	`Index`	Data point index
	`uint16`	`Ratio`	Data point ratio
Description:	Structure used for data point search for index and ratio		

⌋ ()

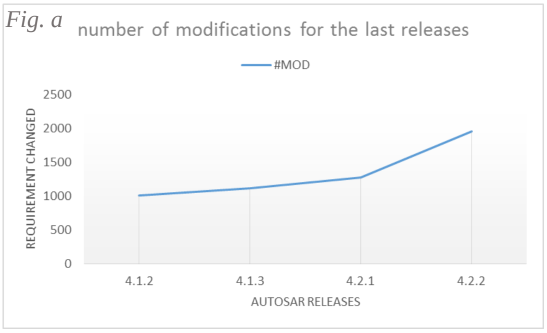
[SWS_Ifx_00003]⌈
Ratio shall have resolution of 2^{-16}
⌋()

Figure 7-1 Requirement taken from different releases of AUTOSAR_SWS_IFXLibrary.

7.2. Considerations on the results

As described in the metrics definition section, the modifications can be of different types and need the judgment of the researchers for being considered as changes or not. During our analysis we found an unexpected behaviour of the *NoM metric* so we decided to investigate it further.

By applying our measurements, we discovered a trend among the number of modifications in the last releases. The *NoC* was straightforward until the last release, when it showed discontinuity, presented in figure 7-2 *a*, in a form of a strong increase. Therefore we decided to analyse, using the tool we developed, different types of modification from 4.2.1 to 4.2.2 in order to understand the cause of this discontinuity. We discovered that AUTOSAR has renamed one requirement specification from *AUTOSAR_SWS_DevelopmentErrorTracer* to *AUTOSAR_SWS_DefaultErrorTracer*. As a consequence, all the requirements which contained the word “development” (in that context, but not necessarily in the same specifications) have been renamed as well. We collected the data again, this time by excluding this common change, and the new results clearly show that the straightforwardness is preserved through all the last releases.(Fig *b*)



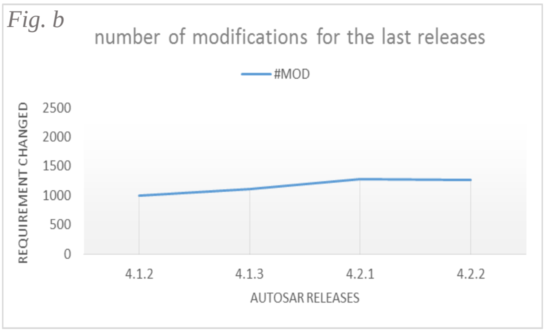


Figure 7-2 NoM for the last releases.

The 27% of modifications were related to this change. We decided to perform our analysis without considering it, since it was not an effective change to the semantic of the requirements.

7.3. Research questions

Our results confirm that AUTOSAR clearly distinguishes the roles of the releases by adding new features and functionalities, under the form of requirements, in the *minor releases*, and by providing bug fixes and refinements in the *revisions*. Furthermore, we assessed that AUTOSAR requirements evolution is characterized by a significant *NoC*, mainly composed by additions and modifications, which implies a substantial enlargement of the standard. Lastly, the metrics stated that AUTOSAR achieved a considerable maturity in the last versions of the major release by decreasing the total *NoC* and increasing the *accuracy* and the *HRMI*. We provide separated answers for each research question described in the research methodology.

SQ1: Which releases of a software system are most stable and which one change most?

By reading our results and analysis, we can observe at least two trends:

I. The *minor* releases are prone to more changes than the *revisions*

II. The first releases (from 4.0.1 to 4.1.1) are more unstable than the following ones.

The AUTOSAR major release 4.x has been significant in terms of *evolution*. With a strong increment of features, stated by the heat map in *Table 6-2* and the chart in 6-7, AUTOSAR experienced a significant number of changes, both additions and modifications. According to the results, the majority of changes come from the first releases. This is caused by two factors. Firstly, between R01 and R11, AUTOSAR changed the structure, the form, and the conventions of the requirements document for several times as described in 7.1, forcing a complete review of the requirements both in terms of content and appearance. As a consequence, many requirements were changed in different ways. Secondly, AUTOSAR has introduced a huge number of features, such as the possibility to switch off the ECUs when they are not used, the automated cruise control and many others. These features not only require the additions of new requirements, specifications or packages, but also the modifications of many already existent items for changing across the dependencies, APIs etc. However, after R11, the AUTOSAR architecture acquired stability and maturity with a less number of changes testified by the *NoC* in Figure 6-8 and the *HRMI* chart in Figure 6-9.

SQ2: Which types of changes are more common and how they are connected with the releases?

The most common types of changes are *additions* and *modifications.* AUTOSAR had a clear expansion during this major release, with a small number of deletions, except for the first *minor* release. We provide these measurements in the charts of the general results section, in Figure 6-10 and 6-11. Even though the total number of additions is alike to the total number of modifications, their behaviour through the releases is completely different. The additions are usually extremely low, with only three high peaks that correspond to two *minor* releases and to R03. The modifications are more straightforward with similar values through all the releases and without a significant peak. In conclusion we can assert that in the *minor* releases (or in the first ones that are more unstable) AUTOSAR adds a huge number of requirements which likely correspond to new features, while in all the *revisions* the main task was to fix bugs and inconsistencies without expands the architecture and the functionalities.

SQ3: Which requirements specifications are most unstable?

As observed in the results section, in figures 6-13 and 6-14, the most unstable specifications usually changes taking in considerations different pairs of releases. However we observed that the majority of unstable specifications are related to ECU communication. *FlexRay, Sae j1939, Ethernet, TcpIp etc.* are network protocols which specify communication buses. The communication between ECUs is one important aspect of the basic software system. The introduction of new features which require new functionalities for the ECU communication (such as a certain speed or a precise way to communicate data) bring many changes to the related modules.

SQ4: Which categories of requirements are most stable and which ones are mostly affected by the evolution of the system?

In the AUTOSAR case study we analysed the requirements by diving them in *SWS,* software functional requirements, and *TPS,* design requirements. For both of them AUTOSAR provides several packages with many specifications documents. The majority of the requirements analysed were from *SWS.* According to the results in Figures 6-15, 6-16, 6-17 and in Table 6-3, we can make the following observation:

Design requirements and basic software requirements are both affected by the evolution of the system, but their behaviour through all the releases is completely different. The TPS requirements have been introduced later and are stable, the SWS have been introduced earlier and are prone to modifications.

For *SWS* and *TPS,* the frequencies of changes are pretty similar, however, their requirements history followed two different ways. The most of the design requirements have been added in the last major release. The *unstability* of the *TPS* documents and specifications is given by the addition, which is considered a change as well as the modification. Apart from that, the *TPS* items and constraints are actually stable and accurate since only few of them are changed after their additions. However we need to consider an addition as an actual change since they imply effort for being adopted by the engineers in OEMs as well as the modifications. On the other hand, the *SWS* specifications have a longer lifecycle, even if there are many additions, and are more affected by changes, especially by modifications.

SQ5: Which types of requirements are more characterized by subsequent changes and which one are more reliable?

According to our study and results, showed in Table 6-4 and Figures 6-18 and 6-19, we can make two further observations:

I. The requirement changed in the *revisions* are usually derived from a change in a *minor* releases.
II. A requirement added in the last major release is likely not changed again in the following versions.

We found these behaviours by using the measurements and by comparing different groups of requirements. In AUTOSAR the requirements added in the last major release are usually stable and are characterized by high accuracy. On the other hand we noticed that a great number of changes in the *revisions* are modifications or deletions of corresponding relevant changes in the *minor* releases. Generally, modifications may lead to other changes in the subsequent releases.

7.4. Recommendations

The results and the discussion are based on the case study of AUTOSAR conducted at Volvo Car Group. Although the objective of this study was to achieve general perspectives and observations on requirements evolution, we had to develop the taxonomy of modifications and the *NoC* according to the automotive context. For this reason, in order to apply the proposed metrics in other domains, we believe that the taxonomy of changes should be refined through the analysis of the analysed industrial context, as we explained in our research methodology. We believe that the rest of the application and interpretation of the metrics is applicable to other contexts as well.

7.5. Threats to validity

In order to give a better understanding of this study and of its outcomes, we discuss the threats to validity, according to Cook and Campbell [24], from four perspectives: Internal, external, construct and conclusion validity.

Internal validity

Internal validity concerns the accuracy of the results. In our study, this means that the measurements and the outcomes should not meet randomly. There are three threats to internal validity that should be considered. The first one was the fact that the measurement process was performed by developing and using a tool for comparing different specifications in different releases. In order to validate the outcomes, we tested the tool several times by exporting simple comparison between single specifications and by comparing them either with a manually revision or with the *micro* and *macro* outcomes.

The second threat was related to what is considered as requirements in AUTOSAR specifications. Not only the specification items and constraints described in this thesis can be considered as requirements, but also everything written in AUTOSAR specifications can be considered as a requirement as it is mandatory to be followed when developing AUTOASR compliant automotive systems. However, AUTOSAR clearly defined as specification items (and constraints) all the important statements contained in the test. By reading and analysing AUTOSAR specifications we noticed that all the most significant information are specified as specification items whilst the remaining parts are usually examples, rationales, and figures for a better understanding of the actual specification items. For this reason, we believe that our results are still valid since they captured most of the main requirements, and they take into consideration changes in the actual content of the requirements.

The third threat concerns the considered releases, i.e. we analyse the evolution for only the major release 4.x. As already explained in the research methodology, this has been decided according to the needs of our industrial partner Volvo Car Group. Furthermore, by discussing this scope of releases with an AUTOSAR expert in several study sessions, we agreed to consider only major release 4.x since it is the newest major release of AUTOSAR that has been used for many years (since 2009) and it contains most features and requirements that are currently used by the majority of car OEMs. It also contains a number of minor releases and revisions which made our evolution analysis possible. Furthermore, the objective of this thesis is to perform measurements in order to facilitate the updates of large software systems based on the changes in the requirements. Therefore it is not considered significant to analyse very old aspects of evolution.

External validity

External validity concerns generalization of results. An external threat to this study was concerned with its aim to make general considerations in contraposition with the outcomes, which are just applied to a single case study. Although we cannot claim a generalization without testing the proposed metrics on others case studies, we believe that these metrics would produce valid and good results also in other contexts. The majority of the metrics are adopted by conducting a literature review on existing studies performed in different contexts and with their own validations.

Construct validity

Construct validity concerns the mismatch between the theory and observations. This study analysed the accuracy of the measures in the context of its application. We ensured this accuracy by relying on the GQM approach, which guarantees an accurate selection of the metrics.

Conclusion validity

Conclusion validity is the degree to which conclusions and relationships in our data are reasonable. In our case study the conclusions were derived by applying the metrics on the data collected, obtaining the results and finally comparing them with the expectations of the experts, as described in section 6.3. The conclusion was that the results could describe properly the characteristics of AUTOSAR.

8. Conclusion

In this paper, we present and evaluate the set of metrics that can be used for analysing the evolution of requirement in large software systems. Most of the metrics are based on the already existing metrics such as the ones counting the number of changes in the requirements and their stability. For counting the number of modified requirements (NoM), we also defined the taxonomy of modifications in order to exclude the changes that do not have any semantic impact on the system (e.g., change of requirement IDs or grammar fixes). Finally, we propose a new metric named *accuracy* that complements the other metrics in assessing the reliability of requirement documents.

We calculate and validate the proposed metrics in a case study at Volvo Car Group using AUTOSAR requirements evolution as a unit of analysis. We used these metrics to assess the impact of AUTOSAR requirements changes on the AUTOSAR based automotive systems. This in turn helped us to provide the answer to our main research question of how to efficiently measure the evolution of system requirements in large software systems in order to facilitate their updates with new features.

Our results show that the requirements evolution should be analysed and measured from different points of view, i.e., from release, specification, and requirement point of view. From the first two it is possible to observe and understand the nature and the size of the evolution, whilst from the third it is possible to identify and measure trends and specific behaviours between different categories or types of requirements.

We also concluded that by applying the set of proposed metrics to different version of a large software system, it is possible to assess the size of the evolution and list the specifications affected by this evolution. This knowledge can help organizations responsible for managing large software systems in understanding the area of the system where most effort is needed for its update and also to make strategic decision based on that, e.g., which features shall be supported in new versions of the system. Furthermore, efficiently measuring and comparing requirements specifications makes the learning process faster for the engineers involved in the system evolution management and can help them to adopt new features in a more efficient way.

We identified several area of interests for potential further work. Since the requirement evolutions is nowadays a challenging process both for the companies and researchers, this thesis is considered as a pilot work in this field. There are several interesting ways for improving the set of metrics we proposed or for developing a method for their industrial application. Some areas we recommend to be further explored are listed below:

I. Validate and test the metrics, especially the *accuracy* metric as a newly proposed metric, in other industrial contexts or products.

II. Analyse the *user* point of you. Measure the effort needed for adopting new features in the new versions of the system and understanding which requirements are critical i.e., their changes are considered challenging to be fulfilled.

III. Developing a method, based on our experience, for defining step by step approach on how to efficiently study the evolution of system requirement in an industrial context based on the set of metrics.

References

[1] S.D.P. Harker, K.D. Eason, and J.E. Dobson. “The change and evolution of requirements as a challenge to the practice of software engineering”. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 266–272, San Diego, California, USA, January. IEEE Computer Society Press. (1993).

[2] Stuart Anderson and Massimo Felici. “Controlling requirements evolution: An avionics case study”. In *Proceedings of SAFECOMP 2000, 19th International Conference on Computer Safety, Reliability and Security*, LNCS 1943, pages 361–370, Rotterdam, The Netherlands, Springer-Verlag (2000).

[3] AUTOSAR. *Automotive Open system architecture*. URL: www.autosar.org, (2003).

[4] IEEE. Std 982.1 - IEEE Standard Dictionary of Measures to Produce Reliable Software, 1988.

[5] Anderson S. and Felici M., “Requirements evolution from process to product oriented management” , in *Proceedings 3rd PROFES* 2001

[6] Shi L et al. “Learning from Evolution History to Predict Future Requirement Changes”, in: *Proceedings of RE* 2013, pp. 135–144.

[7] Sommerville. “Software Requirements”. In *Software Engineering Seventh Edition* (2004).

[8] B. Manfred, E. Simmons. “Characteristics of a good Requirement”. In *Software & Systems Requirements Engineering* (2009), pp.9-15.

[9] Software Engineering Standards Committee. “IEEE Recommended Practice for Software Requirements Specifications”. *IEEE Std 830, (1998).*

[10] Hull E, Jackson K, Dick J. “Requirements Engineering”, 3rd ed. 2011

[11] Davis, N. Nurmuliani; P. Sooyong; D. Zowghi. “Requirements Change: What’s the Alternative?”. In *Proc. 32nd COMPSAC*, pp. 635-638, (2008).

[12] Wang, H.; Li, J.; Wang, Q. and Yang, Y. "Quantitative analysis of requirements evolution across multiple versions of an industrial software product", in *Proceedings 17th APSEC,* pp. 43-49, 2010.

[13] N. Nurmuliani, D. Zowghi, and S. Fowell, "Analysis of Requirements Volatility during Software Development Life Cycle," in *Australian Software Engineering Conference*, 2004, p. 28.

[14] G. Stark, A. Skillicorn, and R. Smeele, "A micro and macro based examination of the effects of requirements changes on aerospace software maintenance," *in Aerospace Conference*, 1998. Proceedings.,IEEE, 1998, pp. 165-172 vol.4.

[15] D. Durisic, M. Staron, M. Tichy, “ARCA - Automated Analysis of AUTOSAR Meta-model Changes”, *in International Workshop on Modelling in Software Engineering* (2015).

[16] D Durisic, M Staron, M Tichy, J Hansson, “Evolution of Long-Term Industrial Meta-Models--An Automotive Case Study of AUTOSAR Software Engineering and Advanced Applications (SEAA)” , *40th EUROMICRO,* 2014

[17] Broy M., Kruger I. H., Pretschner A., and Salzmann C., “Engineering Automotive

Software” *Proceedings of the IEEE | Vol. 95, No. 2,* February 2007

[18] V. Vaishnavi and W. Kuechler. “Design science research Methods and Patterns”. *Second Edition. CRC Press, Boca Raton,* FL. 2015

[19] C. Wohlin. “Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering”. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering.* ACM (2014).

[20] WinMerge Development Team, Version: WinMerge 2.14.0, *http://winmerge.org/* , 2013

[21] V. Basili, G. Caldiera, and H. Rombach. “The Goal Question Metric Approach. “, *Encyclopedia of Software Engineering, Wiley,* 1994.

[22] P. Runeson and M. Host. “Guidelines for conducting and reporting case study research in software engineering”. *Empirical Software Engineering,* pp. 131-164, 2009.

[23] S. Balaji, M.S. Murugaiyan. “Waterfall vs V-Model vs Agile: A comparative Study on SDLC”. In *International Journal of Information Technology and Business Management,* Vol.2 No.1, (2012).

[24] T. Cook and D. Campbell. “Quasi-Experimentation: Design & Analysis Issues for Field Settings”. Houghton Mifflin, 1979.

Appendix A

Micro Results

1) Documents considered : *AUTOSAR SWS COM*
   - Versions compared: 4.2.1 – 4.2.2
   - Legend:
     - Orange: light modification
     - Blue: already existent but without belonging to any requirements.
   - Total number of changes: 29 (Relevant: 19):
     - Modified: 19 (10)
     - Merged: 1
     - Deleted: 3
     - Added: 6

	Mod	Mer	Split	Dele	Add
[SWS_Com_00675]	🟩				
[SWS_Com_00734]	🟩				
[SWS_Com_00762]	🟩				
[SWS_Com_00135]	🟧				
[SWS_Com_00742]	🟧				
[SWS_Com_00743]	🟧				
[SWS_Com_00770]	🟧				
[SWS_Com_00736]				🟩	
[SWS_Com_00789]	🟩				
[SWS_Com_00393]				🟩	
[SWS_Com_00486]	🟧				
[SWS_com_00845]	🟩				
[SWS_com_00863]					🟩
[SWS_Com_00838]	🟧				
[SWS_Com_00578]	🟧				
[SWS_Com_00864]					🟩
[SWS_Com_00865]					🟦
[SWS_Com_00348]	🟩				
[SWS_Com_00861]					🟩
[SWS_Com_00858]	🟩				
[SWS_Com_00862]					🟩
[SWS_Com_00001]	🟧				
[SWS_Com_00260]		🟩		🟩	
[SWS_Com_00475]	🟩	🟩			
[SWS_Com_00670]	🟩				
[SWS_Com_00726]	🟧				
[SWS_Com_00766]	🟩				
[SWS_Com_00859]					🟩
[SWS_Com_00814]				🟩	

2) Document considered : *AUTOSAR TPS SystemTemplate* (first 300pp)

- Versions compared: 4.2.1 – 4.2.2
- Legend: orange colour – light modification (text/grammar not context)

Table 1

- Total number of changes: 23 (relevant 22)
  - Modified: 5 (4)
  - Split: 1
  - Deleted: 1
  - Added: 16

	Mod	Mer	Split	Dele	Add
[constr_3219]					🟩
[constr_3505]			🟩	🟩	
[constr_3215]					🟩
[constr_3198]					🟩
[constr_3199]					🟩
[constr_3081]	🟩				
[TPS_SYST_02082]					🟩
[TPS_SYST_02083]					🟩
[TPS_SYST_02084]					🟩
[TPS_SYST_02085]					🟩
[TPS_SYST_02086]					🟩
[TPS_SYST_02087]					🟩
[TPS_SYST_02088]					🟩
[TPS_SYST_02089]					🟩
[TPS_SYST_02090]					🟩
[TPS_SYST_01052]	🟩				
[TPS_SYST_02091]					🟩
[TPS_SYST_02079]					🟩
[TPS_SYST_02076]					🟩
[TPS_SYST_01065]	🟩				
[TPS_SYST_01066]	🟩				
[TPS_SYST_01157]	🟧				

Table 2

- Changes within meta-classes.

	Nam	Pack	Base	At-trib.
LinSlaveConfig				🟩 3
LinSlaveConfigIdent	🟩	🟩	🟩	🟩
EthernetCommunicationCon-nector				🟩 1
ClientServerToSignalMapping				🟧 2
PncMapping				🟩 1

Macro results

1) Documents considered : *AUTOSAR SWS COM*
   - Version Compared: from 4.0.1 to 4.2.2
   - Requirements analysed: COM696 ; COM 260 ; COM734

	4.0.1	4.0.2	4.0.3	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2
COM696	/	MODI-FIED	/	/	/	SPLIT-TED	/	/
COM260	/	MODI-FIED	/	/	/	/	/	DE-LETED(MERGED)
COM734	/	ADDED	/	MODI-FIED	/	MODI-FIED	MODI-FIED	MODIFIED

Metrics

Frequency, [*Change degree*]: $r = \frac{1}{n-1} \sum_{i=2}^{n} isChanged(r, i)$

- COM696: 0,29
- COM260: 0,29
- COM734: 0,71

Sequence, [*change degree*]: $r = \max(N(r, i)), (2 \leq i \leq n)$

- COM696: 1
- COM260: 2
- COM734: 3

Distance, [*length of change time*]: $r = \sum_{i=2}^{n} (n - i)$

- COM696: 8
- COM260: 6
- COM734: 13

2) Documents considered: *AUTOSAR TPS System Template*
   - Version Compared: from 4.0.1 to 4.2.2
   - Requirements/constraints analysed: [constr_3002], [constr_3018], [TPS_SYST_01052], [TPS_SYST_01056].

	4.0.1	4.0.2	4.0.3	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2
constr_3002	ADDED	/	/	/	/	/	/	/
constr_3018	ADDED	/	/	MODI-FIED	/	/	/	/
TPS_SYST_01052	-	-	-	ADDED	MODI-FIED	/	/	MODI-FIED
TPS_SYST_01056	-	-	-	ADDED	MODI-FIED	/	/	/

Metrics

Frequency, [*Change degree*]: $r = \frac{1}{n-1} \sum_{i=2}^{n} isChanged(r,i)$

- constr_3002: 0
- constr_3018: 0,14
- TPS_SYST_01052: 0,5
- TPS_SYST_01052: 0,25

Sequence, [*change degree*]: $r = \max\big(N(r,i)\big), (2 \leq i \leq n)$

- constr_3002: 0
- constr_3018: 1
- TPS_SYST_01052: 1
- TPS_SYST_01052: 1

Distance, [*length of change time*]: $r = \sum_{i=2}^{n}(n-i)$

- constr_3002: 0
- constr_3018: 4
- TPS_SYST_01052: 3
- TPS_SYST_01052: 4

3) Documents considered: *AUTOSAR TPS System Template*
   - Version Compared: from 4.0.1 to 4.2.2
   - Classes analysed: EcuInstance (ECU), SenderReceiverCompositeElementToSignalMapping (SRCETSM), IPduTriggering (IPDU), FlatInstanceDescriptor (FID)
   - Legend of changes: Name(N) - package (P) – Base (B) – Attribute (A)

	4.0.1	4.0.2	4.0.3	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2
ECU	/	B - As	As	/	/	/	As	/
SRCETSM	/	-	/	ADDED	/	A	/	/
IPDU	-	N - A	/	/	/	/	/	A
FID	/	/	As	/	/	/	/	/

Appendix B

Validation surveys.

I. General questions

1. *Which releases of AUTOSAR are prone to more changes and which are most stable?*

☐ All the releases are equally prone to changes.

☐ The two main releases (i.e., 4.1.1 and 4.2.1) present more changes than others.

☐ Generally the first releases (4.0.2, 4.0.3, 4.1.1) are more unstable than the last ones (from 4.1.2 to 4.2.2).

☐ I don't know

2. *In your opinion there are more additions or deletions taking count of all the releases?*

☐ Additions, AUTOSAR continues to growth.

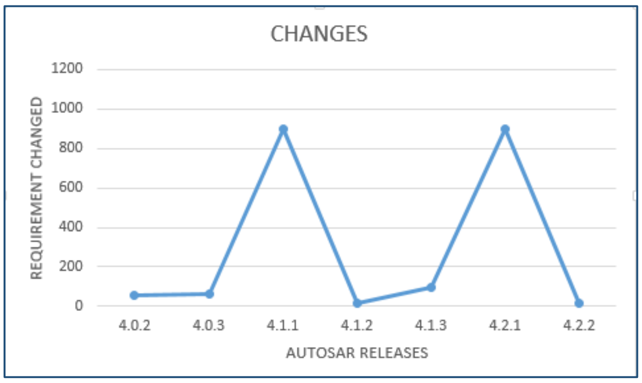
☐ Deletions, AUTOSAR has reduced the number of requirements.

☐ Roughly the same number, AUTOSAR maintains a stable number of requirements.

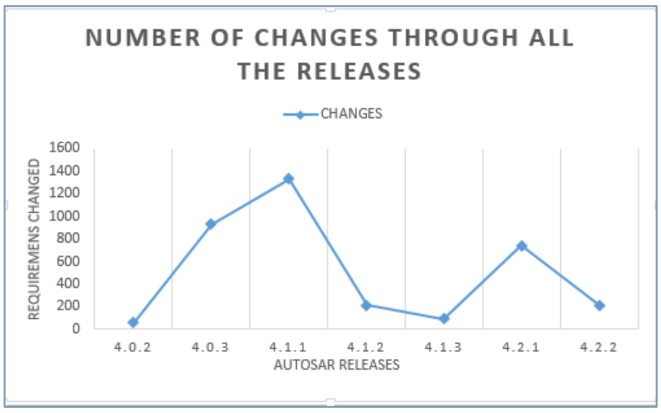
☐ I don't know

3. *Which of the following charts represent better to you the evolution of AUTOSAR through its requirements?*

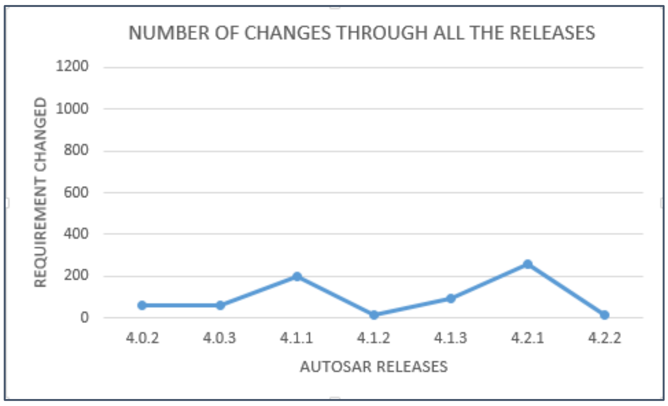
☐



☐



☐



☐ I don’t know.

4. *Based on your knowledge, which documents are mostly affected by changes from 4.0.3 to 4.2.2? (choose more than one answer)*

☐ AUTOSAR_SWS_DiagnosticEventManager

☐ AUTOSAR_SWS_CANInterface

☐ AUTOSAR_SWS_OS

☐ AUTOSAR_SWS_EEPROMAbstraction

☐ AUTOSAR_SWS_DiagnosticCommunicationManager

☐ AUTOSAR_SWS_TTCANDriver

☐ AUTOSAR_TPS_SystemTemplate

☐ I don’t know

II. Specific questions

a. Expert 1

The next questions focus on the following documents:
AUTOSAR_SWS_COM and *AUTOSAR_SWS_BSWModeManager*.

1. *SWS_COM is a document which:*

☐ Always changes a lot (e.g., more than half of requirements changed), especially in the main releases.

☐ Is usually stable, except for the main releases in which it has been strongly changed. (i.e., a high number of additions, modifications and deletions)

☐ It change quite a lot, more in the “past” (i.e., from 4.0.1 until 4.1.1) but since there are many requirements we can consider it quite stable.

☐ I don’t know

2. *SWS_BSWModeManager is a document which:*

☐ Always changes a lot, especially in the main releases.

☐ Is usually stable, except for the release 4.0.3 in which it has more changes. (i.e., a high number of additions) and the last one (4.2.1 to 4.2.2)

☐ Is complete stable through all the versions.

☐ I don’t know

3. *Based on your knowledge, which are the most common type of change in the SWS_ BSWModeManager ? (more than one answer is allowed if necessary)*

☐ Additions

☐ Modifications

☐ Deletions

☐ I don't know

4. *Based on your knowledge, which are the most common type of modification in the SWS_COM from 4.0.2 to 4.0.3? (you can write your own on the “others” field, if you want)*

☐ Changes in API specifications

☐ Changes in the functional specifications

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ No enough changes or no common type of changes

☐ I don't know

☐ *Others*

5. *Based on your knowledge, which are the most common type of modifications in the SWS_COM from 4.0.3 to 4.1.1?*

☐ Changes in API specifications

☐ Changes in the functional specifications

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ No enough changes or no common type of changes

☐ I don't know

☐ *Others*

6. *Based on your knowledge, which are the most common type of modifications in the SWS_COM from 4.1.3 to 4.2.1?*

☐ Changes in the function definitions (in the API specifications). For example the sintax, or the return value

☐ No enough changes or no common type of changes

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ I don't know

☐ *Others*

b. *Expert 2*

The next questions focus on the following documents:
SWS_CANNetworkManagement and *SWS_FlexRayNetworkManagement.*

1. *SWS_CANNetworkManagement is a document which:*

☐ Always changes a lot (e.g., more than half of requirements changed), especially in the main releases.

☐ Is usually stable, except for the main releases in which it has been strongly changed. (i.e., a high number of additions, modifications and deletions)

☐ Is quite stable in general, even for the main releases which present more changes but stil not that many.

☐ I don’t know

2. *SWS_ FlexRayNetworkManagement is a document which:*

☐ Always changes a lot, especially in the main releases.

☐ Is usually stable, except for the main release 4.1.1 in which it has been strongly changed. (i.e., a high number of additions, modifications and deletions).

☐ Is complete stable through all the releases.

☐ I don’t know

3. *Based on your knowledge, which are the most common type of change in the SWS_CANNetworkManagement? (more than one answer is allowed if necessary)*

☐ Additions

☐ Modifications

☐ Deletions

☐ I don't know

4. *Based on your knowledge, which are the most common type of modification in the FlexRayNetworkManagement from 4.0.2 to 4.0.3? (you can write your own on the "others" field if you want)*

☐ Changes in the features of a configuration parameters or switches is reflected in changes on the requirements

☐ No enough changes or no common type of changes

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ I don't know

☐ *Others*

5. *Based on your knowledge, which are the most common type of modifications in the FlexRayNetworkManagement from 4.0.3 to 4.1.1?*

☐ Changes in the features of a configuration parameters or switches is reflected in changes on the requirements

☐ No enough changes or no common type of changes

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ I don't know

☐ *Others*

6. *Based on your knowledge, which are the most common type of modifications in the FlexRayNetworkManagement from 4.1.3 to 4.2.1?*

☐ Changes in the features of a configuration parameters or switches is reflected in changes on the requirements

☐ No enough changes or no common type of changes

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ I don't know

☐ *Others*

c. Expert 3

The next questions focus on the following documents:
AUTOSAR_SWS_COM and *AUTOSAR_SWS_COMManager*.

1. *SWS_COM is a document which:*

☐ Always changes a lot (e.g., more than half of requirements changed), especially in the main releases.

☐ Is usually stable, except for the main releases in which it has been strongly changed. (i.e., a high number of additions, modifications and deletions)

☐ It change quite a lot, more in the “past” (i.e., from 4.0.1 until 4.1.1) but since there are many requirements we can consider it quite stable.

☐ I don’t know

2. *SWS_COMManager is a document which:*

☐ Always changes a lot, especially in the main releases.

☐ Is usually stable, except for the release 4.0.3 in which it has more changes. (i.e., a high number of additions).

☐ Is complete stable through all the releases.

☐ I don’t know

3. *Based on your knowledge, which are the most common type of <u>change</u> in the SWS_COMManager ? (more than one answer is allowed if necessary)*

☐ Additions

☐ Modifications

☐ Deletions

☐ I don't know

4. *Based on your knowledge, which are the most common type of <u>modification</u> in the SWS_COM from 4.0.2 to 4.0.3? (you can write your own reasons on the “others” field, if you want)*

☐ Changes in API specifications

☐ Changes in the functional specifications

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ No enough changes or no common type of changes

☐ I don't know

Others:

5. *Based on your knowledge, which are the most common type of modifications in the SWS_COM from 4.0.3 to 4.1.1? (you can write your own reasons on the “others” field, if you want)*

☐ Changes in API specifications

☐ Changes in the functional specifications

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ No enough changes or no common type of changes

☐ I don't know

☐ *Others*

6. *Based on your knowledge, which are the most common type of modifications in the SWS_COM from 4.1.3 to 4.2.1? (you can write your own reasons on the "others" field, if you want)*

☐ Changes in the function definitions (in the API specifications). For example the syntax, or the return value

☐ No enough changes or no common type of changes

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ I don't know

☐ *Others*

d. *Expert 4*

The next questions focus on the following documents: *AUTOSAR_SWS_DiagnosticCommunicationManager* and *AUTOSAR_SWS_DiagnosticEventManager*.

1. *DCM is a document which:*

☐ Always changes a lot especially in the main releases, but even in the others.

☐ Is usually stable, except for the main releases in which it has been strongly changed. (i.e., a high number of additions, modifications and deletions)

☐ It change a lot in the "past" (i.e., from 4.0.1 until 4.1.1) and now is more stable.

☐ I don't know

2. *DEM is a document which:*

☐ Always changes a lot especially in the main releases, but even in the others.

☐ Is usually stable, except for the main releases in which it has been strongly changed. (i.e., a high number of additions, modifications and deletions)

☐ It change a lot in the "past" (i.e., from 4.0.1 until 4.1.1) and now is more stable.

☐ I don't know

3. *Based on your knowledge, which are the most common type of change in DEM? (more than one answer is allowed if necessary)*

☐ Additions

☐ Modifications

☐ Deletions

☐ I don't know

4. *Based on your knowledge, which are the most common type of modification in the DCM from 4.0.2 to 4.0.3? (you can write your own on the "others" field, if you want)*

☐ Changes in API specifications(e.g., the syntax or the parameters or the description of the functions and the callout).

☐ Changes in the functional specifications

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ No enough changes or no common type of changes

☐ I don't know

☐ *Others*

5. *Based on your knowledge, which are the most common type of modifications in the DCM from 4.0.3 to 4.1.1?*

☐ Changes in API specifications (e.g., the syntax or the parameters or the description of the functions and the callout).

☐ Changes in the functional specifications.

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.).

☐ No enough changes or no common type of changes

☐ I don't know

☐ *Others*

6. *Based on your knowledge, which are the most common type of modifications in the DCM from 4.1.3 to 4.2.1?*

☐ Changes in the API specifications). (e.g., the syntax or the parameters or the description of the functions and the callout).

☐ No enough changes or no common type of changes

☐ Change in AUTOSAR conventions (e.g., SHALL to MUST or changes in the requirement name, structure, form etc.)

☐ I don't know

☐ *Others*

e. Expert 5

The next questions focus on the following documents:
TPS_SystemTemplate and *TPS_ECUConfiguration.*

1. *TPS_SystemTemplate is a document which:*

☐ Always changes a lot, especially in the main releases.

☐ Is usually stable, except for the main releases in which it has been strongly changed. (i.e., a high number of additions, modifications and deletions)

☐ Doesn't have a great number of specifications but just constraints.

☐ I don't know

2. *TPS_ECUConfiguration is a document which:*

☐ Always changes a lot, especially in the main releases.

☐ The requirements have been introduced early (before 4.0.1), thus the requirements are more "mature" and it has the 20% -25% of changes at most in the main releases.

☐ The requirements have been introduced early (before 4.0.1), however the requirements are still unstable and it has 70% -75% of changes in the main releases.

☐ I don't know

3. *Based on your knowledge, which are the most common type of change in the system template? (more than one answer is allowed if necessary)*

☐ Additions

☐ Modifications

☐ Deletions

☐ I don't know

4. *Based on your knowledge, which are the most common type of modification in the system template from 4.0.2 to 4.0.3?*

☐ Changes in the name of some meta-classes

☐ No enough changes or no common type of changes

☐ Additions of values (e.g., the attribute x of y can have the following values)

☐ I don't know

☐ *Others*

5. *Based on your knowledge, which are the most common type of modifications in the system template from 4.0.3 to 4.1.1?*

☐ Changes in the name of some meta-classes

☐ No enough changes or no common type of changes

☐ Additions of values (e.g., the attribute x of y can have the following values)

☐ I don't know

☐ *Others*

6. *Based on your knowledge, which are the most common type of modifications in the system template from 4.1.3 to 4.2.1?*

☐ Changes in the name of some meta-classes

- ☐ No enough changes or no common type of changes
- ☐ Additions of values (e.g., the attribute x of y can have the following values)
- ☐ I don't know
- ☐ *Others*