Automotive Powertrain Software Evaluation Tool

Master Thesis

Submitted in Fulfilment of the Requirements for the Academic Degree M.Sc.

Dept. of Computer Science
Chair of Computer Engineering

Submitted by : Kalkin Powale
Student ID : 358753
Date : 12.07.2017

Supervisors : Prof. Dr. W. Hardt ; M.Sc. O. Khan ; M.Sc. H. Venkitachalam
Abstract

The software is a key differentiator and driver of innovation in the automotive industry. The major challenges for software development are increasing in complexity, shorter time-to-market, increase in development cost and demand of quality assurance. The complexity is increasing due to emission legislations, variants of product and new communication technologies being interfaced with the vehicle. The shorter development time is due to competition in the market, which requires faster feedback loops of verification and validation of developed functionalities. The increase in development cost is contributed by two factors; the first is pre-launch cost, this involves the cost of error correction in development stages. Another is post-launch cost; this involves warranty and guarantees cost. As the development time passes the cost of error correction also increases. Hence it is important to detect the error as early as possible. All these factors affect the software quality; there are several cases where Original Equipment Manufacturer (OEM) have callbacks their product because of the quality defect. Hence, there is increased in the requirement of software quality assurance. The solution for these software challenges can be the early quality evaluation in continuous integration framework environment.

The most prominent in today’s automotive industry AUTomotive Open System ARChitecture (AUTOSAR) reference architecture is used to describe software component and interfaces. AUTOSAR provides the standardised software component architecture elements. It was created to address the issues of growing complexity; the existing AUTOSAR environment does have software quality measures, such as schema validations and protocols for acceptance tests. However, it lacks the quality specification for non-functional qualities such as maintainability, modularity, etc. The tool is required which will evaluate the AUTOSAR based software architecture and give the objective feedback regarding quality.

This thesis aims to provide the quality measurement tool, which will be used for evaluation of AUTOSAR based software architecture. The tool reads the AUTOSAR architecture information from AUTOSAR Extensible Markup Language (ARXML) file. The tool provides configuration ability, continuous evaluation and objective feedback regarding software quality characteristics. The tool was utilised on transmission control project, and results are validated by industry experts.

Keywords: AUTOSAR, Software Quality, Metric, Automotive Software Architecture, Continuous Integration, Agile Methods, Frontloading
Acknowledgement

First of all, I would like to thank Prof. Dr. W. Hardt for allowing me to pursue this thesis in the department of Automotive Software Engineering of Technische Universität Chemnitz. Special thanks to my internal supervisor M.Sc. Owes Khan for providing all kind of support for the fulfilment of my thesis.

Furthermore, I would like to thank my company supervisor M.Sc. Hariharan Venkitachalam of FEV Europe GmbH for giving me this opportunity to work on this thesis topic. His constant support, guidance, constructive feedback and encouragement have contributed in the successful completion of my master thesis. I would like to thank Mr. Chandra Shekhar Jha, Dipl. Ing. Peter Vranken and Mr. Chen Shaoning for giving me invaluable support and inputs throughout my thesis. I would also like to thank my colleague Miss. Rashmi Bharadwaja, Mr. Vjaykumar Konenki and all the team of BES-FEV for creating a pleasant working environment.

I am blessed to have my parents Mrs. Asawari Powale and Mr. Anand Powale, and their continuous support, love, and encouragement for my endeavours, without which none of this could ever be possible. I would also like to thank my friends Rashmi and Shivraj for providing me with unfailing support and encouragement throughout my study period.

Finally, I would like to thank all those people who have supported me in any respect during my thesis. This journey would not have been completed without your helpful hands, thank you.
1. Introduction .................................................. 10
   1.1. Motivation ............................................. 10
   1.1.1. Increase in software complexity .................. 11
   1.1.2. Decrease in development time ..................... 12
   1.1.3. Software quality defects ......................... 13
   1.1.4. Challenges of architecture decision making .... 15
   1.2. Research question .................................... 16
   1.3. Contribution of this thesis ......................... 18
   1.4. Organization of the thesis ......................... 18

2. Technical overview ......................................... 19
   2.1. Automotive powertrain domain ..................... 19
   2.2. Introduction to automotive system architecture ... 20
   2.3. Overview of the AUTOSAR development methodology 23
      2.3.1. Basics of AUTOSAR software architecture .... 26
      2.3.2. AUTOSAR methodology ............................ 30
      2.3.3. AUTOSAR ARXML file features ................ 31
   2.4. Summary .............................................. 33

3. State-of-the-art .......................................... 34
   3.1. Software measurement process ..................... 34
   3.2. Definition of software quality .................... 36
   3.3. Architecture evaluation methods .................. 38
      3.3.1. Scenario-based evaluation method .............. 39
      3.3.2. Metric-based evaluation method ................ 40
      3.3.3. Requirement from metric ........................ 42
   3.4. Architecture evaluation literature review ........ 43
   3.5. Continuous integration framework ................. 45
      3.5.1. Quality measurement toolchain ................. 46
## List of Figures

1.1. Evolution of calibration variables with time [94]. .......................... 11
1.2. Global carbon emission from fossil fuels [20]. ............................. 12

2.1. AUTOSAR layered architecture .................................................. 26
2.2. Graphical representation of software component in AUTOSAR ........ 27
2.3. Overview of AUTOSAR methodology ......................................... 30
2.4. ARXML file of SWC ............................................................. 32

3.1. Measurement process model [8]. .................................................. 35
3.2. Software quality model as per ISO 25010 [93]. ............................. 37
3.3. GQM based derivation of metric for simplicity. ............................. 42
3.4. Basic structure of continuous integration framework [77]. ............ 45
3.5. Quality measurement toolchain [93]. .......................................... 46
3.6. Quality evaluation toolchain [43]. ............................................. 47

4.1. Quality model for AUTOSAR based powertrain software architecture
4.2. State flow of AUTOSAR quality measurement toolchain. ............... 62

5.1. ARXML file parser stages. .......................................................... 66
5.2. Dependency of SWC example. ...................................................... 68
5.3. Runnable level architectural characteristics. .................................. 69
5.4. Integration of AUTOSAR toolchain into continuous integration frame-
    work .................................................................................... 81

6.1. Metric result for Measure of Runnables ....................................... 83
6.2. Metric result for Measure of Inter Runnable Variable .................. 83
6.3. Metric result for Measure of Average Client-Server Interactions ..... 84
6.4. Metric result for Measure of Periodic Trigger Runnable ............... 84
6.5. Visualisation of triggering behaviour of component TGI ............... 84
6.6. Metric result for Measure of Communication Dependency ............ 84
6.7. Metric result for Percentage of Unconnected Report ..................... 85
6.8. Metric result for Interface Re-usability Index ............................... 85
6.9. Metric result for Ease of Integration ........................................... 85
6.10. Metric result for Measure of Dependencies ................................. 85
6.11. Quality analysis process model .................................................. 88
List of Tables

1.1. Scenario-based architecture evaluation methods [61].................. 14
2.1. AUTOSAR container names and its description .......................... 33
3.1. ISO 26262 based design principle for software architecture [2]........ 38
3.2. Scenario-based architecture evaluation methods. ......................... 39
3.3. Existing metric for different quality goal. ............................... 44
4.1. Developed metric at system software component level for different
     quality goal .............................................................................. 52
4.2. Developed metric at system abstraction level for different quality goal 60
5.1. Table depicting parser results. .................................................. 67
5.2. DSM stage results. ................................................................. 68
5.3. DSM for component structure presented in the Figure 5.2. ............ 68
5.4. Dependency metric elements description ................................. 79
6.1. System abstraction level metrics. ............................................ 82
A.1. Developed metric at system software component level for different
     quality goal .............................................................................. 106
A.2. Developed metric at system abstraction level for different quality goal 107
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSAR</td>
<td>AUTomotive Open System ARchitecture</td>
</tr>
<tr>
<td>SECOs</td>
<td>Software Ecosystems</td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial-off-the-shelf</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit</td>
</tr>
<tr>
<td>CBSE</td>
<td>Component Based Software Engineering</td>
</tr>
<tr>
<td>SWC</td>
<td>Software Component</td>
</tr>
<tr>
<td>NvRAM</td>
<td>Non Volatile Random Access Memory</td>
</tr>
<tr>
<td>VFB</td>
<td>Virtual Functional Bus</td>
</tr>
<tr>
<td>RPort</td>
<td>Receiver port</td>
</tr>
<tr>
<td>PPort</td>
<td>Provider port</td>
</tr>
<tr>
<td>IRV</td>
<td>Inter Runnable Variable</td>
</tr>
<tr>
<td>RTE</td>
<td>Run Time Environment</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>ARXML</td>
<td>AUTOSAR Extensible Markup Language</td>
</tr>
<tr>
<td>AAF</td>
<td>Automotive Architecture Framework</td>
</tr>
<tr>
<td>ADF</td>
<td>Automotive Design Framework</td>
</tr>
<tr>
<td>ADL</td>
<td>Architecture Description Language</td>
</tr>
<tr>
<td>AFAS</td>
<td>Architecture Framework For Automotive System</td>
</tr>
<tr>
<td>HMI</td>
<td>Human machine interface</td>
</tr>
<tr>
<td>AF</td>
<td>Architecture Framework</td>
</tr>
<tr>
<td>GQM</td>
<td>Goal-Question-Metric</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>RE</td>
<td>Runnable Entity</td>
</tr>
<tr>
<td>COM</td>
<td>Communication Module</td>
</tr>
<tr>
<td>DSM</td>
<td>Dependency Structure Matrix</td>
</tr>
</tbody>
</table>
1. Introduction

Since the introduction of software in the automotive sector for engine management, it has been utilised for providing various complex and innovative solutions in the automotive industry. Initially, there were 4-5 ECUs per car, but this number has changed very exponentially in recent years. The survey conducted in 2015 showed that a common vehicle had around 70-80 ECUs with several bus systems (e.g. LIN\textsuperscript{1}, CAN\textsuperscript{2}, etc.)\cite{95} and very high number of Line-of-Code (LOC)\cite{23}. The reasons behind this exponential growth are market-driven challenges such as fuel efficiency, emission regulations, safety-critical issues, market regulations and customer specific demands\cite{45}. To cope up with these challenges the automotive industry has expedited innovations in electronics and software technologies. A modern automotive has a large number of system variants, calibration parameter variables, complex communication networks among different sensors, actuators, control units, etc. Because of this, the complexity of a system and its development cost share for software has increased exponentially\cite{99,64}. Around 40\% of vehicle development cost comes from electronic and software system, out which software contributes 50-70\%\cite{40}. Figure 1.1 shows an increase in a number of calibration variables of engine management system over the period, from which we can realise the increase in complexity of development process. The increase in complex functionality has demanded more quality assurance.

Along with the complexity challenges the automotive industry is trying to cope with rapidly changing market demand which has resulted in shorter time-to-market. A study shows that the development process plays a vital role in automotive product's success or failure\cite{47}. To address the growing complexity the automotive industry is moving towards architecture standardisation like AUTOSAR. The AUTOSAR deals with the improvement of certain quality attributes like scalability and portability. However, some other quality characteristics like testability are not directly tackled by AUTOSAR. This approach deals with the automatic evaluation of quality criteria based on metrics for AUTOSAR software architectures.

1.1. Motivation

To understand the importance of quality assessment of automotive software architecture, we first need to understand the current challenges of automotive software.
sector. These challenges are a key motivation behind the development of quality evaluation toolchain.

1.1.1. Increase in software complexity

Today automotive embedded systems is not restricted to only engine management issues [18], but it has provided solutions for different features in vehicles such as powertrain and chassis control (automatic transmission, hybrid control, etc.), body electronics (instrument panel, key, etc.), multimedia applications (car audio, car navigation, etc.) and integrated systems/services (pre-crash safety, parking assistance, etc.). All these functionalities require complex and innovative solutions, which are realised by software technologies.

Considering global environment condition, Figure 1.2 shows an increase in CO2 emission levels over the years; this has triggered the tightening emission norms by the emission regulation authorities across the world. The increase in vehicle emissions has necessitated complex algorithms to control the powertrain [45]. This forces the vehicle manufacturers to advance their existing technologies for more advanced solutions. One of the methods by which emissions were reduced is by electrification of the powertrain. Control of various power sources has increased the algorithmic complexity of the powertrain software.

Moreover, with an increase in globalisation each market has created their own spe-
1. Introduction

![Figure 1.2: Global carbon emission from fossil fuels [20].](image)

1.1.2. Decrease in development time

Today’s competitive environment has pushed automotive industry to develop the cutting edge automotive solutions. Traditionally, an automotive product development cycle requires 4 to 5 years, but recently this time has been reduced [90]. The development of automotive software is based on the V-Model of development process flow [51]. In this process, if a fault or an error is found in software at any given stage then feedback is sent to the developer to make changes in software to mitigate the problem. The mitigation process requires repeating the previous stage or stages depending upon the origin of the problem. This repetition of stages adds additional time constraints on the development cycle. To shorten development time, the development process must have faster and shorter feedback loops of verification and validation by simulation [59] and testing. A faster feedback loop will allow detecting specific requirements [47]. To comply with varying requirements, the automotive manufacturers needed to implement complex management of software variants. Even though the requirements differ from market to market, the product is developed as a single unit. The challenge of variants of the product is realised by automotive software architecture. This has added additional burden on software, resulting in a complex system. The software complexity reduces software quality such as testability, maintainability, reliability, etc. [23]. To produce better software quality product, the extra amount of man-hours need to be spent on the project. This will increase time and cost of the project.
an error soon after it is introduced in the development artefact. The generation of simulation and testing model adds additional cost and consume development time of the project. The test model is developed after completion of certain software development stages. If a fault in architectural design detected during testing than the software needs to be redesign and so as the test model for it, which will add additional cost and time to the project.

Additionally, in a software development cycle, Cost of error detected at later stages of development is high. Hence, the error detection has to be done at early stages of development. This early realisation of verification and validation is called as "front-loading" [92]. To have faster feedback loops, it is required to have frontloading of functionalities. This way we can perform verification and validation tests in early stages itself. The early verification and validation will allow us to detect faults or erroneous problems in advance stages itself. However, there is a decrease in development time due to reduced time-to-market. This coupled with the increasing complexity (described in the previous sub-section 1.1.1) are some of the key problems faced by the automotive software community, hence, the time and effort required to perform early verification and validation by tradition methods will be very high, and it will not be a feasible to perform operation in shorter feedback loops. Hence, to perform this check repeatedly in faster feedback loops an automatic tool is required.

The automotive product is developed in a distributive environment with more than one suppliers responsible for system development. The specification requirement from manufacturer evolves with product development life cycle. The newly added software functionality requires additional time for development. However, the expectation from the supplier is to produce a result in same time. Hence, the industry is moving towards agile development process [64]. In agile environment quality of software can get eroded if new architecture decisions are not coherent with existing architecture. Hence quality monitoring tool will be useful for determining the coherency of architecture.

1.1.3. Software quality defects

The recent study in 2017 shows that within eight years the electric vehicle will take over the automotive market [88]. The automotive sector is introducing more advance technologies such as hybrid or electric vehicle. As a result, there are increasing concerns related to safety and reliability of automotive software. Moreover, as these advanced functionalities which are implemented using electronics and software technologies, it challenges automotive sector to produce defect less software. Since 2015, Ford has recalled nearly 2.9 million vehicles, because of software defects[17]. It is not an exceptional case, in past years several car manufacturers have recalled their vehicle because of the defect. For example, General Motors has recalled around 4.3 million since 2014 [66]. The recall due to software fixes has increased over the recent years. The Table 1.1 shows the "recall" called by major manufacturers and software
1. Introduction

related issues responsible for it. The statistic over major car manufacturers shows that from 2005 to 2012 around 32 recall caused has a software problem, while from 2012 to 2015 this number has increased to around 62 [100].

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Issues</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIAT CHRYSLER</td>
<td>SW security flaws</td>
<td>Recall of 1.4M cars</td>
</tr>
<tr>
<td></td>
<td>SW incompatibility issue caused propulsion system to shut down of</td>
<td>Recall of 5600 electric vehicles</td>
</tr>
<tr>
<td>TOYOTA</td>
<td>SW flaw caused hybrid system to shut down</td>
<td>Recall of 1.9M hybrid cars</td>
</tr>
<tr>
<td></td>
<td>SW flaw caused ABS and traction control system to shut down</td>
<td>Recall of 260000 vehicles</td>
</tr>
<tr>
<td>HONDA</td>
<td>SW flaw caused high stress in drive pulley shaft</td>
<td>Recall of 145000 cars in USA</td>
</tr>
<tr>
<td></td>
<td>Sensor flaw caused sudden loss of power</td>
<td>Recall of 6786 hybrid cars</td>
</tr>
<tr>
<td>LAND ROVER</td>
<td>SW flaw caused vehicle doors to be unlatched</td>
<td>Recall of 65000 cars</td>
</tr>
<tr>
<td></td>
<td>Anti-lock braking system disable stability and control safety systems</td>
<td>Recall of 2687 SUV’s</td>
</tr>
<tr>
<td>Mercedes Benz</td>
<td>SW flaw caused engine to stop</td>
<td>Recall of 3000 cars</td>
</tr>
</tbody>
</table>

Table 1.1: Scenario-based architecture evaluation methods [61].

The defect fixing adds additional cost to manufacturers. Hence there is a high requirement on reducing defects. To minimise the software defects in a vehicle, there is an emphasis on software quality. As a consequence, there is more increase in the importance of software quality assurance.

The software defect adds the cost in different ways depending upon phase in which it detected of vehicle life cycle. If it defects detected after the release of the product, it adds very high cost to manufacturer and also tarnished its brand image. Hence it is very important to detect the defect as early as possible in vehicle development phase.

Another reason for early error detection is the chances of building new functionality on unknown induced error. Let us understand this, in earlier section 1.1.1 we have discussed how the advanced functionality drives complexity and thereby reduces testability and maintainability of the software. Considering V-Model of process development, if on a certain stage of development a design fault or error is induced in
1. Introduction

software, and if the software has poor testability then it may not be able to detect it by validation and verification method. The accidental induced error will percolate to further stages of development. Till the developer realises about this problem, he already has built new functionalities on top of it, which is in the vulnerable position because of earlier dependencies.

Ideally, to fix this problem, the software again has to undergo all previous stages starting from where the error has introduced or originated. This rework cycle takes excess time and human effort which additionally increases cost. Moreover, if the software has poor maintainability, then it is very difficult to fix the problem. More often to fix such a problem and avoid rework, developers are inclined towards "quick-fix" solution i.e. changing or working on the problem at that stage of development itself. The software developed with such quick fixes does not comply with originally decided architectural design, resulting in erosion of software quality.

1.1.4. Challenges of architecture decision making

As discussed in earlier section 1.1.1 and 1.1.2 the automotive industry is facing increasing complexity and shorter time-to-market as a consequence, increase in quality assurance. So far we have studied the how global factor can affect the software quality and increasing its importance in today's automotive industry. This section deals with the more fundamental cause which shapes the software quality.

The software architecture has very close relation with software system quality attribute such as modifiability, security, performance, etc. [29]. As most of the design decision which shapes the overall system software takes place at early stages of the development cycle of vehicle hence, these decision has very strong impact on software quality [34] [82] [29]. The influence of architecture decision has influence on various stages of system software development life cycle such as testing, integration, etc. [34]. Hence it is very important to make right architectural design decisions.

The recent study shows that the most of the software architecture designers follow hybrid method for design decision making, i.e. first they start with the basic framework of software architecture, and as the project and requirement evolve, they make appropriate design decision [34]. The study also reveals that the design decisions mostly are done in informal methods using simple notations rather than some formal methods [34]. The most prominent reason for this, in amongst all developers was a flexible and easy process, i.e. the decision making process they follow was easy to follow, faster, responsive to customer demand, etc. While formal architecture decision making method depends upon an evaluation of various options, communication with several stakeholders and takes the time to reach a conclusion and with the demand of shorten time to market these methods often ignored [87].

\footnote{Solution that appears to be fast and easy, however in long term it is not good for quality.}
1. Introduction

Moreover, the study also revealed that the informal decision making process often lacks proper documentation; hence it gets difficult to track decisions and revisit its design rational. The lack of systematic decision making process and tight schedule of project development often results in poor quality of developed software. Even though current technique yields faster results, it shows an inability to revisit these decisions which will have a negative impact on further stages of development [28] and so on the software quality.

The automotive software development takes place in distributed teams. The software architecture quality can also be deteriorated if there is a change in workforce [78]. For example, if a member of the team is replaced or added, he may not be able to follow with original architectural design, and so the project will be deviated from the required design. As a result poor software quality. The presence of assessment tool can provide quick feedback to the developer regarding such deviation.

With the standardisation of architecture using AUTOSAR, hundreds of specification guidelines and architectural elements are available with software developer for architectural design. However, it challenges developer design decision ability, for example, communication among component can be via Sender-Receiver or Client-Server interface, it can be implicit or explicit, etc. These decisions have a consequence on architectural quality. Hence, if the developer receives an objective feedback about software architecture quality it will be very beneficial, it allow developer with ability to review and change the architectural design by making educated decision based on existing quality condition, in early stages of development itself, resulting in better quality software.

1.2. Research question

In previous section 1.1 we have discussed the increase in the demand of good quality software by the manufacturer. Fundamentally, the architecture of software sets the boundaries of final product’s software quality[34]. Hence, by assessing architecture of the software will help the developer to maintain the quality of software throughout the project’s development cycle.

To assess the software architecture it is very important to have standardised architecture. AUTOSAR provides standardised architecture, in terms of software component implementation view [1]. Today around 202 automotive and IT sector companies are developing AUTOSAR based software [76]. Having tool which can automatically evaluate software architecture quality will be instrumental in making design decisions and quality assessment. As of now, there is no quality assessment solution which can assess AUTOSAR based project’s software architecture quality in early stages of development; research is seeking in the identification of quality
characteristic and its measurement to get objective feedback regarding automotive software architecture.

**RQ1. Which quality attributes of AUTOSAR application software can be evaluated at early stages of development?**

The thesis intends to analyse architecture software quality. Hence, we first need to discuss the definition of quality itself. Then the quality model needs to be established which will show us various quality characteristics associated with the automotive software architecture. There are various standards which are related to software quality [4] [2]. A critical analysis is needed to analyse which quality characteristics can be determined from AUTOSAR based software architecture. This determination is done based on an available artefact from software architecture at the early stage of development. Hence, the study of AUTOSAR architecture and its standardised information sharing file format needs to be done.

**RQ2. How metric should be defined to evaluate quality of the automotive powertrain software architecture?**

Once the quality characteristics have been identified, the quantitative characteristic value will be extracted from the available architectural artefact. Depending upon quality goals, architectural information will be identified. The derivation of a mathematical formula is needed which provides us the quantitative quality value. The method should be derived which will provide a right metric for given quality. The study requires identifying the characteristics of such metric, its methodology of derivation and its application process. The relationship between the derived value and the software quality characteristic has to be established.

**RQ3. How to improve software development quality from evaluation tool by giving objective feedback?**

The research is seeking in development of evaluation tool which can comply with automotive sector demands. The AUTOSAR architecture information is present in ARXML (AUTOSAR XML) file which needs to be parsed. The software architecture decision making process lacks the light weighted evaluation tool (easy to implement and usages) which can assess developer in decision making process by providing objective feedback. As discussed in section 1.1.2 the development process is very agile in nature; hence the tool should able be to integrate with changing requirement specification and evolving system design. The cost and time are very important issues in software development process. Hence automatic and simple-to-operate should be the key requirement from the evaluation tool. To achieve shorter feedback loop, it should be able to produce results in time bound manner. The tool should be able to reproduce same results for same given input condition, i.e., the
1. Introduction

results should not have room for subjective analysis. In automotive project based on AUTOSAR based architecture, architectural information is transferred via standardised file format. Hence, evaluation tool must take this standardised process into consideration.

1.3. Contribution of this thesis

The thesis contributes to a development of support tool for evaluation of software architecture quality. In this regard, the toolchain is developed for an identification of software quality for AUTOSAR based automotive software architecture. The architecture artefacts are extracted from ARXML file of an automotive software project. The mathematical formulation of metric is created to provide objective feedback on software quality. The development of autonomous evaluation toolchain for measuring the quality is specifically developed for automotive powertrain domain projects, but it can be easily extended to other domain as well considering their quality demands.

1.4. Organization of the thesis

The thesis is organised into seven chapters. Chapter 1 is an introductory chapter; it gives a general overview of the thesis. The overview covers the challenges in the automotive software domain, available scope for improvement and need for research in this field. Chapter 2 discusses the technical understanding needed for comprehend the thesis. The automotive software is a vast field hence only relevant concepts are explained in this section. Chapter 3 presents the work done by different research scholars in the field of software quality evaluation. It also outlined the methodology and requirement for quality evaluation tool development. Chapter 4 explains the proposed solution for quality evaluation of automotive software architecture. Chapter 5 describes the proposed solution in detail and the developed toolchain for measuring the software quality. This chapter also deals with the definition of different metrics for quality measurement and inference of their result. Chapter 6 discusses the obtained results from evaluation tool. It explains the analysis of result for deriving objective feedback and gives an overview of an analysis process of metrics results. Finally, chapter 7 describes the thesis in brief and discusses the limitation of this work and future improvement that can be done further in this work.
2. Technical overview

Although, the quality of automotive software architecture is determined by the various domains of this sector, but in this thesis we restrict ourselves to powertrain domain of automotive sector only. Hence, before moving towards architecture aspect of it we should have a basic understanding of powertrain domain. In first section 2.1 we will discuss the automotive powertrain domain. Then in second section 2.2 we will overview different traits in automotive system architecture. In section 2.3, we will overview the AUTOSAR architecture and its software development methodology.

2.1. Automotive powertrain domain

As we know from the previous chapter that how the importance and need of software quality have increased in recent years. Considering the fact that the in-vehicle embedded system is developed as domain specific, these domains are based on functionalities, constrains and models [64]. As per the European ITEA 1 EAST-EEA 2 project, domain is defined as ”a spear of knowledge, influence, and activity in which one or more systems are to be dealt with” [5]. Depending on the functionality, the five common domains are powertrain, chassis, body, Human Machine Interface (HMI), and telematic. The powertrain domain mainly deals with, vehicle engine, transmission system, and all its subsidiary functionalities. The chassis domain is responsible for vehicle control like steering and breaking system. The body domain facilitates vehicle structural entities, like an air-bag, lighting, air conditioning, etc. The HMI and telematic domain are concerned with vehicle communication system. The HMI is responsible for sharing vehicle information with user/driver, this includes control switches, monitoring or control display. The telematic is responsible for information exchange between vehicle and outside world, it allows the user to access outside services like radio, navigation, etc. From automotive software quality perspective, not all domains have an equal quality requirement. The quality demand varies with quality aspect. For example, considering safety criticality quality aspect, telematic will surely have less demand than powertrain. However, based on market demand each domain has its own quality and cost effective demand which needs to be considered for the specific domain itself. This thesis is focusing on powertrain domain quality aspect, but evaluation tool can be of use to other domain considering

---

1ITEA - multination cluster supporting innovation [5].
2EAST-EEA - Embedded electronic architecture [5].
domain quality requirement.

As mentioned earlier, powertrain domain is responsible for engine control system, hence it has very hard real-time constraint and very complex functionalities with around 2000 moving parts [88]. The domain functionalities include vehicle speed control depending on throttle position, reading break paddle sensor position, etc. Apart from that it also has to serve the demand from another domain software components, for example requirement for climate control or transmitting current information of "revolution per minute" (rpm), vehicle speed, fuel availability, etc. The domain also facilitates driver friendly amenities like driving comfort to the user via drive by wire technology, management of optimisation of fuel consumption, etc. From a functional point of view, the powertrain domain operates with different working modes of the motor (partial load, full load, etc.); this corresponds to a number of the complex parameters handling system. Considering high safety criticality and emission regulation standards, its solution lies in the development of the complex system. Moreover, implementation of such functionalities takes place as a part of several tasks with different activation period specified by sampling requirement. This imposes an additional requirement on task scheduling for safe communication of sub systems. The validation of these functionalities is time consuming and complex task. It adds significant cost to the project development. Hence, for creating better cost efficient quality software, there is a requirement for the automatic tool which will allow stakeholder to monitor project’s software quality.

The growing software complexity with demanding quality in today’s automotive systems has pushed the scientists for the development of various solutions. There are various papers has been written in identifying software quality in a higher-level software system. But in recent years, scientists and manufacture communities have recognised that the main contributor to software quality is not a high end solution but very fundamental concepts of its developments i.e. software architecture [18] [36]. In the following section 2.2, let’s understand the impact of architecture on quality and recent trends in the industry.

2.2. Introduction to automotive system architecture

The influence of software architecture on software quality is very fundamental in nature, because most of the design decisions are taken at architecture level i.e. in early stages of the software development [35]. The automotive software architecture is a very vast field. Although there is international standard ISO 42010 present for software architecture, the automotive software architecture is not standardised to any single architecture framework. It is very difficult to comprehend all automotive software architecture traits in one thesis. This section will provide an overview of some of the architectural framework traits in the automotive domain. According to ISO 42010, architecture is generally defined as
2. Technical overview

"The architecture of software system is a fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". [53]

Architecture basically describes the systems inherent structure, and an abstraction of a system in parallel. The architecture of the system is responsible for decomposition of system, separation and integration of concerns, abstraction of a system for information hiding, thus allowing stakeholders in ease of managing the system and controlling its complexity. Hence, the well planned architecture will have a high potential of resulting in a better manageable software system. The system architecture also helps in realising various factors which drives the system development such as, common ground for stakeholders communication, reuse of functionalities, planning and organising the development activities like project planning, distribution of labour, implementation, facilitating early verification, integration, etc.[24] Because of such a wide influence of architecture over system software development, the architecture decisions also have high significance in shaping quality of the system. The tool support developed in this thesis will help the developer in understanding the current software architecture quality resulted in their design decisions. The present quality understanding will help the developer in making a new decision for further improvement of software quality.

Before recognising software architecture quality we must understand different aspects of automotive software architecture itself. In recent years automotive manufacturer also known as original equipment manufacturer (OEM) has started developing automotive functionalities in parts. They distribute the functional development activity to one or more suppliers. It is then the responsibility of suppliers to implement and deliver the assigned functionality in the form of software and/or hardware [23]. To support manufacturing across the industry OEM’s started moving towards common standardised automotive architecture framework. Some of the automotive architecture frameworks have been illustrated in following paragraphs.

With an increase in the complexity of automotive domain software, there was a need to have structure view of system so that it can be able to handle easily. The architecture framework (AF) is used to structure the system in a standardised way. Depending upon the use of the method, some are more generic while others are specific to particular model but the basic idea of structuring and decomposing remain the same. According to ISO 42010 an AF is

Conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders. [53]

Basically AF is defined as (i) the architecture concerns, (ii) a set of stakeholders related to those concerns, (iii) a set of architectural viewpoints which realise or frame those concerns and (iv) a set of correspondence rules to impose constraints.
Concern - concern is the specific area of interest in the system related to stakeholders requirements or priorities.

Stakeholder - a stakeholder is a representative of group which interested in achievement of a set of goals within system under consideration.

Description - architecture description is a fundamental conception and abstraction of a system. It documents architecture in terms of system’s (i) boundaries and the interface to its environment, (ii) traceable requirements, (iii) system decomposition, (iv) components or sub-system interaction, (v) describing system or sub-system functionality and (vi) the principles guiding the system’s design and evaluation.

View - an architecture view is a representation of architecture related concern; the concerns are framed in architectural viewpoint.

Viewpoint - a viewpoint allows the user to examine the system related to a particular concern, its development process and/or its usage. The view and viewpoint allow user to comprehend the complex system for more manageable development process [53].

As we discussed earlier, there is no standardised architecture framework for automotive domain. However, recently there are some attempts been made in this direction. Some of them are describe briefly as follows,

A) Automotive Architecture Framework (AAF) [24] -
AAF is the first attempt towards the standardisation of automotive architecture framework. The AAF gives overall architecture concepts in vehicle development, it provides common layer across functional domains, it reflects the automotive industry’s need of interdependencies between the stakeholders (OEMs, suppliers, etc.) and the necessities coming with the requirements of the vehicle life cycle.

B) Architectural Design Framework (ADF)[42] -
ADF is the specification of an architectural design framework which is developed by and for the automotive industry. As like AAF, the ADF provides an architectural framework for various viewpoints such as operational, functional, constructional and requirement viewpoints. Quite similar to AAF, in ADF functional viewpoint is also concerned with views such as functional breakdown structure, functional architecture and allocation on functions. Unlike the AAF, the ADF specifies the model kind of each functional view. In ADF, SysML model diagrams are used for various functional views. It uses Activity Diagram(AD) for defining system functions, the Block Definition Diagram(BDD) for defining port and connectors of functions external interface and the Internal Block Diagram(IDB) for defining object flow specified in ADs.

C) Automotive Architecture Description Language(ADL)[33] -
According to the ISO 42010, an ADL is any form of expression which is used to describe an architecture [53]. The main benefit of ADL is that it enables formal representation of architecture, which helps in creation of better architectural doc-
umentation and improves communication among stakeholders. The stakeholder concerns can be addressed via various model such as data flow diagram, class diagrams, state machine diagram, etc. In automotive industry there are some ADL prominently present such as, EAST-ADL[32], AADL[39], AML[22], TADL[73] apart from this there are some general-purpose languages present such as SysML[68] and MARTE[44]. Due to the presence of such different variety of approaches, commonalities among companies are very difficult to find. Moreover, ADL must also comply with company's internal process and practices; this raises the question of user acceptance.

Considering, the acceptance level of ADL, AUTOSAR has a very wide range of membership across the industry. Hence, AUTOSAR can be considered for ADL integration. The Runtime Environment (RTE) acts as middleware and virtual function bus (VFB) as ADL [64].

**D) Architecture Framework For Automotive Systems (AFAS)[33]**

In AFAS analysis of various viewpoints and views of AFs such as AAF[24], ADF[42] and ADLs[33] are taken into consideration, and integrated methodology has been proposed. This work proposes some prototype tool but future work is still needed for a comprehensive case study of in an industrial setting.

Apart from such standardised architectural framework automotive companies have their own proprietary architectural framework design guideline. Hence, architecture evaluation tool must be compliant with a wide range of industry partner. In this regards, tool proposal for AUTOSAR architecture evaluation will have a very large audience.

**2.3. Overview of the AUTOSAR development methodology**

Earlier, as the micro-controller based automotive electronics gained popularity their coordination among each other becomes a major challenge. A separate ECU was dedicated for each functionality such as engine management or transmission control, with growing functional requirement number of ECUs also increased. More number of ECUs caused an increased in network coordination complexity and its development cost. The solution came as Controller Area Network (CAN) in 1983, it help in handling large signal data across network. However, in the case of multiple manufacturers developing common functionality on different ECUs involved, it requires additional effort and cost for integration and testing. To provide a solution for such problem automotive OEMs in 2003 come up with the common standardised platform in the form of AUTOSAR consortium.
AUTOSAR was formed to address the growing automotive system complexity. It is realised in layered structure to address three main issues,

1. Quality of automotive software like the reusability, the scalability and the transferability, etc.

2. To Facilitate the usage of commercial-off-the-shelf (COTS) components across the industry.

3. And to optimise product cost and time to market.

As more and more companies are moving towards standardisation of their software development process, AUTOSAR stands out among various options available. As per the latest report of June 2016, there are total 9 core partners, 46 premium partners, 30 development partners and over 117 associate partners who joined the AUTOSAR group[76]. The survey conducted in 2009 shows that around 81% of global car sale was done by AUTOSAR members [58]. With the release of AUTOSAR version 4.3.0, AUTOSAR is looking for long term objectivity of software quality, maturity, backward compatibility, fulfilling a market needs and reducing development effort.

Unlike other architecture development language AUTOSAR does not provide actual architectural design language, it provides an implementation view of a component within architecture [13]. The implementation view holds the architecture in the standardised description. AUTOSAR allows hardware independent software development by separating the application software and associated infrastructure in the form of basic software. By standardising the infrastructure functionalities such as communication stack, memory stack, diagnostic services, operating system, etc., it allows the competition in development of application software functionalities like adaptive cruise control, lane departure warning, advanced front lighting control, etc. The AUTOSAR standards are primarily focused on the overall development process. Hence, AUTOSAR standardise the software architecture specification, interfaces and a methodology for the development process. In order to achieve full benefits of such standardisation it is necessary to implement its element very systematically.

With the presence of specification and methodology process, AUTOSAR makes it possible to distribute the requirements for implementation among different supplier with assured processed and high efficiency. Today, automotive development is not confined to a single company, but each vehicle is manufactured by a group of companies together. The advantage of such distribution is the development of expertise in respective fields such as ECU hardware, basic software, application software, integration and comprehensive testing.
2. Technical overview

The Software Ecosystems (SECOs)\cite{48} for AUTOSAR consist of mainly three major stakeholders, 1) Original Equipment Manufacturers (OEM) who is responsible for coordination; 2) AUTOSAR-Tier-1 suppliers, who are responsible for developing ECUs as per OEMs requirement; 3) AUTOSAR-Tier-2 suppliers, who developed the components for Tier-1 supplier. It does not have a direct connection with OEM. The development of component by the Tier-2 supplier is based on requirements given by OEM to Tier-1 supplier and from Tier-1 to Tier-2 supplier. The quality requirement at the ECU level has to be realised at the component level. The study shows that the quality requirement more often comes in later stages of development. About 75% to 80% of AUTOSAR specific requirements are common across the different OEM projects. Hence, the Tier-2 supplier is well equipped with identifying these requirements and implementing or reusing the previous solution for fulfilling the demands, which requires the good software quality for reusability or portability.

However, 20% of these requirements specifications are very new to the developer, the Tier-2 supplier addresses these requirements using agile development process\cite{83}. The Tier-2 suppliers are also responsible for the development of test cases for validation and verification of requirement, which takes months of work. The introduction of new requirement specification needs to be re-aligning with existing ones. The consistent frontloading is very important with AUTOSSAR. For this the specification must be precise and with formal notations, which requires extra efforts from contractors\cite{81}; it can be simplified by use of tool support. To support the development there are several tools present in market such PREEvision from vector\cite{81}, which provide full support from requirement management to supplier integration. Such tools also provide quality assurance after the implementation but it lacks the assistance during system designing. The tool support is require, which will provides the objective feedback over quality such as maintainability, modifiability or adaptability.

With the introduction of AUTOSAR version 4.0, it had introduced specification of Timing Extension\cite{13}. It provides high level specification in terms of Events, Events Chains, Events Triggering, Latency and Synchronisation. For hard real time application depending upon the level of development stages multiple timing-analysis methods are available such as EAST-ADL\cite{31}. The detailed study of all such tools and methods can be found in\cite{72}, it provides tools and methods for timing analysis. With the introduction of timing extension(TADL2)\cite{86} and AUTOSAR timing extension (TIMEX)\cite{7} the standardisation of timing information exchange format is achieved, which can be used for time analysis. However, it lacks the tool support for architecture design decision making process for improving timing behaviour quality at an early stage of development.

The architecture specification in AUTOSAR has very large number of specification element, which allows very detail architecture specification. It is a challenge for the automotive industry to close the gap between requirement constraints on a system at the design level and the design decision validation and verification considering
their effective properties at the implementation level to improve the overall functional and non-functional quality.

To address all the development related issues, developers are adapting continuous integration environment. There is a need for a tool which can work in such an environment and give objective feedback about the quality of software architecture to the developer. By this tool, the impact of developer’s design decision on the software architecture quality can be monitored throughout the development process. It can also serve as an early verification and validation tool of software architecture.

### 2.3.1. Basics of AUTOSAR software architecture

![AUTOSAR Layered Architecture](image)

AUTOSAR architecture has three layers as seen in Figure 2.1, which are as follows:

- **Application layer** - which is designed to make software independent of underlying hardware.
- **Runtime environment** (RTE) - RTE separates application software from basic software. It encapsulated virtual function bus (VFB), which is responsible for communication among various components.
- **Basic software layer** - which provides an abstraction between hardware and application software, which makes application layer totally independent of hardware. The basic software layer is further divided into sub-layers named as the service layer, the ECU abstraction layer and the microcontroller abstraction layer [1]. AUTOSAR standardises the basic software modules, hence it eases the developer to concentrate...
As we know that the requirement specifications are implemented in application software as an "internal behaviour", hence the key driver of software quality is the application software itself. Some of the important features of AUTOSAR related to application software are introduced below,

**A) AUTOSAR software component** -
To attain the objective of AUTOSAR, component based software engineering (CBSE) has been chosen. It allows separation of concern not only in terms of application functionality but also in terms of infrastructure. The application layer functionalities are developed within the software component (SWC). The Figure 2.2 shows typical software component in AUTOSAR.

![Figure 2.2: Graphical representation of software component in AUTOSAR](image)

The SWC described in AUTOSAR are of two categories 1) composition software component type (CompositionSwComponentTypes) and 2) atomic software component type (AtomicSwComponentType)[1]. The composition software component type is responsible for the creation of abstraction within components. Hence, they contribute towards the implementation of model scalability. The atomic software component type, allows the actual software functionality as "internal behaviour" implementation. It is further classified into seven sub types [1] they are as follows,
2. Technical overview

1. ApplicationSWComponentType (Application software component type)- it represents hardware independent application software functionalities.

2. NvBlockSWComponentType (Non volatile Block software component type)- it represents one or more blocks of NvBlock, which is memory unit available at run time.

3. ServiceSWComponentType (Service software component type)- it represents hybrid concept between the basic software module and a software component type. Each ServiceSWComponentType provide the unique service which is required on that ECU.

4. SensorActuatorSWComponentType (Sensor Actuator software component type)- as per the layered architecture of AUTOSAR, hardware sensor or actuator are represented by this type of software component in the application layer.

5. EcuAbstractionSWComponentType (ECU Abstraction software component type)- similar to sensor-actuator it represents the input output hardware abstraction of electronic control unit (ECU) in application layer.

6. ComplexDeviceDriverSWComponentType (Complex Device Driver software component type)- it used when there is the demand of direct microcontroller access to/from the application layer. It is also used to introduce non-AUTOSAR legacy software functionalities.

7. ServiceProxySWComponentType (Service Proxy software component type)- in AUTOSAR functionality if any basic software module needs to communicate with application software present on other ECU, then in order to realise this communication, service proxy software component type is used. So for RTE it acts as atomic software component but in reality it is a proxy to AUTOSAR service.

All these software component types is visible to the virtual functional bus (VFB) and theoretically will contribute to software quality. But in the scope of this thesis we will consider only ApplicationSWComponentType and its connections to ServiceSWComponentType.

A SWC encapsulate application software functionality in one or more runnable entity (runnable), in Figure 2.2 we can see the three runnables. The runnable within an SWC can communicate among each other via an inter-runnable variable (IRV) or an exclusive area. A SWC connects with other SWCs via virtual functional bus (VFB) through ports. If a SWC is present on different ECU then VFB uses Communication Module (COM)-Signal service of BSW.

B) Ports & Interfaces -
The ports act as a communication point between one software component and the other software components. There are two kinds of ports present, require port (RPort) to receive data and provider port (PPort) to send data. Each port is associated with an interface. Interfaces are communication models in AUTOSAR. According to AUTOSAR version 4.0 and above there are total six types of interface, 1) Sender-Receiver interface, 2) Client-Server interface, 3) Calibration interface, 4) Trigger interface, 5) Mode-Switch interface and 6) Nv-Data interface [1]. AUTOSAR standardised these interfaces.

An SWC port interface describes the data element which will communicate through a port. A port with the particular interface can only communicate with a port which has same or compatible interface type. For example, a port with Sender-Receiv er interface can communicate with a port which has Nv-Data interface. The runnable uses different kinds of mechanisms to read or write data through the port. These mechanisms depend upon interface type and nature of communication. For example, in Client-Server interface, the client invokes the operation of the server component, the server processes the operation and it may or may not (depending upon the functionality) return the signal back to the client component[26].

C) Virtual Functional Bus (VFB) -
The VFB provides a design abstraction to facilitate interconnection among different ports of different SWC with a compatible interface. As this is a virtual mechanism, hence software can be developed without worrying about the underline hardware mechanism[1]. VFB also provides real-time constraint mechanism without considering target platform [57].

D) Run Time Environment (RTE) -
The RTE encapsulates the VFB by providing mechanism to fictionalised VFB at runtime [38]. RTE provide the necessary infrastructure for a SWC to be implemented without reference to hardware ECUs and it also provides integration mechanism for SWC to integrate with ECU without changing application software. The communication can be inter as well as intra ECUs communication. RTE also must arrange real-time scheduling mechanism for SWC, i.e., mapping of runnables to the specific task of the operating system.

E) Basic Software (BSW) -
AUTOSAR standardised BSW architecture in a fixed architectural modules. As mentioned earlier in this section, these modules are described in different abstraction layers. The BSW module comprises of the device driver, communication driver, service manager, etc. The functioning of AUTOSAR operating system (OS) is also the responsibility of the BSW. The BSW partially dependent upon underlying hardware, in a sense, it provides an abstraction layer for application software. Each module for its operation can communicate with another module, SWC above the RTE can not directly access interface mechanism of BSW. These BSW modules are
not in the scope of this thesis.

2.3.2. AUTOSAR methodology

As discussed earlier in section 2.3, some of the main objective goals of AUTOSAR are to provide -

1. Easy transferability of software functions.
2. Ease of integration of SWCs from different suppliers.
3. re-usability of software or hardware functionalities [64].

In order to reach these goals, AUTOSAR has standardised its architecture meta-model, and by following AUTOSAR standards and methodology for developing this standardised architecture, it wishes to achieve above goals.

Figure 2.3.: Overview of AUTOSAR methodology

The AUTOSAR methodology does not precisely describe complete process of development but rather describe the dependencies of activity as common steps and their work-product. This type of approach allows the consistency in development process across the industry. Figure 2.3 shows the different activities of AUTOSAR methodology.

A) System configuration input -
In the first step of methodology, system configuration input is created. This step is based upon identifying software components, hardware components and overall system constraints. This is done by taking architectural decisions and filling it in appropriate templates standardised by AUTOSAR.

B) Configure system -
In configure system step, application SW C are mapped to ECUs. In this step, basically allocation of resources and timing requirement aspects are taken into consideration. This is one of the most crucial steps considering the quality of the overall system, because in this step most important design decisions are made during this steps [64]. The work-product of this step is system configuration description.

C) System configuration description -
This file includes information regarding overall system such as SWCs to ECUs mapping, bus topology, system communication matrix, network information, etc. The implementation of A,B,C step is done in an iterative manner, i.e during development phase if any new system constraints requires to be added then first two steps can be repeated and C stage results can be generated. This facilitates agile development environment, where system design can be improved and evolved over the project development cycle. This type of methodology is very much suited for continuous integration environment. The thesis scope explicitly focuses on this stage result.

D) ECU specific file generation -
In next steps, extraction of ECU specific information takes place. Then in further steps, the configuration of BSW modules, mapping of a runnable entity to the task, task scheduling, etc. And then finally, the ECU executable is generated.

The result of each step is transferred via Extensible Markup Language (XML) format. In AUTOSAR terminology, they are mentioned as ARXML (AUTOSAR Extensible Markup Language) files. As per the AUTOSAR standard, any AUTOSAR development tool must deal with ARXML file format for either importing or exporting system (model) information [14].

2.3.3. AUTOSAR ARXML file features
As mentioned earlier in section 2.3.2, ARXML files are used for the designing system, its configuration, its transportation and storing the information. This facilitates reusability of SWC from one system to another, also it is possible to develop the system as separate SWC modules and share it via ARXML file [95]. Generally, in any given development environment there are two types of ARXML files are present, one which relates to a specific software component and other which comprises of the overall system including all the SWC information. For example, information related to any one SWC can be described separately, independent of any other SWC. This information includes ports, interfaces, runnables, etc. This allows
2. Technical overview

reusability of SWC from one system to another. The separate SWC ARXML file, then can be integrated with system specification ARXML file. The system specification ARXML file holds the information regarding SWCs as well as overall system such as communication matrix, the composition of the system, ECU configurations, etc.

In the scope of the thesis, we are working with system specification ARXML file. This choice is made because of the wider scope of the system view, but even individual SWC ARXML file can be treated as input. Figure 2.4 shows typical SWC ARXML template.

```xml
<xml version="1.0" encoding="utf-8"/>
<!--Created with SystemDesk 4.3.1.7975-->
<AR-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>Swc</SHORT-NAME>
        <AR-PACKAGES>
            <SHORT-NAME>SWC_Package_Name</SHORT-NAME>
            <ELEMENTS>
                <APPLICATION-SW-COMPONENT-TYPE>
                    <APPLICATION-SW-COMPONENT-TYPE>
                        <SHORT-NAME>SWC_Name</SHORT-NAME>
                        <PORTS>
                            <INTERNAL-BEHAVIORS>
                                <SWC-INTERNAL-BEHAVIOR>
                                    <SHORT-NAME>SWC_Internal_Behaviour_Name</SHORT-NAME>
                                    <CONSTANT-VALUE-MAPPING-REFS>
                                    </CONSTANT-VALUE-MAPPING-REFS>
                                    <DATA-TYPE-MAPPING-REFS>
                                    </DATA-TYPE-MAPPING-REFS>
                                    <EVENTS>
                                    </EVENTS>
                                    <IMPLICIT-INTER-RUNNABLE-VARIABLES>
                                    </IMPLICIT-INTER-RUNNABLE-VARIABLES>
                                </SWC-INTERNAL-BEHAVIOR>
                                </INTERNAL-BEHAVIORS>
                            </APPLICATION-SW-COMPONENT-TYPE>
                        </ELEMENTS>
                        <AR-PACKAGES>
                            </AR-PACKAGES>
                        </APPLICATION-SW-COMPONENT-TYPE>
                    </APPLICATION-SW-COMPONENT-TYPE>
                    </APPLICATION-SW-COMPONENT-TYPE>
                    </APPLICATION-SW-COMPONENT-TYPE>
                </ELEMENTS>
            </APPLICATION-SW-COMPONENT-TYPE>
        </AR-PACKAGES>
    </AR-PACKAGE>
</AR-PACKAGES>
</AUTOSAR>
```

Figure 2.4.: ARXML file of SWC

Using SystemDesk 4.3 modelling tool, application software component has been created, which is then exported in the form of ARXML file. The ARXML file is comprised of packages and elements structure. These packages and elements are same as XML file format but AUTOSAR describes them in its own terminologies. The information related to any specific aspect of SWC is defined under (in specific package) particular container name, for example SWC input output ports are defined by PORTS container. AUTOSAR standardised these container names and
revised them as per the new versions.

For example, prior to AUTOSAR revision version 4.X, application software component is defined as APPLICATION-SOFTWARE-COMPONENT-TYPE, while from 4.X it is defined as APPLICATION-SW-COMPONENT-TYPE. To read any specific information from file it is required to have knowledge about the container name in which that information is present. The quality evaluation tool which relies upon ARXML file elements information, it should have ability to consider these changes in naming format as per the AUTOSAR file version. Table 2.1 shows some of the containers name and their information,

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION-SW-COMPONENT-TYPE</td>
<td>Application SWC</td>
</tr>
<tr>
<td>SERVICE-SW-COMPONENT-TYPE</td>
<td>Service SWC</td>
</tr>
<tr>
<td>PORTS</td>
<td>Ports of a SWC</td>
</tr>
<tr>
<td>INTERNAL-BEHAVIOUR</td>
<td>Behaviour description of SWC</td>
</tr>
<tr>
<td>EVENTS</td>
<td>Runnable triggering mechanism</td>
</tr>
<tr>
<td>INTER-RUNNABLE-VARIABLES</td>
<td>Communication among runnable</td>
</tr>
<tr>
<td>SENDER-RECEIVER-INTERFACE</td>
<td>Sender receiver interface</td>
</tr>
<tr>
<td>CLIENT-SERVER-INTERFACE</td>
<td>Client server interface</td>
</tr>
<tr>
<td>ECU-INSTANCE</td>
<td>ECU definition</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Map logical to physical system</td>
</tr>
<tr>
<td>SWC-TO-ECU-MAPPING</td>
<td>Map SWC to ECU</td>
</tr>
<tr>
<td>CONNECTORS</td>
<td>Connection between ports</td>
</tr>
</tbody>
</table>

Table 2.1.: AUTOSAR container names and its description

2.4. Summary

In this chapter overview of the automotive powertrain domain is presented. The chapter also outlined the current automotive software architecture traits such as AAF, ADF, ADL and AFAS. However, the AUTOSAR reference architecture is not use to describe the entire automotive system architecture, but it provides standardised architecture elements for describing implementation view of software component. These architectural elements are used to understand a software architecture. The brief study of AUTOSAR architecture and its development methodology is presented for better understanding of the thesis. The chapter also helps in identification of the AUTOSAR architectural elements for the architectural quality measurement and evaluation.
3. State-of-the-art

The main objective of architecture quality evaluation is to get objective feedback about the current quality of software and to improve the architecture design. Hence, in order to achieve this, first, we must understand different quality attributes and how we can measure these attributes. In the first section 3.1, we will understand the industry norms for software measurement process and the challenges faced by the automotive industry in implementing this processes. Then in second section 3.2, we will establish the quality definition and will discuss the industry standards for deriving quality goals. Then in next section 3.3, we will take an overview of the different architecture evaluation methods that exist and analyse the suitable technique for given scenario of the automotive industry. Finally, in section 3.4, we will do a critical analysis of the work done by different authors in this field of architecture evaluation.

3.1. Software measurement process

The software measurement is basically done for managing system, software life cycle activities, determining the feasibility of project plans and ability to adhere to propose project plan [8]. There exists an international standard for software measurement called ISO/IEC 15939. As shown in the Figure 3.1, the standard classifies the measurement process in four activities,

1. Establish and sustain measurement commitment
2. Plan the measurement process
3. Perform the measurement process
4. Evaluate measurement

Most importantly, these activities are in an iterative cycle allowing continuous quality feedback to the management (stakeholders).

The first activity involves the commitment from stakeholder and identification of the scope of measurement. Then in the second activity, needed resources, methods of extraction of these resources, rational for evaluation, required permissions and necessary tools are identified and planned. In the third activity, actual measurement takes place, and their results get communicated to appropriate stakeholders.
Finally, based on evaluation criteria the software is evaluated, and possible improvement suggestions are given as feedback to the stakeholders [8]. To perform these activities, the company needs to allocate dedicated team and resources. The intend to quality measurement tool must adhere to measurement process concept. Hence critical analysis is required in order to adapt its activities in the toolchain.

To the establishment of such measurement process in today’s automotive industry will be very productive. Although, performing it in an iterative cycle will require proper planning and fixed system constraints. There are several challenges while performing measurement process such as,

1. Measurement process requires the stakeholder to set up the measurement goal; this restricts the coverage of goal to the ability of the stakeholder in understanding the system [97].

2. The measurement process is derived from software artefact and process. In distributed development environment, the software artefact used by the measurement process may get changed with a third-party artefact, which may have different artefact or process. For example, measurement process designed for system configuration description level cannot be directly used for ECU configuration description level of AUTOSAR methodology. Also, measurement process designed for non-AUTOSAR based software development process cannot be used for AUTOSAR based software development process. This will require new measurement plan and rework [97].
3. State-of-the-art

3. As discussed earlier in section 1.1.2, in the automotive industry which is focused on reducing development time and cost. The measurements at the architecture stage will increase the effort for the developers at early stages and requires stakeholders to be assigned the assessment team to measure the architecture quality. This additional workforce will contribute to the additional cost to the project.

4. In agile development scenario where requirement specification and the subsequent system design evolve with time, tracking the impact of requirement and architectural changes is difficult. This increases the efforts for the measurement process.

Hence, to cope up with the demand of automotive industry for evaluation of software quality implementation of such measurement process is necessary. Although, considering the demand of reducing development time and cost for the automotive product, the prospect of measurement process lies in tool based automatic process. While developing the measurement toolchain for architectural quality calculation, all the above factors need to be considered.

3.2. Definition of software quality

The quality, in general, is a degree of excellence. Hence software quality is also not exception to this, according to international standard ISO/IEC 25010:2011 software quality can be stated as,

\[\text{Degree to which a software product satisfies stated and implied needs when used under specified conditions.} \quad [4]\]

Depending upon stakeholders, software quality can be classified into two broad aspects, functional quality and structural quality [25]. The functional quality gives the degree to which the software complies with functional requirement and specification. It basically deals in functional behaviour correctness, and it is measured by performing tests on the software. The structural quality basically complies with the non-functional requirement of software, i.e. it deals with the structure of software itself. The quality attribute belonging to this category are testability, maintainability, understandability, portability, etc. While functional quality can easily be measured with appropriate test cases, it is difficult to measure structural quality. Although in an early stages of development where software is not fully developed, it is hard to identify functional quality, however, non-functional quality measurement can give objective feedback to the developer, which enables then in architectural decision making [35]. In the scope of this thesis, structural software quality is measured.

The ISO 25010 classifies eight major quality characteristics and several sub characteristics. However, not all these characteristics rely on software architectural design.
Apart from the ISO 25010, software architecture quality is also influenced by several other standards and guidelines. Such as ISO 26262, this is an international standard for the functional safety of electrical and electronic systems in the automotive domain. It is used as a guideline for the development and the verification of software architecture. The standard describes the quality attribute such as encapsulation, modularity, simplicity, freedom from interference, testability, etc. [2].

The Table 3.1 shows the automotive safety integrity level (ASIL) described by ISO 26262 for the architectural development process.
Additionally, the architectural design is driven by certain guidelines, which recommends ways to formulate a system. These guidelines are based upon critical analysis of architectural patterns, their future consequence on the system and the best practices in the industry. AUTOSAR has specified modelling guidelines for powertrain domain application based on current practices in powertrain domain[12]. While designing metric tool, these guidelines must be taken into consideration. As AUTOSAR architecture concept is relatively very new, considering the history of the automotive sector, hence most of the architectural concepts are very new to the field. AUTOSAR has specific concepts like RTE (Real-Time Environment), VFB(Virtual Function Bus), Complex Device Drivers, Ports, Delegation and Assembly connectors which are specific to the architecture designed by the consortium. The availability of such concept has allowed the developer to specify the architecture very precisely and as per the requirement specification. For example, in non-AUTOSAR domain, the component is connected via ports with the common interface type, but starting from AUTOSAR architecture, design decision maker has more than one option available to them like Sender-Receiver interface, Client-Server interface, etc. Hence, the impact of available architecture elements on architectural quality still needs to be investigated thoroughly. Expert knowledge is required to arrive at a design decision in the AUTOSAR domain based on the knowledge of the architectural concepts and elements. Hence, the expert suggestions must be taken into consideration while designing a tool to analyse the architecture quality.

### 3.3. Architecture evaluation methods

In previous chapters 2, the importance of software quality has been established. The impact of software architecture on its quality was studied. The challenges faced by the industry in architectural decision making were also discussed. An architecture
development involves tradeoffs and adherences to design guidelines. The extent to which architecture adheres to design guidelines specified by AUTOSAR is not verified. Hence, architecture needs to be evaluated. Based on the stage of development, an architecture evaluation can be possible in early or later stages of development. Early architecture evaluation will minimise the risk of design failure and so the rework. Moreover, the quality goal can be better realised if architecture evaluation happens in early stages of software development [69]. The early architecture evaluation is based on requirement specification and its description; moreover, considering agile development environment newly added functionality to the software can impact the architecture. The extent to which the architecture fulfills the quality requirements needs to be evaluated after these changes. Additionally, as discussed earlier chapter 1 section 1.1.2, an automotive industry has very high requirement of shorter time to market and lower cost of production. Hence key requirement from evaluation technique should benefit more from the evaluation of project than its implementation cost [9].

The software architecture evaluation is a very well established topic; several methods exist for architecture evaluation [78]. Evaluation methods can be divided into two broad categories, 1) Questioning techniques and 2) Measuring techniques [36]. The questioning technique involves qualitative analysis based on scenarios, questionnaires, and check list, in short, it is scenario-based architecture analysis. While the measuring technique involves quantitative measurement analysis based on metrics, simulations, etc. [36] each of these techniques are discussed below.

### 3.3.1. Scenario-based evaluation method

The scenario-based evaluation investigates software architecture in systematic methods using particular scenario. The results of this investigation show whether or not the software architecture will be able to deliver its required functionality for that scenario [69]. Among other methods of architecture evaluation, the scenario-based method is very well evolved [71]. There are several scenario-based evaluation methods that exists, commonly used scenario-based methods are shown in Table 3.2

<table>
<thead>
<tr>
<th>Evaluation method</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Architecture Analysis method (SAAM)[55]</td>
<td>Maintainability</td>
</tr>
<tr>
<td>Architecture-Level Modiﬁability Analysis (ALMA)[19]</td>
<td>Modiﬁability</td>
</tr>
<tr>
<td>Cost-Beneﬁt Analysis Method (CBAM)[30]</td>
<td>Costs, Beneﬁts, Schedule Implications</td>
</tr>
</tbody>
</table>
3. State-of-the-art

The scenario can be developed as per the specific demands of the stakeholders for the evaluation of specific quality requirement from the specific software architecture, i.e. scenarios are built for the particular architecture only. If the right scenario is developed then it could capture all user/use-cases specific specification requirements, different quality aspects attributes and their associated weighting that the architecture needs to attain. Hence using such scenario-based architecture technique is very useful in automotive industry. However, it has some challenges as well they are as follow,

1. In the process of developing scenario, stakeholder, and scenario development team must have very good experience and skills in order to achieve full possible architecture analysis coverage [78].

2. The scenario developed for one architecture cannot be directly used for another architecture analysis.

3. Today software development process is very agile in nature thus requirement from software are subject to change which in turn may bring changes in architecture, interfaces, algorithm, etc. So the scenario also needs to be changed as per the newly introduced requirement, but developing scenario takes a significant amount of time and effort.

4. The scenario-based development is a very subjective process, and reproducibility of the result is very challenging task [78].

The main objective of this thesis is to produce software architecture quality assessment in short time and iterative manner considering the continuous integrating developing environment. Thus, scenario-based assessment does provide this option to stakeholder as scenario development process is very difficult to automate.

3.3.2. Metric-based evaluation method

The metric-based evaluation is based on quantitative measurement of architecture quality. These techniques address the specific questions on software quality of architecture [36]. Generally, metric-based evaluation method is termed as "Late Evaluation Method", as it is deployed over developed software architecture for functional or non-functional quality assessment [78]. However, a tool-based assessment of the software architecture can only be possible if the inputs are provided in a certain format. Hence, the architecture needs to be developed before the assessment is performed. In the scope of this thesis, early software architecture evaluation will be done by measuring non-functional quality goals of software architecture for providing objective feedback of software architecture quality. The term early is used considering V-Model of automotive software life cycle phases, where architecture design
3. State-of-the-art

comes in early stages of software development. In late software architecture evaluation, implemented data is available for architecture evaluation, whereas in the early software architecture evaluation the software description elements data is available.

The according to IEEE standard 1061 [6] software quality metrics can be defined as,

"A function whose inputs is software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality."

The standard also provides a methodology for developing software quality metrics. The methodology is a systematic process which comprises of five steps as given below:

1. **Establish software quality requirements** - this step involves identification of quality goal which is possible and required to be extracted from given software architecture.

2. **Identify software quality metrics** - this step identifies the metrics which can be used to quantify previously stated quality goals. This step is also concerned with the feasibility of metrics in given environment regarding cost-benefit analysis.

3. **Implement the software quality metrics** - this step involves identification of source artefact and description of the procedure for information collection. As a result of this stage, the metric results are generated.

4. **Analyse the software metrics results** - in this step the interpretation of result takes place. Depending on the metric value the quality of software component is determined. Then the investigation of anticipated tolerance intervals is identified for further study (i.e. evaluation of quality) [43].

5. **Validate the software quality metrics** - the metric validation is very vast subject. The separate toolchain is dedicated for evaluation of metric results [43]. The evaluation toolchain performs empirical validation and evaluation of results, based on statistical analysis of metric results from different projects.

With systematic and objectively analysing the metric results, metric can be validated [62]. The metrics which are well documented and known to the developer are considered to be valid. For example, the quality Testability which is dependent on a number of input signals to SWC, so metric number of input signal representing this quality can be considered to be validated metric.

---

1Described in the previous chapter 2
3. State-of-the-art

The direct metrics whose results are directly based on a property of architectural elements or guidelines are validated with expert opinion. While complex metric is validated by correlation analysis among contrast quality of software component, such as maintainability will be reduced if complexity is more, so if component showing higher complexity quality while poor maintainability should be considered as valid results hence metric is valid. Further, the validation can even be possible by experimental analysis, for example, quality value testing can be analysed with actual testing time for software component and if relation among its quality value matches then metric can be validated [6].

There are several metric derivation approaches are presents, in the scope of this thesis Goal-Question-Metric (GQM) [67] has been adapted. The GQM is a systematic method for defining and evaluating software quality, using measurement. In the GQM methodology, the quality goal is identified and defined, then the quantifiable question addressing the required quality goal for given software artefact is formulated. Based on the question and available information of software artefact, the mathematical expression as a metric is formulated. The metric will provide quantitative data for propose question, in turn, quality goal interpretation. Figure 3.3 shows example for GQM based metric derivation for quality simplicity [2], the metric was derived for information technology software scripts [49].

![Diagram of GQM based derivation of metric for simplicity.](image)

Figure 3.3.: GQM based derivation of metric for simplicity.

### 3.3.3. Requirement from metric

As discussed in previous section 3.3.2, the fact that the there is need to evaluate software architecture continuously. Hence, the metrics have an advantage over the
3. State-of-the-art

scenario-based methods in terms of the fact that they can track changes in architecture using a data-driven approach. Moreover, the metric should have some basic characteristics such as,

1. It should be simple and precisely defined to give accurate, objective feedback [62].

2. It should not be too sensitive or too robust; it should be able to respond according to software changes in an appropriate manner [96]. i.e., it should not respond to insignificant changes in the process or product.

3. It should be validated with its objective, i.e. it should be producing correct results for intended quality goal [62].

4. It should be able to orient itself to software product line goals and automotive standards [91].

Moreover, metric should have some basic functional, performance and economic qualities. The functional quality can be reliability, repeatability, maintainability, etc.; performance based quality such as response time, throughput, etc.; and it should be economically viable for the organization to use those metrics [62].

3.4. Architecture evaluation literature review

The software measurement using metrics is a very mature topic. Many authors have published various metrics model. These metrics varies depending upon quality goals, source artefact, the domain of artefact, etc. Table 3.3 shows some of the prominent metrics which are analysed for this thesis.

There are very few metrics which are proposed for automotive domain architecture evaluation; most of the given metrics are used in the information domain. The metric [33] [80] is designed for automotive domain but it uses Simulink model as source artefact. The metric [93] [92], is also designed for automotive domain but is based on general architecture specification. The metric [85] is based on AUTOSAR based architecture; it proposes the estimation of software components development effort in the early stage of development. The estimation is based on ISO 19761 [3] standard. It specifies its usages for quality analysis but does not provide any metric.

Müller et al. had proposed early AUTOSAR based architecture evaluation methodology based on scenario-based evaluation technique [63], which does not satisfy our automated requirement. Apart from that there is some work regarding early software testing by verification or validation methods [31] [74] [37], but these research does not provide feedback regarding architecture quality itself. During literature survey for AUTOSAR based architectural quality evaluation, there was no research work found for quality measurement using metric-based evaluation methods.
### 3. State-of-the-art

<table>
<thead>
<tr>
<th>Metric</th>
<th>Quality</th>
<th>Source artefact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupling, Cohesion, Nesting metrics [33]</td>
<td>Modularity</td>
<td>Simulink model</td>
</tr>
<tr>
<td>Model based metrics [80]</td>
<td>Efficiency, Correctness, Robustness, Testability, Understandability, Maintainability</td>
<td>Simulink model</td>
</tr>
<tr>
<td>Component balance, Dependency profile [21]</td>
<td>Analysability, Dependency</td>
<td>script</td>
</tr>
<tr>
<td>Module size uniformity [79]</td>
<td>Modularity</td>
<td>API based script</td>
</tr>
<tr>
<td>Model Halstead volume [46] [52]</td>
<td>Complexity</td>
<td>Simulink model</td>
</tr>
<tr>
<td>Coupling between component [27] [93]</td>
<td>Coupling</td>
<td>Architecture specification</td>
</tr>
<tr>
<td>Input interface complexity [92]</td>
<td>Testability</td>
<td>Architecture specification</td>
</tr>
<tr>
<td>Function size measurement [85] [84]</td>
<td>Effort</td>
<td>AUTOSAR TargetLink model</td>
</tr>
</tbody>
</table>

Table 3.3.: Existing metric for different quality goal.
3. State-of-the-art

3.5. Continuous integration framework

In current scenario automotive industry is moving towards agile development [77], this is mostly because of two reasons,

1. Distributed developing environment.
2. Continuously evolving requirement from stakeholders.

In CBSE, the development work is divided among different developers; they develop the system into pieces (units/SWC) and integrate as a whole in continuous integration framework. Secondly, in most of the cases, the requirement specifications from OEM to the supplier grows over the project development phase, this is mostly because of improper communication of requirement from various stakeholders [83]. To adapt this evolving and improving design demand, developer again has to rework on the system. This kind of scenario can be easily handled by continuous integration process. Figure 3.4 shows the basic structure of continuous integration framework.

![Diagram](image_url)

Figure 3.4.: Basic structure of continuous integration framework [77].

Using CBSE, each software component can be developed and tested separately. In an agile environment, continuous integration can be adapted for quality assessment of the software. In continuous integration framework, the system software is developed in a distributed manner. The data is stored in a central repository in which the
developer can access project data. In certain phases of a development, project stakeholder can trigger the continuous integration server to perform the specific operation from available tools. The result of this operation is sent back to the developer as feedback. These operations can be Unit Test, Quality Measurement, etc.

3.5.1. Quality measurement toolchain

Based on the idea for early architecture evaluation, FEV GmbH has developed the quality measurement toolchain [93]. The toolchain is intended to help data driven software architecture evaluation. The architectural information is stored in semi formal description, mainly consistent with company’s internal guideline. Figure 3.5 shows the metric measurement toolchain.

The quality measurement toolchain is triggered by stakeholder via continuous integration framework. The first step of the toolchain is gathering of software architectural related information. Then, the Dependency Structure Matrix (DSM) is
created. The DSM is dependency matrix between various software components. Depending upon dependency matrix (DSM) various quality measurement is calculated. The results of the earlier stages are stored appropriately in visualisation stage. This visual and data results are available for stakeholder for appropriate actions.

The toolchain only accepts the input file format based on company’s internal software architectural guideline. Hence it cannot be easily used for third-party specific software architectural format. To resolve this issue, additional customer specific plug-in needs to be created. Also, toolchain lacks the metric for the project based on standardised architecture such as AUTOSAR.

3.5.2. Evaluation toolchain

Once the stakeholder has metric measurement result report, a stakeholder can get an objective feedback regarding the software architecture. However, this result does not reflects the severity of the result. For that, thresholds for each metrics need to be evaluated. The evaluation toolchain determines these threshold values for each project [43]. Figure 3.6 shows the evaluation toolchain.

![Quality Evaluation Toolchain Diagram](image)

Figure 3.6.: Quality evaluation toolchain [43].
3. State-of-the-art

The stakeholder triggers the evaluation toolchain via continuous integration server. The metric measurement results of different projects are transferred to the evaluation toolchain. Then, based on cumulative technique the threshold value for each metric is calculated. Considering project type (rapid or series production project), project maturity and legacy of a project each metric get scaled, this is called metric prioritisation. Then the average metric mark over entire project metric values is determined. The average of all individual metric mark is calculated to be a global mark; this global mark represents the overall quality value of the project. Once all quality threshold mark is calculated, then the individual SWC metric value is evaluated. Finally, the evaluated results of an each project is visualised by appropriate methods and saved in a repository for stakeholders analysis.

The evaluation toolchain require multiple projects for quality thresholds determination. It is preferential to have multiple projects of similar nature for good threshold determination, which limits its usage to the single available project type.

3.6. Summary

The chapter presents the study of methodologies for developing the software quality measurement tool. In the chapter requirement from a measurement tool as per the ISO/IEC 15939 standard outlined. For identification of non-functional software quality, the study of quality goals based on the standards such as ISO 25010 and ISO 26262 is conducted and the quality goal results were presented. The quality goals also derived from the AUTOSAR modelling guideline. The architecture evaluation techniques such as scenario-based evaluation and metric-based evaluation are studied, and concluded that the metric-based architecture evaluation is most suited for early-stage architecture quality evaluation. The literature review of existing research in metric-based evaluation of AUTOSAR based automotive architecture is studied. The work done by various authors in the field of metric-based quality evaluation was outlined. However, no direct research in metric-based early architecture evaluation was found for AUTOSAR domain. The tool requires to function as part of existing toolchain of the company. The overview of existing toolchain is also provided.
4. Solution

In chapter 3, we have discussed the quality model and techniques for metric derivation. In this chapter, we identified the quality goals specific to AUTOSAR architecture discussed in chapter 2. Section 4.1, we outlined the proposed quality model. Then in section 4.2, we have illustrate the Goal-Question-Metric (GQM) list for quality measurement. The GQM list is used to derive metrics, which will give us quantitative values for proposed quality goals. To satisfy the demand of quality evaluation in the continuously evolving environment as discussed in chapter 1, we have integrated the proposed metrics in a toolchain, this toolchain is discussed in section 4.3.

4.1. Proposed quality model

Based on the first research question (RQ1) the identification of quality goals based on the architectural artefact is established in this section. As discussed in the previous chapter 3, quality goals are influenced by available architectural artefact, but rational on which these goals are established is based on different standards and modelling guideline. There are international standards present such as ISO 25010 [4], ISO 26262 [2] which we have discussed in the previous chapter 3 section 3.2, but not all the quality goals mentioned in these standards can be realised at architecture level. For example, quality goal Accessibility, which is the degree to which a system can be used by the user to achieve specific goal [4], such goal is influenced by the characteristic of software and hardware at integration level hence we cannot be able to extract its quality at the architecture level. Apart from quality standards, AUTOSAR provides guidelines for modelling software component, however, within powertrain domain, some additional compatible assumptions are outlined. These assumptions are used to make powertrain domain application software more compatible, safety critical, understandable and tailored as per its domain requirement. With careful study of various architectural aspects of AUTOSAR, the quality goals are identified. The quality model is developed considering all influencing factors. Figure 4.1 shows the tailored quality model specific to available AUTOSAR architectural information.
4. Solution

Figure 4.1.: Quality model for AUTOSAR based powertrain software architecture [4]

4.2. Implemented GQM list

This section will provide understanding for second research question (RQ2). The analysis of quality goals and architecture concluded with the definition of the metric for providing quantitative feedback over quality. Based on the quality model of previous section 4.1, the architectural artefacts are mapped to quality goals. For AUTOSAR based automotive project, the ARXML file is the source artefact for software architecture evaluation. A metric is defined based on the detailed analysis of the architectural parameters. The Goal-Question-Metric (GQM) [67] approach is followed to derive metrics. Depending upon architectural abstraction level, the metric can be classified into three categories - 1) Runnable level, 2) Software component level, 3) Composition or system abstraction level. This considers various viewpoints of the architecture. At runnable level, the implementation of functionalities is performed in a Simulink /Targetlink model or as direct C code, here information regarding functional Meta-Model can be studied. As explained in section 2.3.3, architecture artefacts are present in containers; each container has some tag for example SYMBOL[1]. In ARXML file "SYMBOL" provide function name of this C file. However, its implementation is not in the scope of ARXML itself. At SWC component level, the interaction between SWC as well as the internal component structure can be studied. The SWC component level includes -
4. Solution

- Runnable
- Inter-runnable variable
- Exclusive area
- Port interface
- Port
- Communication connections within SWCs
- Event
- Data type
- and so on...

At system abstraction level overall system parameters are considered which includes, composition structure of the system, SWC mapping, signal mapping, etc. In the scope of this thesis, SWC level and some of the system abstraction level metrics has been designed. However, data is also made available for runnable level for SWC metric analysis and future metric implementation.

4.2.1. SWC level based GQM

The metrics in this section deal with the quality evaluation with respect to the software component. The behaviour of the SWC, such as runnable entities\(^1\), runnables triggering events, inter-runnable variables, exclusive areas, etc. are studied, and GQM list has been prepare. For the sake of simplicity, only a few of the quality goals and their resulting metrics are discussed here, but similar to these rest of the quality goals and artefacts are analyse, and respective metrics have been developed.

Table 4.1 shows the list of quality goals and the metrics at SWC level. These metric are part of toolchain.

\(^1\)From here on it is refer as runnable/runnables
### 4. Solution

<table>
<thead>
<tr>
<th>No.</th>
<th>Goal</th>
<th>Question</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Simplicity, Analysability, Time behaviour</td>
<td>How many runnables are present in SWC?</td>
<td>Number of Runnable</td>
</tr>
<tr>
<td>2.</td>
<td>Simplicity, Modularity (cohesion), Time behaviour</td>
<td>How many inter runnable variables are present in SWC?</td>
<td>Measure of Inter-Runnable Variable</td>
</tr>
<tr>
<td>3.</td>
<td>Simplicity, Maintainability, Time behaviour</td>
<td>What is an average number of operation usage per runnable in SWC?</td>
<td>Measure of Average Client-Server Interactions</td>
</tr>
<tr>
<td>4.</td>
<td>Time behaviour, Maintainability</td>
<td>How many Exclusive Areas are present in SWC?</td>
<td>Measure of Exclusive Area</td>
</tr>
<tr>
<td>5.</td>
<td>Maturity, Analysability, Modularity</td>
<td>What is a number of runnable entities which are periodically triggered in SWC?</td>
<td>Measure of Periodic Trigger Runnable</td>
</tr>
<tr>
<td>6.</td>
<td>Maturity, Simplicity, Time behaviour, Modularity</td>
<td>How much runnable entities are triggered by communication event in SWC?</td>
<td>Measure of Communication Dependency</td>
</tr>
<tr>
<td>7.</td>
<td>Modifiability, Functional completeness, Interoperability</td>
<td>How many unconnected input ports are present in SWC?</td>
<td>Percentage of Unconnected Report</td>
</tr>
<tr>
<td>8.</td>
<td>Analysability, Compatibility</td>
<td>How many numbers of ports use the same interface in SWC?</td>
<td>Interface Re-usability Index [12]</td>
</tr>
<tr>
<td>9.</td>
<td>Portability</td>
<td>What is the ratio of implantation reference of platform type to total implementation reference in SWC?</td>
<td>Ease of Integration [38]</td>
</tr>
<tr>
<td>10.</td>
<td>Testability</td>
<td>What is input and output dependency of SWC?</td>
<td>Measure of Dependencies [92]</td>
</tr>
</tbody>
</table>

Table 4.1.: Developed metric at system software component level for different quality goal

The Goal-Question-Metric approach is concerned with the process of identifying quality goal which can be mapped with the help of architecture and quantified by metric value. The definition of metric is an detailed process as explained in section
3.3.2 which involves,

1. Detailing of quality goals.

2. Derivation of questions that identify the quality attributes in the software architecture.

3. Identification of architectural parameters

4. Identification of software artefacts containing the architectural parameters

An example of the detailed process for a set of quality criteria (simplicity, timing, and testability) is explained below. Similarly, all the quality criteria listed in the Table 4.1 and 4.2 were evaluated.

Goal (Simplicity) - Consider quality goal Simplicity, it is a degree to which interconnection among different module are present [4].

Question - To derive the questioned study regarding architecture itself should be carried out. Hence considering AUTOSAR architectural Meta-Model, the behaviour of SWC is realised in the runnable entity of components. The communication link coming from various components through RTE are connected to SWC externally via ports, within software component via various interfaces to the runnable. Although, AUTOSAR has introduced the concept of RTE to minimize structural complexity i.e. the connection between various component are taken care by RTE allowing ease of placement of components anywhere in the system, yet connection within SWC component has to be realised by the developer. For example, data element coming from single RPort can be used by two separate runnable, the communication between runnable is carried out by inter runnable variable or shared exclusive areas, events for triggering of runnable, etc. Also, not all ports are connected to the other SWC, and the used interface can have multiple operations or data elements, out of which only those get used which is required by the runnable. Hence, in AUTOSAR structural complexity may be reduced to some extent, but internal complexity is still dependent upon the internal connection within SWC.

If SWC has only one runnable then all the connection links are attached to a single runnable entity. Hence, internal interconnection will be more or less equal to the external interconnection, although because entire SWC functionality has been placed within the single runnable entity, its internal behaviour would be complex with poor maintainability [93]. On the other hand, if a number of runnable entity is more then the functionality of SWC is distributed, and each runnable gets more maintainable, but the interaction among them in the form of inter runnable variable, exclusive areas, triggering events could increase and so will its complexity resulting poor quality value for simplicity. So, the conclusion can be derived that runnables have an effect on SWC simplicity quality value.
So, the question should be asked such that it will provide objective feedback regarding concern quality.

1. How many runnables are present in SWC?
2. How many inter runnable variables are present in SWC?
3. What is an average number of operation use per runnable in SWC?
4. How much runnable entities is triggered by communication event?

**Metric** - The metric will provide quantitative values for the questions,

1. Number of Runnables, i.e. Number of runnable in SWC
2. Measure of Inter Runnable Variable, i.e. Number of inter runnable variable present in SWC
3. Measure of Average Client-Server Interaction, it is ratio of total number of client and server operation used in SWC to the total number of runnable
4. Measure of Interaction Links, i.e. input dependency, output dependency and inter runnable communication.

The particular architecture artefact can also be used for evaluating more than one quality attribute. In this example considering quality goal ”Time Behaviour”, where architecture element runnables is again used for quality measurement.

**Goal (Time Behaviour)** - Time behaviour is defined as the degree to which the processing and response time of a system meets the requirement specification while performing its function [4]. In the context of AUTOSAR, timing behaviour of the functionalities depends upon execution of the task on ECU. The study requires which architectural parameter influence execution of functionalities.

**Question** - The timing analysis in powertrain domain is very important activity, because of the safety criticality of its functions. Starting from AUTOSAR revision 4.1, AUTOSAR has introduced the specification for timing analysis [13]. In order to determine architecture elements which influence the timing properties, the architecture study is carried out. The timing properties such as task execution time, bus latencies, and sensor sampling rates.

As we discussed earlier, runnable being the smallest unit in SWC possess executable code. Each SWC contains one or more number of runnables. In order to execute runnable functionalities on the base operating system, they are mapped to operating system task by ECU configuration process. This mapping is later used by RTE for scheduling and execution of runnable. Depending upon the runnable implementation there are two categories of runnable are possible 1) basic runnable entities and
4. Solution

2) extended runnable entities. The basic category runnable has finite amount of time for its execution and termination. So the runnable entity which does not contain 
Wait Point are of this category, and they are mapped to basic operating system 
task (basic task), for example, runnable with periodic events (Timing Events[1]) 
is of this category. The extended category runnable contains waiting point, which 
causes runnable to wait for certain Events (associated to wait for point) to occur, a 
runnable entity which has sporadic events (communication events) are generally (if 
they have wait point) of this category. If multiple extended categories are mapped 
to a single task and if they wait for the event to occur, it will cause the timing 
(jitter) problem for other runnable entities in the task.

The architect has to make task scheduling design decision by considering various 
parameter of runnable, such as if runnable is periodic then what is trigger timing 
of runnable? Because if runnable with different execution event time period 
is assigned to the same task then the separate counter mechanism is required for 
correct execution of runnables. Another decision can be if periodic and sporadic 
event runnable are assigned to the same task, then the task becomes extended, and 
there are several other scenarios, which makes task extended and task scheduling 
get complex to handle. The optimisation of task scheduling is a very challenging 
subject and many authors have proposed solution in this field [75] [101] [98]. This 
work discusses interaction among runnables and their assignment to the task can be 
optimised for secure execution of runnable.

According to ARXML component template client component and server component 
interact with each other via Client-Server interface [1]. It is possible to have same 
SWC to be a client for some component as well as a server for another component. 
The Client-Server functionalities are realised using runnable entity. When the SWC 
require server operation functionalities, it assigns one of its runnable as a client by 
introducing Server Call Point[1]. On server SWC, the runnable entity is assigned 
as a server, get invoke by Operation Invoke Event and execute operation (function) 
and return the result to client SWC. Depending on communication call sequence 
the communication is divided into two types Synchronous communication[1] and 
Asynchronous communication[1]. In synchronous, client entity functioning gets sus-
pended till the server complete the execution of the operation. It again resumes 
when the server returns the results. Where as in asynchronous client continues to 
execute after making a request to the server. When the server returns, the opera-
tion results in client get it by either polling port or Asynchronous server return 
call event[1]. As synchronous communication directly returns request, not ex-
tra functionalities are required. whereas asynchronous uses system resources to 
monitor the return result from server hence it is complex in nature. In each of the 
client request suspension time is specified as Time Out[1] it is the responsibility of 
the developer to ensure finishing of operation execution within specified time frame 
[26]. While scheduling the runnable entity to the task, its time behaviour needs to 
be taken care of.
4. Solution

The time behaviour of a SWC can also be affected by use of inter-runnable variables with multicore processor [64]. The powertrain application are generally concurrent task with hard real-time constraints. Usually, the software is deployed on single core ECU for each application, hence concurrency is easily achievable with no competition for shared resources, as single task executing at any given moment on ECU. In multicore processor the multiple task of same application can coexist on same ECU with trying to access the same shared resource. When different runnables of same SWC deployed on different task with having shared memory resources such as inter-runnable variable, then for executing their operation will try to compete for resource access. As a consequence, delayed in execution of operation of SWC application, called temporal interference [41]. Because of the interference in execution the time required of completion runnable functionality varies and so does the execution of task. To minimise this various scheduling mechanism have been suggested [41], it revolves around coordination among cores, memories and cashes. Hence when using multicore processor, more the inter-runnable variables present in the SWC (single application), the complexer will be the coordination mechanism with higher chances of temporal interference.

The inter-runnable communication using exclusive areas involves sharing of a resource among runnable, which are critical section and protected by pre-emption techniques. It uses OS resource (spin locks, interrupts, etc.) to provide atomic control of the resources to the runnable. If any runnable with a particular task (TaskA) wish to take an exclusive area which is already locked by another task (TaskB), then this scenario may result in delayed response to requiring task (TaskA) activations by OS. As a result, temporal interference can happen causing execution delay of task. It is the responsibility of the developer to make sure that execution timing of runnable for using exclusive area are optimised correctly so that task starving can be avoided. While designing the software priority of the task for entering the exclusive area needs to be analyse carefully. Otherwise, response time will get increase, which is very sever for safety critical applications.

At network level, the network protocol (e.g. CAN, LIN, FlexRay), the size of frames, transmission pattern, etc. affect the time behaviour [13]. The inter-ECU communication depends upon Communication Module (COM) signals. The COM module connects the SWC signals to a bus such as Control Area Network (CAN) bus. The COM signals drive a number of messages or signals traffic that can be transmitted on the network and the communication mechanisms for transmission. The improper configuration of network brings the latency in the network and so as in function execution. The developers have to make correct design decisions for setting the network configuration so that latency in the network can be avoided. This decision depends upon ECU resources, the number of COM signals [1], the number of COM group signals[1].
4. Solution

Hence we can conclude that the time behaviour is affected by various architectural factors such as, a number of runnables in SWC, triggering mechanism of this runnables, communication between runnables i.e. if it is inter-ECU or intra-ECU communication, a number of inter-runnable variables when using multicore processor and presence exclusive areas among runnables.

Considering all these factors the question can be proposed at architecture,

1. How many runnables are present in SWC?
2. What is the ratio of a number of tasks on which SWC runnable Map to Number of the runnable maps?
3. How many runnables are triggered by sporadic (communication, trigger, etc.) events?
4. How many runnables are with synchronous or asynchronous communication?
5. How inter-runnable variables are present in SWC?
6. How many exclusive area present in SWC?
7. How many COM signals are present in the system?

Metric - The metric will provide quantitative values for these question,

1. Number of Runnables, i.e. a number of runnable in the SWC.
2. Ratio of Runnable to Task, i.e. a ratio of total number of SWC to a total number of task on which they are assigned.
3. Measure of Communication Dependency, i.e. a number of entities which are triggered by communication event.
4. Measure of Synchronisation or Measure of Asynchronisation, i.e. a number of synchronous or asynchronous communication.
5. Measure of Inter-Runnable Variable, i.e. a number of inter-runnable variables present in the SWC.
6. Measure of Exclusive Area, i.e. a number of exclusive areas presents in the SWC.
7. Measure of Inter-ECU Communication, i.e. a number of COM or COM group signal available.
Let us look at another quality goal example Testability.

**Goal (Testability)** - The testability is given as, it is a degree of effectiveness and efficiency with which test criteria can be established for a given system, and the ability to perform tests to determine whether those criteria have been met [4].

**Question** - The testing of software is done for verification and validation of requirements. The testing can be done by the method of static test or dynamic test. The static test can be code review, coding rule checks, etc., tools and process are available which can perform these test for AUTOSAR software [60]. The dynamic test is carried out for verification of software temporal and functional behaviour; it involves white box testing and black box testing. The white box test is done considering internal structure of the component. The black box test is concerned with the software behaviour and its functionalities. Generally, the dynamic testing process is carried out in three steps [50]

1. The first step involves identification of functional data (black box) and its test cases as per the specification requirement, generation of the test case and its expected results.
2. In the second step, setting up the test object, execution of test and determination of test coverage and evaluation of result.
3. On the basis of data generated, identify the further test cases and execute them by repeating step 1 and 2.

In above steps, the developer have to develop the test cases based on developed functionality, specification document, requirements, industry norms and so on [89] [65]. The proposed test model for early stage testing requires simulation of VFB, for software-in-loop testing [54]. The test developer uses the available interface to the component for the development of a test for AUTOSAR application software component [70]. Hence at the architecture level, the input output connection to SWC can be the key to understand the effort for test development. However, connection to SWC will not provide connection to functional behaviour because the functional behaviour is constituted in runnables and test are developed for functional behaviour requirement. We need to look at input and output dependency to the runnable entities in order to establish a relation between testability and architecture elements.

The input dependencies can be input communication links from outside of SWC to all runnables present within SWC. The input communication links can be communication events which trigger the runnable such as *Data Receive Event, Asynchronous Server Call Returns Event, Operation Invoked Event*, etc. [1], it can also be receiving data communication link to runnable; for example, explicit data read access such as *(Data Receive Point By Value or implicit data read access such as *(Data Read Access)[1]). In Client-Server interface communication, the client SWC will make
4. Solution

request to server SWC for execution of operation by either by *Synchronous Server Call Point*[1] or *Asynchronous Server Call Point*[1] depending on upon type of communication, and the request response will be received by same port from which has client made request or by another port it depends upon mechanism for reception, so this will provide input and output dependency to SWC. As timing signal is not provided as part of the interface, but for execution of functionality they are provided by BSW, and as part of VFB simulation for testing it also gets considered for test cases [54]. Hence, we must take timing events such as *Time Event or Background Event*[1] as part of input dependencies.

Similarly, output dependencies can be, the response from *Operation Invoke Event*[1], it can be sending data communication links from runnable; for example, explicit communication such as *Data Send Point*[1] or implicit communication such as *Data Write Access*[1]. As explained above, for Client-Server communication, *Synchronous Server Call Point*[1] or *Asynchronous Server Call Point*[1].

It is also important to note that, communication events which are not passed externally but internally triggered such as *Internal Trigger Event*, also affect the granularity of test cases as these are driven by the internal behaviour of runnable. Hence, considering the above requirement for testing and contribution of architecture elements, the question can be proposed at architectural level for testability goal,

1. How many runnable entities are triggered by another entity in SWC?
2. What is an average number of arguments per operation used in SWC?
3. How many modes are present in SWC?
4. What is input and output dependency of SWC?

**Metric** - The metric will provide the quantitative value for the question,

1. Measure of Functional Behaviour Dependency, i.e. number of internally triggered events trigger the runnable functionalities.
2. Measure of Average Arguments, i.e. number of arguments present per operations.
3. Measure of Modes, i.e. number of modes of operation present in SWC.
4. Measure of Dependencies, i.e. product of input and output dependencies of SWC.
4. Solution

4.2.2. System based GQM

The system based GQM represents the overall architectural quality of a system. Table 4.2 shows the architectural metrics at compositional or system abstraction level. The powertrain software has very high safety critical aspect to it. In previous chapter 3, safety requirement of ISO 26262 standards were discussed. In the quality model, these requirements are realised in quality goal simplicity. One of the requirements is reduce size of software component, at system abstraction level such quality can be realised.

<table>
<thead>
<tr>
<th>No.</th>
<th>Goal</th>
<th>Question</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.</td>
<td>Time behaviour, Simplicity, Resource Utilisation</td>
<td>What is the number of inter-ECU communication channel?</td>
<td>Measure of Inter-ECU Communication</td>
</tr>
<tr>
<td>12.</td>
<td>Simplicity</td>
<td>How many SWC to SWC connections are present in the system?</td>
<td>Number of Assembly Connector</td>
</tr>
<tr>
<td>13.</td>
<td>Simplicity</td>
<td>How many SWC to composition connections are present in the system?</td>
<td>Number of Delegation Connector</td>
</tr>
<tr>
<td>14.</td>
<td>Simplicity</td>
<td>How many application SWCs are present in the system?</td>
<td>Measure of Application Software Component</td>
</tr>
<tr>
<td>15.</td>
<td>Portability</td>
<td>How many Complex-Device-Driver SWCs are present in the system?</td>
<td>Measure of Complex-Device-Driver</td>
</tr>
<tr>
<td>16.</td>
<td>Simplicity, Analysability</td>
<td>How many modes are present in the system?</td>
<td>Number of Modes in System</td>
</tr>
</tbody>
</table>

Table 4.2.: Developed metric at system abstraction level for different quality goal

Goal (Simplicity) - The goal is realisation of safety requirements such as,

1. Separation of concerns [2]
2. Reduced size of software components [2]
3. Reduced interface size [2]
4. Realisation of a functionality[2]
Simplicity allows the software component to be tested and maintained in an easy way.

**Question** - The powertrain software is logically divided into many software compositions and components. As the number of software component increases the handling of these software components becomes very difficult. The verification and validation of this software system become very challenging. Hence ISO 26262 in part 6 has made the requirement of restricted size of software component[2]. However, a standard does not specify the number of software components allowed in the system. From the assertion of logic behind the requirement, the argument has suggested that the number of software components should be such that it can be maintainable by developer considering the satisfaction of rest of the standard goal. However, if two software components have similar functionality (interconnection linkage), then they can be grouped together to form a single component. Hence, the software developer has to satisfy safety requirement as well as make design decision of formation of component or fusion of different functionality into the component and for this, he must know the current status of the software component in the system. The question can raise as following,

1. What is the number of SWC in composition?
2. What is the number of application SWC in composition?
3. What is the number of sensor-actuator SWC in composition?
4. What is the number of service SWC in composition?

**Metric** - The metric will provide quantitative values for these question,

1. Measure of software component, i.e. number of SWC in composition irrespective of their type.
2. Measure of Application Software Component, i.e. the number of application SWC in composition.

### 4.3. Developed Measurement toolchain

To address the third research question RQ3, the toolchain is developed. The toolchain is designed as per the measurement process requirement [8] and following methodology provided by IEEE standard 1061 [6]. Figure 4.2 shows the state flow of developed toolchain. As discussed earlier the AUTOSAR architecture specification is present in system description ARXML file. The path of this file is needed for architectural data extraction. Once the data is parsed dependency among various component is get identified, and the matrix is created (DSM). Depending upon parsed information
4. Solution

and DSM metrics is get calculated, and their result is get stored for further analysis. The detail explanation of the toolchain is discussed in next chapter 5.

Figure 4.2.: State flow of AUTOSAR quality measurement toolchain.

4.4. Summary

The chapter provides the solution for proposed research questions. After studying the architectural artefacts, the quality model is proposed. To measure the proposed quality characteristics metrics were derived. For the metric derivation GQM
method is used. Depending upon the abstraction level of architectural elements, the proposed GQM list is classified for metrics at software component level and composition/system abstraction level. Using the example of simplicity, time behaviour and testability, the metric derivation process is demonstrated. As per requirement of continuous integration framework, the AUTOSAR measurement toolchain is developed.
5. Implementation

In this chapter the detail implementation of toolchain is presented. The implementation of toolchain is carried out according to the measurement process standard [8] as explained in section 3.1. In first section 5.1, parsing process of the ARXML file is explained and its output results are outlined. In second section 5.2, the dependency matrix creation and its usages are mention. In third section 5.3, the metric implemented in toolchain are presented. And finally in section 5.4, the integration of developed toolchain into continuous integration framework is presented.

5.1. Database structure of parsed ARXML file

The AUTOSAR architecture description of any given project is transferable from one developing an environment to another via ARXML file [14]. This file contains architectural information of the system architecture. It means all SWCs architecture level information is present in this file. The file is referred as system description file. It is possible to have an individual file for each SWC, but individual SWC file does not contain information about their interconnection with other SWC. Hence for this reason the system description ARXML file has been chosen. Figure 2.3 shows typical ARXML file. Logically, the file is divided into two parts, first part contain SWC information such as ports, interfaces, runnables, events, etc. and the second part contains information regarding the integration of these components with one another.

As per the measurement process standards explain in section 3.1, the first step of evaluation toolchain should be the identification of quality goals, which we have discussed in previous section 4.1. The second step would be the identification of architecture artefact related to those goals. To identify the architectural artefact several meeting with developer and domain expert were carried out. During a discussion with the developer aspect related to current state of the ARXML file were understood. For example, during the meeting, it was realised that the project was developed by 3 different supplier. Hence, referencing methodology for the ARXML package were different and tool was adapted as per demand. By repeating this process the architectural artefacts were identified.

The company’s ARXML file structure was developed as per the development specification of RTE development tool (RTA-RTE version 5.6.0) [38]. These specifications influence the architecture design. For example, in the ARXML file when a port is cre-
5. Implementation

ated it can optionally provide communication specification which will determine port behaviour. This assignment is done by providing communication specification sub package. The provider port will have PROVIDED-COM-SPECS [1] (provider COM specification) and require port will have REQUIRED-COM-SPECS [1] (required COM specification). This enables the usages of various communication attributes such as transmission acknowledgement, initial values, allocation of buffer size, filters, etc. These attributes help in reducing the load of the system. Let us understand this by an example of filter attribute. Providing filter attribute to a port for detecting the data only when it has a specific property. Such kind of attribute is useful when runnable entities are triggered by the DATA-RECEIVE EVENTS [1], by specifying filter property as detect when a change in receive value. By this filter, the amount of events which are triggering the runnable entity can be reduced. Moreover, because of filtering the event trigger occurrence, the OS time consumed in processing there signals which are not deemed useful can be saved. Hence, the developer will consider such attributes while developing the software component. However, the present version (RTA-RTE 5.6.0) of RTE development tool does not provide specification attribute for client port i.e. CLIENT-COM-SPEC [1] (client COM specification) is not used for this version. Hence while establishing measurement process, one should have knowledge of what architectural artefact can be present in ARXML file. The tool should also be flexible enough to adapt new changes considering another version of RTE development tool.
5. Implementation

By studying the development tool specification parser was developed. As shown in Figure 5.1 the parser has two stages. In the first stage of the parser, it convert the ARXML file into the Matlab structure, in this no information is extracted, but the format of the file is changed, from XML to structure (MAT). This change in the format provides ease of information handling. In the second stage, different architectural artefacts are parsed. This parsing process is made as modular as possible to enable adaptation of evolving architectural needs of developer or stakeholders and improve maintainability of the tool. As per the requirement specified in section 3.1, the changes in AUTOSAR version package tags are considered, and version elements library is created. This version elements library store the package tag as per the AUTOSAR version, at present this enables tool in AUTOSAR version revision 4.0, revision 4.1 and revision 4.2. Moreover, the new version can be enabled by adapting their feature to version library. Also, as currently information is extracted from system specification ARXML file, it can be possible that the project is in early stages and system specification file is not generated yet. Hence, to enable architectural quality evaluation at this stage additional functionality is provided for extracting information from individual SWC ARXML files. By this the system level quality metric cannot be detected, but individual SWC quality metric can be measured.
5. Implementation

The results of the parser is in structure format which has following information as mentioned in Table 5.1.

<table>
<thead>
<tr>
<th>Parser database modules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECU DB</td>
<td>ECU configuration [1] information</td>
</tr>
<tr>
<td>ECU Instance</td>
<td>ECU Instances [1] information</td>
</tr>
<tr>
<td>Systems</td>
<td>Software component and data mapping [1] information</td>
</tr>
<tr>
<td>Component Interface</td>
<td>Assembly connector [1] information</td>
</tr>
<tr>
<td>Delegation Interface</td>
<td>Delegation connector [1] information</td>
</tr>
<tr>
<td>Composition Layer</td>
<td>Composition layers and containing SWC information</td>
</tr>
<tr>
<td>COM signals</td>
<td>COM signal [1] present in system</td>
</tr>
<tr>
<td>COM Group signals</td>
<td>COM group signal [1] present in system</td>
</tr>
<tr>
<td>Components DB</td>
<td>Parsed individual SWC</td>
</tr>
<tr>
<td>System Modes</td>
<td>Modes [1] information per composition</td>
</tr>
</tbody>
</table>

Table 5.1.: Table depicting parser results.

5.2. Creation of dependency structure matrix

As per the AUTOSAR design specification, each port of SWC is supposed to be optional [12], this allows reusability of SWC from one project to another with using only relevant ports for another project. For example, the project requires the only partial functionality of already developed component; hence not all ports will be used, but only those which are associated with required functionality are connected to another component. The connection among different SWC within the same composition is connected using ASSEMBLY-CONNECTOR [1] while the connection between SWC and its composition is connected using DELEGA TION-CONNECTOR [1]. The dependency structure matrix (DSM) presents the input and output dependency of SWCs in tabular form [93]. The Table 5.3, shows the DSM matrix for component structure 5.2, it shows that component A is dependent on component B and C, and dependency among them is 5 and 1 respectively. The dependency structure matrix (DSM) presents the input and output dependency of SWCs in tabular form. This allows identification of connected SWC ports and associated architecture artefacts. Along with DSM matrix, some other matrix and information table is also get generated. The Table 5.2 enlist results of the DSM generation stage.
5. Implementation

<table>
<thead>
<tr>
<th>DSM stage result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSM</td>
<td>Dependency matrix between SWCs contains port connection names</td>
</tr>
<tr>
<td>DSM count</td>
<td>Dependency matrix between SWCs contains number of share signal</td>
</tr>
<tr>
<td>Unassigned signal</td>
<td>List containing unassigned ports per SWC</td>
</tr>
</tbody>
</table>

Table 5.2.: DSM stage results.

The DSM results allow the developer to keep track of development process and change in structural complexity. During metrics analysis process, having such matrix helps the developer to identify the weak spot in architecture. For example, consider component dependency as shown in Figure 5.2, after studying the metric "Component Coupling" one may find that component A is dependent on component B and component C. But metric does not show us the number of connection among them. Hence, by referring to DSM matrix developer can understand the links among different components. In Figure 5.2 we can notice that the SWC B, and the SWC A have very high interconnection, hence if these two components combine to form a single SWC, then input dependency to the newly formed component will be three i.e. complexity will be reduced. Also, the new component will be very modular. Hence quality modularity [4] will get increase as a by-product. The DSM matrix and other results will provide complementary assistance to architecture quality analysis.

<table>
<thead>
<tr>
<th>Component Names</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.3.: DSM for component structure presented in the Figure 5.2.

Figure 5.2.: Dependency of SWC example.
5.3. Derived metrics

In section 4.2, we have established the GQM list. In this section, derivation of metrics mathematical formula and its inference is discussed.

The GQM list discussed in section 4.2 has only defined metrics for software component level and system level. However, there is also the metric table generated for runnable level architectural characteristics. The Figure 5.3 shows the runnable level architectural characteristic table.

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Runnable Name</th>
<th>DataSendPoint</th>
<th>SyncPoint</th>
<th>ModeAccessPoint</th>
<th>ModeSwitchPoint</th>
<th>ExclusionArea</th>
<th>InterfaceName</th>
<th>InterfacePeer</th>
<th>ReadPeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>re_INTER_AWP2_100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ARM</td>
<td>re_MSG_AWP2_100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AccP</td>
<td>re_AccP_Run_MED</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AccP</td>
<td>re_AccP_Run_MED</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AccP</td>
<td>re_AccP_Run_MED</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sync</td>
<td>re_Sync_Run_MED</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sync</td>
<td>re_Sync_Run_MED</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sync</td>
<td>re_Sync_Run_MED</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sync</td>
<td>re_Sync_Run_MED</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sync</td>
<td>re_Sync_Run_MED</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5.3.: Runnable level architectural characteristics.

Similar to DSM matrix, this table also provides assistance in quality analysis. The table contains the information of architectural elements of runnable entities and presents this information at component level abstraction. The architectural elements such as data send point, synchronous server call point, mode access point, mode switch point, explicit exclusive area, etc. [1], are presented in this table. The table information is provided to the stakeholders in the form of file. The stakeholder can access this file along with the metrics results to analyse which elements of architecture contributing the metric values. For example, consider component Accelerator Padde (AccP) which has one runnable re_AccP_Run_MED as shown in the Figure 5.3. The third column presents the data send point of runnable. The cell associated to runnable entity will contain information regarding data send point such as the data elements, the port name, the interface name and the type of com-
5. Implementation

communication; for each data send point communication link available to the runnable.

5.3.1. Metric expression and inference

In section 4.2, as part of GQM list, the questions and associated metrics for required quality goal have been proposed. In this section, their mathematical expressions are discussed. For each metric, we have described the mathematical formula and its results inference.

No.1: Number of Runnable

Quality :- Simplicity, Analysability, Time Behaviour

Formula :-

\[
Number\ of\ Runnable = \text{TotalRunnablePerSWC} \tag{5.1}
\]

Here, \(\text{TotalRunnablePerSWC}\) is total number of runnable entities [1] present in the SWC. The equation 5.1 represents number of runnables presents in the SWC.

Background :-
As explained in section 2.3.1, the SWC’s functional requirements are decomposed into smaller functional unit called runnable entities. The runnables inside any SWC are group together because they have functional cohesion, and to realise their required functionality they may need to interact with each other. The more number of runnables present in the SWC increase liklability of the number of interconnections among each other, which increases the complexity. The runnable entities also need to satisfy the time decomposition of the functional behaviour [13]. This is realised by associating the event to runnable entity, which is responsible for triggering of functionality for execution of its behaviour, and the execution of behaviour holds the property such as response time (the time it was taken by function from activation till termination) or execution time (the time require for function to complete its assigned behaviour). Each runnable is triggered by an event, which may be a periodic event or sporadic event as explained in section 4.2. The runnable entities are assigned to the ECU task for execution. Depending upon runnable functionality and triggering mechanism the tasks are scheduled. It is the responsibility of the developer to schedule tasks in such a way that they carry out the given functionality without any hindrance.

Design consequence :-
From analysability perspective, if the SWC has only one runnable it suggests that the functionality of SWC is realised in a single entity, which makes difficult to comprehend the entire component functionality, while too many runnables in SWC distribute the functionalities in across many smaller modules, making it complex system to understand [21]. From simplicity perspective, if runnable have more interaction it makes the SWC functional behaviour complex to understand resulting the poor quality of simplicity. From time behaviour perspective, as number of runnable entities increases in SWC, the design decision require for scheduling these runnable
5. Implementation

to task also get increase. This also increases the chances of wrong scheduling and so as it affects time behaviour. While scheduling runnable decision such as execution time, response time, triggering mechanism, communication mechanism and relation among other runnables are need considered.

Recommendation :-
The lower the metric value will increases the analysability of SWC. According to the expert understandability of the SWC functional behaviour is low for a component having more than two runnables, while less than two makes the component difficult to analyse. To improve simplicity, the metric value should be less. This can be achieved by fusing the runnables which having more interaction links (inter-runnable variable or exclusive area). To improve the time behaviour, the metric value should be lower.

No.2: Measure of Inter-Runnable Variable
Quality :- Simplicity, Modularity (cohesion), Time behaviour
Formula :-

\[
Measure \text{ of Inter Runnable Variable} = TotalIRV
\] (5.2)

Here, TotalIRV is total number of inter-runnable variable [1] present in software component. The equation 5.2 represents number of inter runnable variables present in SWC.

Background :- In SWC the inter-runnable variables are used for the exchange of information among runnables. The SWC with more inter-runnable communication than inter-component communication indicates cohesive functional behaviour [93]. With the introduction of multicore AUTOSAR operation, it may be possible to place an SWC runnables on different tasks in different cores [64].

Design consequence :- From simplicity perspective, more the inter runnable variable more will be communication links between runnable, which increases the complexity. According to safety experts for safety critical application it is better to design software component with less inter-runnable variables. From modularity perspective, more the number of inter runnable variables indicates that the SWC have very strong cohesion, which means the component is modular. From time behaviour perspective, the runnables with inter-runnable variables placed on different cores can introduce the temporal interference in execution if their execution time is not matched or if they dependent on some external events [41]. As explained in section 4.2.1, the developer needs to make critical coordination analysis for correct scheduling.

Recommendation :- To improve the simplicity, the number of inter-runnable variables should be lower. To improve the modularity, the number of inter-runnable
5. Implementation

should be higher. By having less number of inter-runnable variables while using multicore will improve the developer scheduling ability, and so will be the timing behaviour. The interdependent runnable should be places of single core, so that the resource management can be conducted correctly.

No.3: Measure of Average Client-Server Interactions

Quality :- Simplicity, Maintainability, Time behaviour

Formula :-

\[
\text{Measure of Average Client-Server Interactions} = \frac{\sum_{i=1}^{N} \text{SerCallPt}_i + \text{OpInvkEv}_i + \text{AsyncSerReCallEv}_i}{\text{Total number of runnables}}
\] (5.3)

Here, \(\text{SerCallPt}\) is server call point [1], \(\text{OpInvkEv}\) is operation invoke event [1], and \(\text{AsyncSerReCallEv}\) is asynchronous server return call events [1]; and the summation of all such parameter present in all the runnable \(N\) of SWC. The equation 5.3 represents the average number of client-server interface communication links present per runnable in SWC.

Background :- The Client-Server communication used for utilisation of AUTOSAR services. The communication link is based on execution of operations by the server component which usually represents the basic software module in the application domain. Each interface has one or more associated operation (execution function), and each operation will have 0 or multiple arguments [1] [38]. Moreover, it is also possible to have multiple clients making a request to the same server, in that case, either concurrency operation is triggered, or if concurrency is not activated then the request is stacked, then it is the responsibility of RTE to return the result back to requested client runnable entity. As explained in section 4.2, the Client-Server linkage depends upon execution time of operation.

Design consequence :- As the number of Client-Server linkage in the runnable increases the execution of each runnable becomes very complex. For safety critical application with real-time constraint requires quick response from server operation, and if server has stacked client request then the execution of client gets delayed. According to AUTOSAR application developer experts, application software should have less number of Client-Service communication linkages to minimise software complexity. As the server component has very cohesive linkages to multiple client components, the change in server component will affect to various client components. The time behaviour of the software component is also affected by Client-Server communication.

Recommendation :- To improve the simplicity the metric value should be lower, especially for safety critical application should have less number of Client-Server linkages. To improve the maintainability of the component the metric value should
be lower. The less number of client linkages will allow developer to manage and modify SWC considering all the client specific timing and functional needs. For client component less server linkages will have better execution response time without any jitter. While for server component, less client linkage will reduce sporadic behaviour of SWC. It will increase SWC predictability. Hence, to improve time behaviour of SWC this metric value should be lower.

No.4: Measure of Exclusive Area  
**Quality :-** Time behaviour, Maintainability  
**Formula :-**  
\[
\text{Measure of Exclusive Area} = \frac{\text{Total ExclAr}}{\text{runnables}} \tag{5.4}
\]

Here, TotalExclAr is total number of implicit or explicit exclusive areas [1] in SWC. The equation 5.4 represents the average number of exclusive area used per runnable in SWC.

**Background :-** The inter-task and inter-runnable communication are protected with the help of exclusive areas. This could be shared memory which needs to be protected; it is OSEK\(^1\) critical section. The pre-emption mechanism is provided for a safe period for resource utilisation since one task might be writing on to the shared resource while another task might try to reading it. The OS resources are used to establish this critical section. The access to given area is provided based on the priority of runnable or depending upon scheduling mechanism.

**Design consequence :-** As explained in section 4.2, exclusive area influences the timing behaviour of the software component. If the exclusive area is set by using interrupts or by OS resources, then they increase the latency and requires high ECU resources. This is because, the access to resource needs to be disabled and enable the every time runnable wish to enter or exit an exclusive area. The runnable which get block for using the resource its functionality is affected, it waits till the resource get available. Because of the delayed execution of the runnable functionality latency may get introduce in the form of temporal interference in the task execution. In safety critical application response time is very important, and delay respond may results in sever problem. Moreover, the improper use of exclusive area may also results in deadlock.

Although, for safety critical application the exclusive area provide protection over the resources, in that case, critical precaution needs to be taken for developing runnable entity functionality which uses the exclusive area. The precaution is necessary for smooth execution of all tasks which use the same resources so that the execution of any given runnable entity will not starve the other runnable entities from the usage of the resources. This requires component with good the maintainability quality. However, if the usage of exclusive area increases it makes SWC more

\(^1\)Open Systems and their Interfaces for the Electronics in Motor Vehicles
5. Implementation

complex and time critical, new changes in SWC functionalities requires extra precaution from the developer. This reflects in a poor maintainability of SWC.

**Recommendation**: To improve the time behaviour metric value should be lower. The lower number of exclusive area will reduce the complexity for the runnable task scheduling. To improve maintainability metric value should be lower. The low metric value will allow developer to modify runnable easily.

**No.5: Measure of Periodic Trigger Runnable**

**Quality**: Maturity, Analysability, Modularity

**Formula** :-

\[
\text{Measure of Periodic Trigger Runnable} = \sum_{i=1}^{N} \text{PeriodicRunnable}\_i
\]  

(5.5)

Here, \(\text{PeriodicRunnable}\) is number of runnable \(N\) in SWC which are triggered by timing event or background event. The equation 5.5 represents the number of runnable entities which are triggered by either timing event or background event.

**Background** :- The executions of such runnable entities are in a very scheduled manner, i.e. the chances of jitter in task execution can be controlled easily by controlling runnable functional behaviour. The behaviour of SWC is dependent on internal time triggering, which make the component modular.

**Design consequence**: From maturity of the functional behaviour perspective, the behaviour of time triggered entity can be predictable. The functionality of such component can be designed to produce reliable results. From analysability perspective, the better functional behaviour predictability will increase the stakeholder’s ability to understand the functional behaviour of an SWC. From modularity perspective, functional dependency of SWC for its fulfilment of requirements on the other SWC is minimised.

**Recommendation**: To improve the maturity of an SWC the value of this metric should be higher. For analysability, higher the value of this metric better will be the SWC analysability. For better modularity of SWC, this metric value should be higher.

**No.6: Measure of Communication Dependency**

**Quality**: Maturity, Simplicity, Time Behaviour, Modularity

**Formula** :-

\[
\text{Measure of Communication Dependency} = \sum_{i=1}^{N} \text{SporadicRunnable}\_i
\]  

(5.6)

Here, \(\text{SporadicRunnable}\) is the number of runnable entities \(N\) in the SWC which are triggered by sporadic communication events such as,
5. Implementation

1. Data receive events [1]
2. Asynchronous server call return events [1]
3. Operation invoke events [1]
4. Data send complete event [1]
5. Data write complete event [1]
6. Mode switch event [1]
7. Mode switch acknowledgement event [1]
8. Inter trigger event [1]
9. External trigger event [1]
10. Data receive error event [1]

As opposed to metric 5.5, this metric 5.6 presents the sporadic behaviour of the SWC.

Background: - In the case of a runnable with sporadic triggering events, the behaviour of SWC is dependent upon other SWC. To enable the functional behaviour of runnable entity the triggering signal (Event [1]) needs to get provided by other SWC, which requires the communication links between two components. The SWC functionality is reactive in behaviour which react to sporadic external events. Because of the sporadic nature of events in such SWC, the runnable entities of such components are mapped to extended tasks.

Design consequence: - From maturity perspective, the functionality result of the SWC is dependent upon goodness in the functionality of other SWC runnables, from which triggering signals comes in. For example, if a SWC is waiting for event to occur and due to failure in other SWC triggering event never occurred. After finishing of wait time runnable get suspended and functionality execution is negatively affected. From modularity perspective, the dependency on other SWC reduces the SWC’s modularity. From simplicity perspective, excessive connection links for events contribute to the complexity of SWC. From timing behaviour perspective, the higher the value of this metric require more scheduling design decision, which puts the excess burden on the developer for making a scheduling decision, as consequences it compromises their scheduling ability. As explained in section 4.2.1, the incorrect scheduling of the sporadic events task can bring jitter in the task execution, which will deteriorate the time behaviour quality.

Recommendation: - To improve the maturity this metric value should be lower. The SWC which requires good modularity it should have lower this metric value.
5. Implementation

For SWC which requires good simplicity value, this metric value should be minimum. To assure good time behaviour this metric value should be lower.

No.7: Percentage of Unconnected Report
Quality :- Modifiability, Functional Completeness, Interoperability
Formula :-

\[
\text{Percentage of Unconnected Report} = \frac{\text{TotalUnconReport}}{\text{TotalReport}} \times 100
\]  

Here, \(\text{TotalUnconReport}\) is the number of unconnected required port in SWC, \(\text{TotalReport}\) is total number of required port present in SWC. The metric 5.7 presents percentage of utilised required port in SWC.

Background :- When the assembly connector or delegation connector does not have a link for the required port of a component then it is supposed to be unconnected port. Such type of unconnected port situation usually happens when there are multiple suppliers involved in system development. The metric also reflect the future extension of component functionality, i.e. the component may require to implement the additional functionality in its future release.

Design consequence:- From functional completeness perspective, it may be possible that requirement specification is not yet completely built, this means the functional behaviour of the component may change. Alternatively, it may be possible that the customer does not require some of the functionality of existing legacy component. Hence, the ports related to those functionality are not get utilise. From modifiability and interoperability perspective, for additional changes in SWC, maintainability should be high for this component. According to experts, the unconnected ports increases complexity during the integration because the separately developed component has a probability of miscommunication in specification requirement, which is important for system integration.

Recommendation:- The low metric value suggest that SWC has good functional completeness. This metric will provide an overview of project’s progress, which will help project manager for planning development process. In some cases, it is possible that the developer has implemented port functionality directly as a C code in its internal function. For example, NV block [1] memory communication needs to be passed to NV component using NV-Data interface. However legacy NV software component can directly write to NV memory block as C function. Such kind of functionality violates the AUTOSAR architecture norms, and it is difficult to implement modelling standards in such components. Hence, to improve the modifiability, functional completeness and interoperability, the metric value should be lower.

No.8: Interface Re-Usability Index
Quality :- Analysability, Compatibility

76
5. Implementation

Formula :-

\[
\text{Interface Re-Usability Index} = \frac{\text{TotalRport} + \text{TotalPport}}{\text{TotalInterface}}
\] (5.8)

Here, \(\text{TotalRport}\) and \(\text{TotalPport}\) is the number of connected required ports \([1]\) and provider ports \([1]\) respectively in SWC, \(\text{TotalInterface}\) is total number of unique interfaces presents in SWC. The metric 5.8 presents re-usability of interfaces present in SWC. The ratio of total number of ports in component to the interfaces (unique Client-Server, Sender-Receiver, etc.) shows that how many interfaces are reused for specifying ports specification.

**Background :-** As per the general AUTOSAR modelling guideline the reuse of one interface for multiple ports is encouraged \([15]\). However, for powertrain domain separate design recommendations are provided, which specifies that the port should have 1:1 relationship with interface \([12]\). The rational for such modelling recommendation is that, upto AUTOSAR version 3.1 the ports were part of SWC which were not standardised, only port interfaces were standardised for the usages, and staring from version 4.0, \textit{PortPrototypeBlueprint}\([1]\) allows the standardisation of ports. Even though the AUTOSAR revision 4.3 is introduced, the older version is still in practice.

**Design consequence :-** From analysability perspective, in previous version of AUTOSAR, the standardisation of application interface used to involve involuntary creation of \textit{SwCoponentType}\([1]\), which creates unnecessary complexity \([10]\). Considering distributed development of system, compatibility of SWC is very essential. Also the table of application interfaces \([11]\) does not support interconnection between compatible interface with different short names \([12]\).

**Recommendation :-** To avoid complexity and improve analysability in safety critical applications such type of interface standardisation was used to be avoided in the powertrain domain. To improve the quality of interoperability and compatibility, the value of this metric should be closer to 1 for application SWC.

**No.9: Ease of Integration**

**Quality :- Portability**

Formula :-

\[
\text{Ease of Integration} = \frac{\text{TotalPlatformTypeRef}}{\text{TotalBaseType}}
\] (5.9)

Here, \(\text{TotalPlatformTypeRef}\) is total number of implementation data type \([1]\) which refers to platform type \([1]\) for base type \([1]\) and \(\text{TotalBaseType}\) is the total number of base type (native as well as non-native declaration type \([1]\)) present in SWC. The metric 5.9 presents the degree to which SWC is hardware independent.

**Background :-** The AUTOSAR revision 4.0 specifies 3 main types of data types which are Application Data Type, Implementation Data Type and Base type \([1]\).
The application data type is used in function development. The implementation data type acts as typedef in generated C code, application data type references to implementation type. Moreover, the base type describes the data type regarding hardware (encoding, memory alignment, size in bits), they form a basis on which implementation data type is built.

**Design consequence:** From portability perspective, setting the attributes of the base type, the semantic of implementation data type can be configured to any given hardware platform. The AUTOSAR specifies the platform type[1], which is implementation data type, with base types defining the same semantics across the different platform. The higher value of this metric suggests that the most of the implementation data type is having compatibility with different hardware platforms; hence the portability of SWC is better for such component.

When the developer wants to develop the functionality which requires optimised use of hardware resources, in that case, the developer chooses non platform data type option for implementation data type. However, it will reduce the portability of SWC to another hardware platform.

**Recommendation:** The developed components with high metric value are hardware independent. While lower value shows, the component is very hardware dependent, which also creates a problem during integration. Hence, to improve the portability the metric value should be higher.

**No.10: Measure of Dependencies**

**Quality:** Testability

**Formula:**

\[
\text{Measure of Dependencies} = \text{Input\_Dependency} \times \text{Output\_Dependency} \quad (5.10)
\]

Here,

\[
\text{Input\_Dependency} = \sum_{i=1}^{N} \text{Data\_Rec\_Event}_i + \text{Async\_Ser\_Call\_Re\_Events}_i + \text{Op\_Invoke\_Events}_i + \text{Data\_Send\_Compl\_Event}_i + \text{Exter\_Trigger\_Event}_i + \text{Data\_Rec\_Error\_Event}_i + \text{Data\_Send\_Compl\_Event}_i + \text{Data\_Read\_Point}_i + \text{Mode\_Switch\_Ack\_Event}_i + \text{Async\_Serv\_Point}_i + \text{Data\_Write\_Compl\_Event}_i + \text{Sync\_Serv\_Point}_i + \text{Mode\_Switch\_Event}_i + \text{Data\_Rec\_Point\_By\_Val}_i + \text{Mode\_Switch\_Point}_i + \text{Data\_Rec\_Point\_By\_Arg}_i + \text{Timing\_Events}_i + \text{Back\_Ground\_Event}_i \quad (5.11)
\]
5. Implementation

\[
Output\_Hdependence = \sum_{i=1}^{N} OpInvokEvents + ModeAccessPoint \\
+ DataSendPoint + DataWritePoint \\
+ SynchServPoint + AsynchServPoint
\] (5.12)

Equation 5.10 provide formula for metric Measure of Dependencies, it is given as product of input dependencies 5.11 and output dependencies 5.12 of SWC. The elements of equation 5.11 and 5.12 are described in Table 5.4,

<table>
<thead>
<tr>
<th>Metric elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataRecEvent</td>
<td>Data receive event [1]</td>
</tr>
<tr>
<td>AsynSerCallRecEvent</td>
<td>Asynchronous server call return event [1]</td>
</tr>
<tr>
<td>OpInvokEvent</td>
<td>Operation invoke event [1]</td>
</tr>
<tr>
<td>DataRecErrorEvent</td>
<td>Data receive error event [1]</td>
</tr>
<tr>
<td>DataSendComplEvent</td>
<td>Data send complete event [1]</td>
</tr>
<tr>
<td>ExterTriggerEvent</td>
<td>External trigger occurring event [1]</td>
</tr>
<tr>
<td>ModeSwitchAckEvent</td>
<td>Mode switch acknowledgement event [1]</td>
</tr>
<tr>
<td>ModeSwitchEvent</td>
<td>Mode switch event [1]</td>
</tr>
<tr>
<td>DataReadPoint</td>
<td>Data read point [1]</td>
</tr>
<tr>
<td>AsynchServPoint</td>
<td>Asynchronous server call point [1]</td>
</tr>
<tr>
<td>DataWriteComplEvent</td>
<td>Data write complete event [1]</td>
</tr>
<tr>
<td>SynchServPoint</td>
<td>Synchronous server call point [1]</td>
</tr>
<tr>
<td>DataRecPointByVal</td>
<td>Data receive point by value [1]</td>
</tr>
<tr>
<td>ModeSwitchPoint</td>
<td>Mode switch point [1]</td>
</tr>
<tr>
<td>ModeAccessPoint</td>
<td>Mode access point [1]</td>
</tr>
<tr>
<td>DataRecPointByArg</td>
<td>Data receive point by argument [1]</td>
</tr>
<tr>
<td>DataSendPoint</td>
<td>Data send point [1]</td>
</tr>
<tr>
<td>DataWritePoint</td>
<td>Data write point [1]</td>
</tr>
</tbody>
</table>

Table 5.4.: Dependency metric elements description

**Background :-** As explained in section 4.2.1, for test case generation test developer have to study the requirement specifications in terms of input and output dependency.
5. Implementation

**Design consequence:** From testability perspective, the higher the value for this metrics will suggest that higher amount of test case generation design decisions test developer has to take for full test coverage. It also reflects the complexity of test generation operation. The high value also suggest that, the SWC will take more amount of time for generation and execution of test cases.

**Recommendation:** A low value of the metric will simplify the process of test case development resulting in better test coverage leading to better software quality. The metric value can be reduced by bifurcating the SWC so that the separate SWC will have lower input-output dependency resulting lower metric values for each component.

### 5.4. Integration to continuous integration framework

To allow the developer to utilise AUTOSAR based architecture evaluation toolchain along with project development, it needs to be integrated with continuous integration framework. Figure 5.4 shows the integration of developed toolchain in pre-existing framework. In Figure 5.4, the AUTOSAR metric operations block contain the AUTOSAR quality measurement toolchain as presented in section 4.3. The AUTOSAR toolchain requires project’s specification file (ARXML) path, the version information and the system’s error log. It will return the DSM matrix, the runnable level architecture artefact information table and calculated metric results at the different levels of abstractions. Because of the modularity of the AUTOSAR quality measurement toolchain, it can be used as a standalone tool as well as it can also be incorporated into the AUTOSAR development tools or dedicated the AUTOSAR continuous integration framework. The results of the toolchain are visualise and stored by using existing visualisation module. Once the toolchain finished its operations, the continuous integration framework will notify the availability of new results to the concerned stakeholders. Once the multiple projects results are available, the concerned stakeholder can use this projects results as input to the evaluation toolchain.
5. Implementation

Figure 5.4.: Integration of AUTOSAR toolchain into continuous integration framework.

5.5. Summary

The chapter elaborates the quality measurement toolchain proposed in chapter 4. The toolchain is explained in three stages, parsing of architectural information, creation of DSM and metric derivation. The architectural information is parsed from ARXML file. The dependency matrix (DSM) is created, which is used to know the dependencies of the software components for the quality analysis. Based on GQM list, the mathematical formula for proposed metrics are presented. The brief study of architectural design elements and its effect on quality is presented as a background for derivation of each metric. Depending on this study, the quality inference of the metric value can be deduced. Finally, the recommendations for each metric is provided for architectural design changes to improve software quality. The developed AUTOSAR quality measurement toolchain is integrated into the continuous integration framework.
6. Results and analysis

In this chapter results generated out of quality measurement toolchain are presented. In first section 6.1, metrics results were visualised and its analysis is explained with the help of an example. There are in total 41 metrics were implemented, but here only some of the metric results are discussed. Having the quantitative value for quality metrics are not enough for improving the software quality, the developer or concerned stakeholder must understand what its value means. In section 6.2, analysis process of metric results is discussed.

6.1. Measurement and visualization of metrics

The quality measurement tool is deployed on a transmission control software project for its architectural quality evaluation. The project is developed by three supplier companies. The FEV Gmbh develops the powertrain application software. In total there are 72 SWCs present in 3 composition layers. The project has two variants, and work on both of these variants is in progress. Table 6.1, shows the some of the system abstraction level metrics result which will provide the insight of the project.

<table>
<thead>
<tr>
<th>System level metric</th>
<th>Measured values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measure of Inter-ECU Communication</td>
<td>570</td>
</tr>
<tr>
<td>Number of Assembly Connector</td>
<td>2163</td>
</tr>
<tr>
<td>Number of Delegation Connector</td>
<td>34</td>
</tr>
<tr>
<td>Measure of Application SWC</td>
<td>62</td>
</tr>
<tr>
<td>Measure of Service SWC</td>
<td>8</td>
</tr>
<tr>
<td>Measure of Complex-Device-Driver</td>
<td>3</td>
</tr>
<tr>
<td>Number of Modes in System</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 6.1.: System abstraction level metrics.

The measured values from above table reflect the complexity of the project. The metric such as Measure of Application SWC, which is provided by the ISO 26262 standards reflects which provide a requirement for the restricted size of software components. However, the critical analysis of this value is only possible after the statistical evaluation, because then we can conclude the ideal number (threshold) of
the component in a system for maintaining the safety criticality of the system. At the stage of quality evaluation, from the resulting quality measurement toolchain, one can realise the complex nature of the system. This result will also help stakeholders to keep track of overall project progress.

Following are the SWC level metric results. Here also only the some of the metric results are shown, and their interpretation is explained in later part of this section. The identification of threshold for given metric requires critical analysis of the statistical data of several projects, which is part of separate quality evaluation toolchain hence it is not given in these results.

![Component name](chart1.png)

![Component name](chart2.png)

Figure 6.1.: Metric result for Measure of Runnables

Figure 6.2.: Metric result for Measure of Inter Runnable Variable
6. Results and analysis

Figure 6.3.: Metric result for Measure of Average Client-Server Interactions

Figure 6.4.: Metric result for Measure of Periodic Trigger Runnable

Figure 6.5.: Visualisation of triggering behaviour of component TGI

Figure 6.6.: Metric result for Measure of Communication Dependency
6. Results and analysis

In the visualisation stage of metric toolchain 4.3, the graphs for each metric were plotted and stored. Here, plots for only 5 components per metric are shown, but in the toolchain all the software components plots or selected number of component metric results are plotted. The plots show the component metric in descending order, i.e. from most critical component to lower criticality component.
Figure 6.1 shows the Measure of runnable metric values in each SWC. To understand this metric value let us consider SWC TGI. The component used to route the FEV components signals to other supplier component signal. Hence, in terms of functionality wise it providing data transfer mechanism. According to the metric, component TGI has highest number of runnable (16). As explained in the section 4.2.1, it reflects the simplicity, analysability and time behaviour software quality. Moreover, it is recommended that to improve these qualities the metric value, i.e. some runnable should be reduced. To make this two possible approach are present either bifurcated the SWC into a small number of runnable component of fuse some of the runnable functionalities to form single runnable. The decision to choose from these options is up to the developer; he has to come up with the decision with an objective analysis of metric data. However, the graph itself will not provide developer the objective feedback, but it will trigger the thought process for the developer to come up with the decision.

After identifying the result from the metric, the concerned stakeholder will follow the other metric of the same component. For example, metric Measure of Periodic Trigger Runnable and Measure of Communication Dependency, are represented by the Figure 6.5. It shows the percentage of sporadic events and periodic events in the SWC. For TGI, this reflect that 16 out of 8 runnable behaviour is periodic and other 8 is sporadic. Hence, from analysability perspective decision can be made that SWC should be bifurcated into two component one which hold sporadic events and other with timing event. This will increase the behaviour predictability of SWC.

Even though if we bifurcate the 16 runnable component into two component with 8 runnable each, these component still have very high amount of runnables. Hence, reanalysis of component with another metric is needed. For example, metric Measure of Inter-Runnable Variable, which will provide how cohesive these runnables are can they even further bifurcated or fuse together. Another metric could be Measure of Application SWC, which will present number of application SWC present in system. It will raise the question like, can we increase the number of component by bifurcating runnable?, does bifurcation going to affect over all system analysability?

Till now we study the metric from perspective of quality analysability, similarly other quality such as simplicity or time behaviour needs to be analyse before coming to final improvement decision.

In above example of measurement metric, we can see how the metric results will trigger the design improvement and provide platform and data for critical analysis of the architecture. For each metric value such critical analysis of different metric results, DSM or Runnable Table, needs to be considered.
6.2. Analysis of metric results

The analysis of project quality is done in two parts, first by studying the individual metric values for the specific quality goal. These metric values are generated by quality measurement toolchain. For example, for AUTOSAR project we have AUTOSAR quality measurement toolchain. These measurements are independent of the other AUTOSAR projects. The quality measurement toolchain provides one or more than one metric for the individual goal. From Table 4.1 and 4.2 in section 4.2, we can notice that the specific quality goal can be given by more than one metric results. These values can be used for analysis of an individual aspect of architectural quality, but it lacks the overall picture of project’s quality. To achieve this shortcoming the second part of the quality analysis is used.

The second part of the quality analysis is done by quality evaluation toolchain, as discussed in section 3.5.2. The evaluation toolchain performs the statistical analysis over multiple projects and generates overall quality goal values. The quality evaluation toolchain aggregate the individual SWC metric values of specific quality goal with weighted values for each metric. These weighted values are based on several factors such as the influence of metric on overall quality goal, present development stage of the project, the legacy of the project, etc. Moreover, values of such aggregation from several projects are used to calculate the threshold for the specific metric which will be applicable for all projects. The thresholds are necessary to determine acceptable metric values. The overall quality values and the threshold for respective metrics will help the stakeholder to analyse the overall project quality; it also enables them to compare individual project quality with different projects. Currently, the quality measurement toolchain is used to analyse only single AUTOSAR based project, because of the current unavailability of other AUTOSAR based projects. Hence quality evaluation toolchain cannot be used for current AUTOSAR based project.

The Figure 6.11 shows the evaluation concept for the software architecture on the basis of metrics. Once the quality measurement toolchain produces the results for each project. The analysis of result in done as per following steps,

1. Results from quality measurement and their threshold from quality evaluation toolchain are get available to concerned stakeholders.

2. Stakeholders study the results and its effect on software quality.

3. Meeting of stakeholders is held and design improvements in the project is suggested.

4. Depending on viability of suggested improvement over cost of modifications and benefits of modifications design decisions implemented in the project.
6. Results and analysis

Figure 6.11.: Quality analysis process model

Different stakeholders analyse the result in different perspectives and perform actions accordingly. For example, application developer will analyse the critical component which are above threshold level for maintainability quality he will make design improvement to improve maintainability, and after improvement will monitor the quality of the component. For tool developer, after reviewing the metric results, if he realised that certain quality needs to be present in more clearer way then he can adapt new metrics to toolchain. It can also be possible that, source artefact may get changed then the tool needs to be updated for correct result. For project manager, he can plan the project progress according to the obtained metric result. The obtained result can show which component reflects high complexity, so he can allocate the resources for those components for better development.

Even though quality evaluation results are not available, the quality measurement result also can provide objective feedback regarding quality. For example, consider quality testability which is measured from metric Measure of Dependencies, it reflects the efforts and time required by the test developer for test case generation. Figure 6.10 shows the TTS is have highest metric results, which mean complexity of test case generation is going to be highest among other. Hence, even though quality evaluation results are unavailable, project management team will know for sure which of the components is showing the poor testability value, and hence they can start working on these components well in advance so that its verification and
6. Results and analysis

validation can be completed in scheduled time.

6.3. Summary

The chapter presents the quality measurement result for transmission control project. Depending on metrics derived in chapter 5, the metric results are presented in this chapter. With the help of an example of metric Number of Runnable, the objective feedback derivation process is explained. Due to lack of availability of more AUTOSAR based projects, the quality evaluation toolchain is not used to generate threshold results for further quality analysis. However, the use of quality measurement results for quality evaluation is explained. The metric results were validated using empirical validation methods with the help of domain experts.
7. Conclusion

This chapter summarises the work done during this thesis. The section 7.1 gives an overview of automotive industry challenges and contribution of this thesis. It also provides an overview of the quality metric developed. In section 7.2, we discuss the limitations of the quality measurement toolchain. In section 7.3, the future outlook of this topic is discussed.

Currently, the automotive industry is facing challenges related to increasing complexity, shorter time-to-market and increase in the cost of development; the demand from the automotive industry for better quality software is also increasing. The software architecture has very significant impact on software quality, as an initial design decision for shaping the software is taken at the architectural level. Considering the distributed development environment, there is a need for standardisation of automotive architecture for better transferability of specifications. Several attempts are going on, such as ADL or AAF for development of the standardised architecture specification. The AUTOSAR also provides an implementation view of software component architecture. The standardisation of the architectural element will provide a better specification of software architecture, which can be useful in an early software quality determination. Although AUTOSAR was developed to address the growing challenges of software quality such as complexity, scalability or reusability, it still lacks the quality monitoring specification for non-functional qualities, such as maintainability or analysability. The automotive domain requires a lightweight tool for the software quality monitoring, which can provide an objective feedback regarding software quality. The tool developed as part of this thesis is one step forward towards achieving the goal.

7.1. Contribution

To determine the architecture quality the Goal-Question-Metric (GQM) approach was used. The quality goals were derived from the standards such as ISO 25010, ISO 26262 and AUTOSAR modelling guidelines. In total 26 quality goals were identified, from which objective feedback can be provided at the architecture level using the metric based technique. After careful study of software architecture specification using ARXML file, the metric is derived. In total 32 component level metrics and 9 system level metrics were proposed. These metrics were developed as part of the tool which can work in the continuous integration framework. The metric correctness, reproducibility, and sensitivity were verified by providing control data and
7. Conclusion

Evaluating results. The metric tool was deployed on a transmission control software project, and the results were obtained. The results were verified and validated by industry experts.

The developed quality measurement toolchain along with quality evaluation toolchain will provide the objective feedback regarding architectural quality. This will help stakeholders to monitor the non-functional quality characteristics of the automotive software.

The recommendations given for any particular metric is not a hard and fast remedy for that particular quality, the stakeholder has to take other metrics into consideration, because there is a possibility present, that the recommendation to improve one architecture quality may affect the other quality. For example, the recommendation of fusion of two runnables because of the high interconnection among them is given as a part of metric ”Number of Runnable”. This recommendation also has to see from the other quality perspective, such as analysability. The analysability of SWC may get reduced if the fusion of two runnables results into SWC with one runnable. Hence, stakeholder needs to consider the metric which are interdependent and trade-off analysis of their quality should be carried out for taking quality improvement decision.

The quality evaluation tool is not intended to replace the traditional methods for verification and validation or even architecture evaluation technique based on scenarios based methods. Because traditional methods provide comprehensive and stakeholder requirements specific analysis results. However, these methods require lots of project resources in terms of time and effort. Hence, they cannot be readily and repetitively be used in the continuous evolving environment. The tool developed can fill this gap by monitoring the quality of software architecture, and providing the objective feedback over the quality in short and quick loops.

7.2. Limitation

Although with standardisation of AUTOSAR, the architectural specification information becomes much clearer, however, it does not provide information regarding the functional behaviour of SWC. Hence, the quality result obtained from toolchain will not be able to reflect the functional quality i.e. the internal software functionality quality.

The quality measurement and evaluation toolchain provide quantitative values regarding architecture quality, however, to get objective feedback from the results the stakeholders must be able to understand the quality definition. And they must able to answer the quality related questions such as, how the metric value reflects the software quality?, what measure should be taken to improve the quality? And what
will be the impact of this quality improvement measure on other quality? Hence, these questions regarding the tool, requires the stakeholders either rely on tool developer or it expects the stakeholder to understand the quality definitions and its possible design improvements from quality results, which brings the rigidity to the process.

The quality evaluation tool performs statistical analysis on multiple projects to generate thresholds for the metric result and allows the developer to compare quality values among these projects. However, the thresholds for the metric results generated within local group will not be able to compare their quality results with projects outside the local group. The standardised open source software architecture project is require to evaluate the local project architecture with standardised project, which will be available to all stakeholders from different companies. This will allow stakeholders to make local project quality evaluation analysis considering standardised open source projects.

The tool developed for the AUTOSAR standardised architecture will only work for AUTOSAR based software projects. For other standardised architecture, separate measurement tool will be required to implement.

7.3. Future outlook

The analysis of goodness of the result is provided by threshold values of the metric results. These values are obtained from a statistical analysis of metric data of multiple projects. Due to lack of availability of other projects at the present situation, this analysis remains open for future investigation. However, with the empirical analysis of quality measurement results, experts have suggested the threshold values for the metrics.

The metrics are derived using quality standards, architecture guidelines and expert opinion in the domain. The expert group needs to be enlarged, and contribution from different levels of industry experts and stakeholders need to be involved, such as OEM’s, suppliers or other domain experts. By doing this, quality concerns related to all these sectors can be addressed by toolchain.

Currently, the developer or stakeholder has to derive the objective feedback from the obtained results. With good documentation and visualisation, the process can be automated, and inference of the results can be automatically provided to the stakeholder. This will increase the ease-of-handle for the tool.

As per the classical verification and validation methods, the metric values should be validated through control experiments or correlation of results. However, the automotive software architecture is very complex and it need not be necessary that the
results reflected for small control experiment will be validated to all the projects. Hence, to optimise the quality measurement and the quality evaluation tool, the quality analysis of a number of projects needs to be conducted and tool should be continuously reviewed over the various project results.

The quality measurement tool is designed by considering powertrain domain of automotive software. The domain has its modelling guidelines; hence to facilitate other domains architecture quality evaluation, metric specific to that particular domain or analysis of results as per that particular domain specification needs to be introduced.

The AUTOSAR platform is regularly updated, the new specifications and functionality are introduced every year to the standard. For example, providing solution for multicore processor in automotive application [41] or integration of a more dynamic system such as AUTOSAR adaptive platform initiative [16]. The metric needs to be proposed for such new technologies and the toolchain must be adapted for new architectural elements.
Bibliography


[4] Iso/iec 25010:2011 systems and software engineering systems and software quality requirements and evaluation (square) system and software quality models


BIBLIOGRAPHY


96
[38] ETAS GmbH: Rta-rte v5.6.0 user guide (2013), https://www.etas.com


[86] TADL2: Tool support for the analysis of tdl2 timing constraints using timesquare, http://hal.inria.fr/docs/00/85/06/73/PDF/paper.pdf


A. Proposed GQM list

The chapter contains the complete list of proposed GQM list. Firstly, the proposed
metrics at software component level are mentioned. The Table A.1 shows the GQM
list of developed metric for the software component level architectural elements.
Then the metric proposed at system or composition level are mentioned. The table
A.2 shows the GQM list of developed metric for the system level architectural ele-
ments.

In the table metric quality goal, its definition question and proposed metric is pre-
sented. The metric definition question should be use for making design decision
for the quality improvement based on metric data, the section 5.3 illustrates this
process with examples.

<table>
<thead>
<tr>
<th>No.</th>
<th>Goal</th>
<th>Question</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Simplicity, Analysability, Time behaviour</td>
<td>How many runnables are present in SWC?</td>
<td>Number of Runnable per SWC</td>
</tr>
<tr>
<td>2.</td>
<td>Simplicity, Modularity (cohesion), Time behaviour</td>
<td>How many inter runnable variables are present in SWC?</td>
<td>Measure of Inter-Runnable Variable</td>
</tr>
<tr>
<td>3.</td>
<td>Simplicity, Maintainability, Time behaviour</td>
<td>What is an average number of operation usage per runnable in SWC?</td>
<td>Measure of Average Client-Server Interactions</td>
</tr>
<tr>
<td>4.</td>
<td>Time behaviour, Maintainability</td>
<td>How many Exclusive Areas are present in SWC?</td>
<td>Measure of Exclusive Area</td>
</tr>
<tr>
<td>5.</td>
<td>Maturity, Analysability, Modularity</td>
<td>What is a number of runnable entities which are periodically triggered in SWC?</td>
<td>Measure of Periodic Trigger Runnable</td>
</tr>
<tr>
<td>6.</td>
<td>Simplicity, Analysability, Time behaviour</td>
<td>What is the ratio of a number of the runnable entity which is periodically triggered to the total number of runnable in SWC?</td>
<td>Measure of Predictability</td>
</tr>
</tbody>
</table>
### A. Proposed GQM list

<table>
<thead>
<tr>
<th>No.</th>
<th>Domain</th>
<th>Question</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td>Maturity, Simplicity, Time behaviour, Modularity</td>
<td>How much runnable entities are triggered by communication event in SWC?</td>
<td>Measure of Communication Dependency</td>
</tr>
<tr>
<td>8.</td>
<td>Testability, Analysability</td>
<td>How many runnable entities are directly triggered by another entity in SWC?</td>
<td>Measure of Functional Behaviour Dependency</td>
</tr>
<tr>
<td>9.</td>
<td>Testability, Simplicity, Capacity</td>
<td>What is an average number of arguments per operation used in SWC?</td>
<td>Measure of Average Arguments</td>
</tr>
<tr>
<td>10.</td>
<td>Modifiability, Functional completeness, Interoperability</td>
<td>How many unconnected input ports are present in SWC?</td>
<td>Percentage of Unconnected Port</td>
</tr>
<tr>
<td>11.</td>
<td>Modifiability, Functional completeness</td>
<td>How many unconnected ports are present in SWC?</td>
<td>Percentage of Unconnected Port</td>
</tr>
<tr>
<td>12.</td>
<td>Co-Existence, Resource Utilization, Maturity, Time behaviour</td>
<td>How many synchronous client-server communications are present in SWC?</td>
<td>Measure of Synchronous Communication</td>
</tr>
<tr>
<td>13.</td>
<td>Co-Existence, Resource Utilization, Maturity, Time behaviour</td>
<td>How many asynchronous client-server communications are present in SWC?</td>
<td>Measure of Asynchronous Communication</td>
</tr>
<tr>
<td>14.</td>
<td>Reliability</td>
<td>How many explicit communication links are present in SWC?</td>
<td>Measure of Explicit Communication</td>
</tr>
<tr>
<td>15.</td>
<td>Reliability, Simplicity</td>
<td>How many implicit communication links are present in SWC?</td>
<td>Measure of Implicit Communication</td>
</tr>
<tr>
<td>16.</td>
<td>Analysability, Compatibility</td>
<td>How many numbers of ports use the same interface in SWC?</td>
<td>Interface Re-usability Index [12]</td>
</tr>
</tbody>
</table>
A. Proposed GQM list

<table>
<thead>
<tr>
<th>No.</th>
<th>Category</th>
<th>Question</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Resource utilisation</td>
<td>How many number of data elements (Sender-receiver, parameter, Nv-Data) use same application data type in SWC?</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Resource utilisation</td>
<td>How many application data types use same implementation data type in SWC?</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Resource utilisation</td>
<td>How many implantation data types use same base data type in SWC?</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Maintainability</td>
<td>How many data element are presents per port interface in SWC?</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Simplicity</td>
<td>How many record data types use per port interface in SWC?</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Modularity, Modifiability</td>
<td>How many number of input side couplings are present in SWC?</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Modularity</td>
<td>How many number of output side couplings are present in SWC?</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Analysability</td>
<td>How many components are connected to SWC?</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Testability, Understandability</td>
<td>How many Modes are present in SWC?</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Portability</td>
<td>What is the ratio of implantation reference of platform type to total implementation reference in SWC?</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Modularity</td>
<td>What is the percentage of internal interaction in SWC?</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Testability</td>
<td>What is input and output dependency of SWC?</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Simplicity</td>
<td>What is the degree of total interaction links within SWC?</td>
<td></td>
</tr>
</tbody>
</table>
### A. Proposed GQM list

<table>
<thead>
<tr>
<th>30.</th>
<th>Maintainability</th>
<th>What is functional size of SWC?</th>
<th>Measure of Functional Size</th>
<th>84</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.</td>
<td>Analysability</td>
<td>How much functional size is distributed within SWC?</td>
<td>Measure of Function Size Distribution</td>
<td>21</td>
</tr>
<tr>
<td>32.</td>
<td>Time behaviour</td>
<td>What is the ratio of a number of the task assigned (to SWC) to the total number of runnables in SWC?</td>
<td>Ratio of Task Assigned to Total Number of Runnables</td>
<td></td>
</tr>
</tbody>
</table>

Table A.1.: Developed metric at system software component level for different quality goal
### A. Proposed GQM list

The Table A.2 contains the system level GQM list.

<table>
<thead>
<tr>
<th>No.</th>
<th>Goal</th>
<th>Question</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Time behaviour</td>
<td>What is an average number of runnables per task in ECU?</td>
<td>Ratio of Runnable to Task</td>
</tr>
<tr>
<td>2.</td>
<td>Portability</td>
<td>What is the percentage of sensor-actuator components in composition?</td>
<td>Sensor-Actuator Portability</td>
</tr>
<tr>
<td>3.</td>
<td>Time behaviour, Simplicity, Resource Utilisation</td>
<td>What is the number of inter-ECU communication channel?</td>
<td>Measure of Inter-ECU Communication</td>
</tr>
<tr>
<td>4.</td>
<td>Simplicity</td>
<td>How many SWCs present in system?</td>
<td>Measure of Software Component</td>
</tr>
<tr>
<td>5.</td>
<td>Simplicity</td>
<td>How many application SWCs are present in the system?</td>
<td>Measure of Application Software Component</td>
</tr>
<tr>
<td>6.</td>
<td>Portability</td>
<td>How many complex-device-driver SWCs are present in the system?</td>
<td>Measure of Complex-Device-Driver</td>
</tr>
<tr>
<td>7.</td>
<td>Simplicity, Analysability</td>
<td>How many modes are present in the system?</td>
<td>Number of Modes in System</td>
</tr>
<tr>
<td>8.</td>
<td>Simplicity</td>
<td>How many SWC to SWC connections are present in the system?</td>
<td>Number of Assembly Connector</td>
</tr>
<tr>
<td>9.</td>
<td>Simplicity</td>
<td>How many SWC to composition connections are present in the system?</td>
<td>Number of Delegation Connector</td>
</tr>
</tbody>
</table>

Table A.2.: Developed metric at system abstraction level for different quality goal