Investigation of Resources Types for OSLC domains Targeting ISO 26262

Focus on Knowledge Representation of the Right side of the ISO 26262 Software V-model

Kathyayani Padira

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden
This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:
Author(s):
Kathyayani Padira
E-mail: kapa15@student.bth.se

Industrial advisor:
Barbara Gallina
Senior Lecturer at Mälardalen University
Visiting Researcher
GenReUsableSafety
Scania CV AB, Södertälje, Sweden

Mattias Nyberg
Project Manager
ESPRESSO
Scania CV AB, Södertälje, Sweden

University advisor:
Dr. Krzysztof Wnuk
University Lecturer
Department of Software Engineering

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Abstract

Context ISO 26262 requires compilation of traceable work products across the application lifecycle as product based safety evidence. The compilation of such safety evidence is a time consuming and arduous task. Open Services Lifecycle Collaboration (OSLC) is an initiative that supports traceability through tool interoperability. The meta modelling of the ISO 26262 work products in the structure of Resource Description Framework (RDF) can be used for achieving interoperability. Thus, OSLC services used on the RDF exchanged between interoperating tools aids in an effective way of compiling the product based safety evidence for ISO 26262 safety case.

Objectives Representing the compilation of traceable work product types for the software testing and verification in ISO 26262, in form of a RDF based conceptual meta model. Testing and extending the concepts by instantiating the meta model with work products to be represented in RDF for a case of a truck Electronic Control Unit (ECU) system. Lastly, validating the effectiveness of the conceptual meta model for its compliance to ISO 26262.

Methods To realise the objectives, a case study was conducted at Scania CV AB, Södertälje, Sweden, a manufacturer of safety critical ECU systems used in heavy automobiles. The case study was conducted in three consecutive cycles. The first cycle of qualitative inductive content analysis of the ISO 26262 standard and its related document at the company for defining the conceptual meta model. The second cycle of qualitative deductive content analysis for testing, extending and refining the conceptual meta model. The last cycle of validating the effectiveness of the tested and extended conceptual meta model for compliance to ISO 26262.

Results The main result was the tested, extended and refined RDF based ISO 26262 conceptual meta model depicting traceable work product types for software testing and verification of a safety critical ECU system. The testing and extending of the conceptual meta model was performed with respect to the Main1 (M1) ECU system at Scania. The RDF was defined for the work products of M1 ECU system. Finally, the conceptual meta model was validated for its ef-
fectiveness in realising the criteria of abstraction, confirmability and traceability based on ISO 26262.

**Conclusions** Thus, the RDF based conceptual meta model depicting product based safety evidence provides a structure for realising the traceability required for compiling the software testing and verification part of ISO 26262 safety case. The meta model was tested by defining the RDF for the work products of a truck ECU system that would be exchanged for achieving interoperability. Finally, the conceptual meta model was validated for representing the knowledge required for showing traceable product based safety evidence for ISO 26262 safety case.

**Keywords:** ISO 26262, Safety evidence, Software testing, Traceability, Open Services Lifecycle Collaboration, Interoperability, Knowledge representation.
Acknowledgments

I had a truly enriching and fulfilling experience in performing my master thesis. It was an opportunity to learn and improve my knowledge in the current topics of computing used in the automotive sector. I am grateful for having the opportunity to perform an industrial master thesis at Scania CV, AB, Södertälje, Sweden. It gave me the opportunity for interacting and learning from some of the most innovative and creative minds in the industry.

I would like to take this opportunity to express my sincere gratitude to my supervisor at university, Dr. Krzysztof Wnuk and to my industry supervisor, Barbara Gallina. Their constant guidance, support and advice have motivated and encouraged me to perform my best. I thank Krzysztof Wnuk for always being supportive and positive of my work, which has motivated me and helped to improve my confidence. I thank Barbara, for advising and constantly encouraging me to give my best. Her guidance helped me to move in the right direction at the company. This has been a journey full of ups and downs with a fair amount of success and few setbacks. In the end, I have grown both as a person and as student in learning to have perseverance and self belief. I am thankful to Mattias Nyberg, Project Leader, ESPRESSO, for giving the opportunity for contributing to the project. Thanks to Julieth, co-master thesis student and all my colleagues at Scania who have been highly supportive and helpful.

Finally, and most importantly I would like to express my heartfelt gratitude to my father, Mr.P. Madhusudan Reddy, my mother, Mrs.P. Lavanya Reddy and my brother, P. Mithileshwar Reddy, for their everlasting love, support, belief and well wishes. Lastly, I would like to thank all others who had directly or indirectly been a part of this journey.

I would like to acknowledge and mention the two projects within which this master thesis dissertation was conceived, ESPRESSO [1] and Gen&ReUsableSafety [2].
# Contents

Abstract .......................................................... ii

Acknowledgments ................................................. iv

1 Introduction ...................................................... 1
   1.1 Overview .................................................... 1
   1.2 Problem Description ....................................... 3
   1.3 Research Aim and Objectives ............................... 4
   1.4 Research Questions and Instrument ....................... 4
   1.5 Expected Research Outcomes ............................... 6

2 Background and Related Work ................................. 7
   2.1 Background .................................................. 7
      2.1.1 ISO 26262 ............................................... 7
         2.1.1.1 ISO 26262 Part-6: Right Side of the V-model ...... 8
         2.1.1.2 Safety Case: Traceable Safety Evidence ............ 9
      2.1.2 Traceability and Interoperability ...................... 11
      2.1.3 Open Services Life cycle Collaboration (OSLC) .......... 11
         2.1.3.1 OSLC Domain Specifications ....................... 11
         2.1.3.2 OSLC Resources: Resource Description Framework 12
      2.1.4 Conceptual Meta Model ................................ 13
   2.2 Related Work ............................................... 15

3 Research Method .................................................. 18
   3.1 Research Design ............................................ 18
      3.1.1 Case Description .................................. 20
   3.2 First Cycle: Data Collection and Analysis ............... 20
      3.2.1 First Data Collection Cycle ........................... 20
      3.2.2 First Data Analysis Cycle ........................... 21
         3.2.2.1 Phase 1: Preparation ............................. 21
         3.2.2.2 Phase 2: Organising ............................. 22
         3.2.2.3 Phase 3: Reporting of Results .................... 24
   3.3 Second Cycle: Data Collection and Analysis ............. 24
      3.3.1 Case Description .................................. 25
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>ISO 26262 Software V-model as specified in [3]</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Mind Map for Literature Review</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Overview of ISO 26262 as specified in [3]</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Safety Evidence in a Safety Case [4][5]</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>ASIL Inheritance from Requirements to Testing based on Part 2,4 and 6 in [3]</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td>OSLC core services and specifications based on [6]</td>
<td>12</td>
</tr>
<tr>
<td>2.6</td>
<td>RDF Triple based on [7]</td>
<td>13</td>
</tr>
<tr>
<td>2.7</td>
<td>UML Class Diagram Concepts used in Conceptual Meta Model</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Research Design</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Embedded Case Study based on [8]</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Qualitative Content Analysis: Inductive Content Analysis</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>M1 used in Fuel Level Display System (FLDS)</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>M1 ECU System</td>
<td>26</td>
</tr>
<tr>
<td>3.6</td>
<td>Qualitative Content Analysis: Deductive Content Analysis</td>
<td>29</td>
</tr>
<tr>
<td>3.7</td>
<td>Structured Analysis Matrix: Categories and codes</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Conceptual Meta Model</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>ISO 26262 specific Enumerations</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>WP available for M1 ECU System: Results of Closed Ended Questions</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>SVP Class Instantiation for M1 ECU system: Software unit test level</td>
<td>52</td>
</tr>
<tr>
<td>4.5</td>
<td>Corresponding RDF graph for SVP Work Product</td>
<td>53</td>
</tr>
<tr>
<td>4.6</td>
<td>Refined and Tested Conceptual Meta Model</td>
<td>54</td>
</tr>
<tr>
<td>5.1</td>
<td>Categorization for Main Category- Software Verification Plan</td>
<td>64</td>
</tr>
<tr>
<td>5.2</td>
<td>Categorization for Main Category- Software Verification Specification</td>
<td>65</td>
</tr>
<tr>
<td>5.3</td>
<td>Categorization for Main Category- Software Verification Report</td>
<td>65</td>
</tr>
<tr>
<td>8.1</td>
<td>Questionnaire for Validation of the Tested and Extended Conceptual Meta Model</td>
<td>101</td>
</tr>
<tr>
<td>8.2</td>
<td>Histogram with Results of Validation</td>
<td>102</td>
</tr>
<tr>
<td>8.3</td>
<td>Express Scribe- Example Screen Shot 1</td>
<td>103</td>
</tr>
</tbody>
</table>
# List of Tables

1.1 Mapping of Terms ......................................................... 5
3.1 Mapping of ISO 26262 and Scania Terms ............................... 27
4.1 Open coding: Extracted Open Codes .................................. 35
4.2 Relationships in the Conceptual Meta Model ......................... 44
4.3 General Information of the Interviews ................................. 46
4.4 Data coding according Structured Analysis Matrix- Work Products and their Attributes ................................. 49
4.5 Interviewee Profile for Validation .................................... 56
5.1 Open Codes- Software Verification Plan ............................... 62
5.2 Open Connector Codes- Software Verification Plan ................ 63
A list of few of the terminologies used in the report are presented in an alphabetical order below

*Allocation Element (AE)* is an abstract concept of software functionality, used at Scania. Allocation elements are used to realise the user functions. They are independent entities that are possible to allocate to any Scania ECU [9].

*Allocation Element Requirements (AER)* are the implementation specific requirements allocated to the Allocation elements realised by the ECUs [9].

*C1 Calibration parameters* are Scania specific diagnostic parameter setting software used for communication with ECU systems. These parameters are pre defined and used for automatic calibration while executing the ECU systems.

*Configuration Management SOP* stands for the Start Of Production (SOP). SOP are used for managing and documenting the configurations for verification.

*Controller Area Network (CAN) bus* is a physical signal designed to allow microcontrollers in ECUs and devices to communicate without a host computer in a vehicle.

*Electronic Control Unit (ECU) system* is a system that consists of embedded software and hardware. An ECU system consists of sensors, switches and actuators.

*Element* is a system or part of a system including components, software, and software units [3].

*Embedded Software* is used to realise a specific functions of the ECU system often related with real-time computing constraints.

*Functional Allocation Description* is a document that describes the allocation of AEs to an ECU and its interfaces to physical signals. It is in form of a graphical diagram [9].
**Functional Safety Requirements** are specifications for implementation independent safety behaviour, or implementation-independent safety measure, including its safety-related attributes [3].

**Hazard Analysis** is a way of risk assessment to identify and categorize hazardous events of items and to specify safety goals and ASILs related to the prevention or mitigation of the associated hazards in order to avoid unreasonable risk [3].

**ISO 26262- Work Product Type** is the deliverable or work artefact as result of realising one or more associated requirements in ISO 26262 [3].

**Item** is a system or group of systems to implement a user function, for which ISO 26262 specifications are applied to [3].

**Jira Tool** is a project management tool used for Agile development Methodology.

**Middleware Software** is the basic and common software used in all ECUs at the company. It consists of software that handles the RTDB variables, files and other signals.

**Model** is a representation in a certain medium, of something in the same or a different medium. It captures the important aspects of the what is being modelled from the certain point of view and omits the rest [10]. In our case from the point of view of representing product based requirements in the ISO 26262 Standard as traceable work product types and omitting the parts of the standard pertaining to the process based requirements.

**Process based safety evidence** are the part of the deliverables that depict the way in which the work deliverable showing safety can be achieved from following the requirements and recommendations in the ISO 26262 standard, to show evidence of safety [3].

**Product based safety evidence** are the work artifacts or deliverables that depicts what must be contained and possessed from following the requirements and recommendations in the ISO 26262 standard, to show evidence of safety [3].

**Real Time Database (RTDB)** is a variable or signal that carry the real time data values stored in the database while executing the ECU system.

**Simulink Model** is a way of depicting the model based design and implementation. It is a way of modelling, used in model based development[11].

**Software Component** is the complete integration of the software units or a part of the integration of the software units. Thus, it could be the embedded software or a model.
Software Safety Requirements are derived for implementation of software related specifications of the Functional Safety Requirements.

Software Unit is an atomic, smallest and independent part of the software subject to standalone testing. For model based development, the unit can be a model.

Software Unit Design specification is either a document or a model (i.e. in case of model based development) that describes the software unit [3].

System is a set of elements that relates at least a sensor, a controller and an actuator with one another [3]. It is essentially an Electronic Control Unit (ECU).

Technical Safety Requirements are derived for implementation of associated Functional Safety Requirements [3].

User Function describe the needs of the user in terms of the usage he requires from the vehicle. It is a requirements assigned at the vehicle level and is stated independent of the implementation [3].

Variant at Scania are used for product line development. The ECU systems developed are used in different models of trucks and buses. These models have variations with respect to fuel type, engine types, engine capacity etc. These are the variants referred to in the report.

Verification is a more comprehensive quality management activity that includes testing as a part of it [3].
Chapter 1

Introduction

1.1 Overview

The ISO 26262 standard was introduced to comply with the functional safety needs specific to the electrical / electronic systems (E/E systems) used for road vehicles up to 3.5-ton gross mass [3]. It was released in 2011, and is relatively new standard for the automotive industry. The ISO 26262 standard is applicable to all activities during the safety lifecycle of vehicles composed of Electronic Control Unit (ECU) systems. These ECU systems have embedded software elements that realise safety-related functionalities [12].

The process for developing safety-related software in compliance with ISO 26262 is based on the V-model of software development (see Figure 1.1). The left-hand side Work Products (WPs) of the V-model represent artifacts like requirements specification and design, the so called immediate evidence. The right-hand side of the V-model represents software testing and verification related Work Product (WP) like software verification specifications, software verification report etc. that represent direct evidence [13]. Thus, the right side of the V-model pertains to software testing activities. The current thesis is scoped down to focus on the right-hand side of the standard. Refer to Figure 1.1 for the V-model. For each clause in the V-model, the requirements for the clause activities and the WPs produced as a result are defined in the standard [14]. The role of the software testing part of the functional safety standard, ISO 26262, is to ensure that the embedded software in the ECU systems behave as expected, even in the face of unplanned or unexpected circumstances [15].

Scania is a major Swedish automotive manufacturer of heavy trucks and buses. Scania develops, manufactures and sells trucks with a gross vehicle weight of more than 16 tones. Currently, the standard only addresses functional safety of road vehicles with a maximum gross weight of 3.5 tons. Heavy trucks are not yet contemplated by the standard. A new version of the standard is expected to be issued by 2018 [12]. Thus, certification to the standard would become mandatory for heavy vehicle manufacturers by 2018. This drives the necessity for heavy vehicle manufacturers to constantly monitor the ongoing revisions of the standard and start adapting them. This is needed to be able to continue selling
their products when the standard is enforced. The certification for ISO 26262 requires the generation of safety case. Currently there is a need of efficient ways for building the safety case, which shows evidence of process and product based functional safety required for ISO 26262 certification [13].

![ISO 26262 Software V-model as specified in [3]](image)

Figure 1.1: ISO 26262 Software V-model as specified in [3]

Open Services for Lifecycle Collaboration (OSLC) [16] is a set of specifications for interoperability of software development lifecycle tools. Interoperability is the ability of two or more systems or tools to exchange data and use the data that has been exchanged [17]. OSLC is a community of software developers, researchers and professionals aiming to standardize the way in which software tools interoperate with each other across non-homogenous platforms. This data, which is exchanged between tools is called a resource in OSLC. The interoperability of resources can be traced and shown across the Application Lifecycle Management (ALM) phases using OSLC domain specifications. ALM consist of various phases required during governance, development and maintenance of an application [18]. The OSLC domain specifications are defined for each phase of the ALM and specify of the types of resources produced in the respective phase of ALM. OSLC domain specifications consist of the descriptions of the structure for resources called the resource type. The resource types is a static structure to define resources in a common machine readable format for tools to interoperate with each other. The common machine readable format used for the resources in OSLC is Resource Description Framework (RDF). Currently, OSLC domain specifications already have a few types of resources specified.
1.2 Problem Description

A safety case is “a structured argument, supported by a body of evidence that provides a convincing and valid case that a system (see nomenclature) is safe for a given application, in a given operating environment” [3][19]. The safety case has certain requirements of traceability that are required to compile a valid case for safety. The compilation and construction of safety case is one of the most time consuming, costly and challenging task [20]. But, the construction, review and acceptance of safety case is a critical step in the safety assurance process of safety critical systems and is needed for ISO 26262 certification. The ISO 26262 standard has both process based and product based requirements. The safety case for ISO 26262 is constructed for showing safety evidence of both WP (i.e product based requirements) and the processes used to define these WPs (i.e process based requirements), across the ALM. The focus in the thesis is on result of realising the product based requirements for the software testing and verification in ISO 26262. The result are the traceable WPs showing safety evidence as specified on the right hand side of the ISO 26262 V-model.

The safety case, thus, requires traceability links of software testing related WPs for the right hand side of the ISO 26262 V-model. Therefore, interoperability that provides for efficient realisation of traceability between the work products, would aid in a cost and time effective way of building the safety case. ISO 26262, released in 2011, is a relatively new standard in the automotive industry. There have been investigations on the ways of building a safety case for ISO 26262 by using Goal Structure Notation (GSN) [21] [22] and safety contracts [23]. Yet there is little literature pertaining to ways of effectively and less time consumingly building the safety case [11] [24]. Most such literature was related to the projects of Advanced Research and Technology for Embedded Intelligent systems (ARTEMIS), Cost Efficient methods and processes for safety relevant embedded systems (CESAR), Model based Analysis and Testing (MBAT), Industrial Framework for Embedded Systems Tools (iFEST) cited in [17], [25], [16], [26] and [27]. These projects are related to achieving safety compliance for safety critical embedded systems in the automotive industry. They have proposed and used OSLC to achieve interoperability for efficiently generating the safety case. As the WPs generated by heterogeneous tools co-evolve, the changes in one WP can be traced to changes in another WP and these changes can be efficiently reflected in a safety case [19]. Thus, achieving interoperability is expected to guarantee the availability, search ability and maintainability of the resources as well as enhanced precision and structure as required by ISO 26262, part 10 [13]. Therefore, the WP types and their traces for the right hand side of the ISO 26262 V-model must be identified to define them as ISO 26262 specific resource types for OSLC. Such resource types are the interoperated WPs, that can effectively show product based safety evidence for software testing and verification. The thesis aims at a proposal for an effective way of getting certified by ISO 26262 standard. Hence,
the research gap is the need for the identification of ISO 26262 based traceable WP types (i.e safety evidence for ISO 26262 safety case) and their representation as proposed extensions to OSLC domain resource types. This representation depicts the structure for the traceable WPs as required by RDF. The RDF for ISO 26262 WPs can be traced, exchanged, updated, maintained, created and deleted between interoperating tools. Thus, the the proposed solution is in form that can be used to achieve interoperability for showing compliance to the ISO 26262 in an effective, fast and easy way.

1.3 Research Aim and Objectives

The research aim was to define, test the usability, extend and verify the effectiveness of a proposal that depicts traceable product based safety evidence represented as OSLC domain resource types. The relevant research objectives for the aim were:

O1: Analysing the software testing and verification clauses of the ISO 26262 standard and company specific documents on ISO 26262, for defining the WP types and their attributes that show product based safety evidence.

O2: Identifying traceability requirements from ISO 26262 and company specific documents on ISO 26262, for defining the traceability links between the WP types.

O3: Creating a conceptual meta model of the WP types, their attributes and traceability links identified from the previous two objectives. The meta model has been conceptualised to propose resource types for extending OSLC domains.

O4: Identifying the data for the WP types and their attributes in the conceptual meta model for software testing and verification of a truck ECU system. Defining the WPs as OSLC resources, in RDF, so as to be used by OSLC for achieving interoperability.

O5: Verifying the effectiveness of conceptual meta model in defining ISO 26262 specific resource types for extending OSLC domains.

1.4 Research Questions and Instrument

RQ1: What are the work product types and their attributes specific to software testing and verification specifications of ISO 26262 for extending the OSLC domain resource types?
RQ2: What is the traceability required between the work product types specific to software testing and verification specifications of ISO 26262 for extending the OSLC domain resource types?

RQ3: What are the work products and their attribute values specific to software testing and verification of a truck ECU system software for extending the OSLC resources?

RQ4: What is the effectiveness of the proposed conceptual meta model for extending OSLC domain resource types in showing compliance to ISO 26262?

Motivation
The motivation for the elicited research questions and the instrument used to address them is explained below.

RQ1 and RQ2 were answered by performing qualitative inductive content analysis of the ISO 26262 standard documents and company specific documents on ISO 26262. These documents were analysed for defining the software testing related WP types, their attributes and their traces. These WP types, their attributes and traces were represented in a conceptual meta model as ISO 26262 resource types for extending OSLC domains. Here, the conceptual meta model is the generated hypothesis.

RQ3 was answered by first, identifying the WPs involved during the software testing and verification of a truck ECU system software. Then, the conceptual meta model was tested for its usability, by instantiating the attribute values for the WPs. These WPs are the OSLC resources that have been defined in RDF. Further, while testing the usability of the meta model, additional software testing and verification based WP types or attributes, that can be used for showing product based evidence of safety have been identified and defined as an extensions or refinements to the concepts in the meta model. Hence, the hypothesis is confirmed by testing its usability and extending it if required. [8].

RQ4 was answered by a statistical analysis of the responses for closed ended questionnaire targeting various criteria that validate the effectiveness of the conceptual meta model. Subjects with varying levels of knowledge in ISO 26262 and software testing and verification at the company were used as the source of data collection.

Table 1.1 outlines the mapping of the terms in ISO 26262, OSLC and conceptual meta model defined based on the paper [28]. These mapped terms are used interchangeably in the report.
Chapter 1. Introduction

Table 1.1: Mapping of Terms

<table>
<thead>
<tr>
<th>ISO 26262</th>
<th>OSLC</th>
<th>Conceptual meta model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Product (WP) Type</td>
<td>Resource Type</td>
<td>Class</td>
</tr>
<tr>
<td>WP</td>
<td>Resource</td>
<td>Object of a Class</td>
</tr>
<tr>
<td>Attributes</td>
<td>Properties</td>
<td>Class attributes</td>
</tr>
<tr>
<td>Traceability links between the various Work products (WPs)</td>
<td>Relationship properties</td>
<td>UML Dependency Relationship Type</td>
</tr>
</tbody>
</table>

1.5 Expected Research Outcomes

The expected outcomes of the thesis dissertation are:

- Conceptual meta model defining the traceable WP types and their attributes, as a proposed extension to OSLC domain resource types.

- Testing the usage of the conceptual meta model by the instantiating with WPs and their attribute values for software testing and verification of Main1 (M1) ECU system. These are the OSLC resources used for achieving interoperability. Further extending of the concepts in the meta model in the case of identification of new WP types or attributes showing product based safety evidence.

- The validation of the effectiveness of the proposed conceptual meta model for defining an ISO 26262 specific extension for OSLC domain resource types.
Chapter 2

Background and Related Work

2.1 Background

A literature review was performed according to guidelines in [29]. All research needs to be informed by existing knowledge in a subject area. The purpose of the literature review was to use the literature required for understanding the background of the subject of study. The literature review was conducted by accessing the scientific database, Google scholar. Google scholar was chosen to avoid any specific publisher bias [30]. Different keywords were formulated based on guidelines provided in [29]. This resulted in the retrieval of various sources of literature. The sources used were peer reviewed papers, published dissertations, online citations and published books. An initial mind map with the key concepts related to the subject of study is elicited as presented in Figure 2.1. This mind map was used as the basis for conducting the literature review for describing the key concepts presented in the Section 2.1.

Figure 2.1: Mind Map for Literature Review

2.1.1 ISO 26262

Chapter 2. Background and Related Work

The focus in the current thesis is on the software V-model for the Part 6: Product development at the software level as highlighted in the Figure 2.2.

Figure 2.2: Overview of ISO 26262 as specified in [3]

2.1.1.1 ISO 26262 Part-6: Right Side of the V-model

The right side of the V-model consists of the Clauses 9, 10 and 11 [3]. These clauses depict software testing and verification performed against the clauses 8, 7 and 6 respectively as seen in Figure 1.1. All the clauses are composed of the subclauses, Objectives, General, Inputs for the clause, Recommendations and requirements to be fulfilled, and Work Products that are to be generated. These clauses and sub clauses state the requirements that are recommended to be adhered, along with notes that help in understanding and interpreting these requirements. Additionally, obligations on the corresponding quality assurance methods are also imposed based on the assigned Automotive Safety Integrity Level (ASIL) [14].

The clauses 9, 10 and 11 (i.e. right hand side of V-model) as described in Part 6 of ISO 26262 [3] is defined below:

- **Part 6 Clause 9: Software unit testing**

  The main objective here is “to verify that the implemented software units are as per the software unit design specification and do not contain any undesired behaviour.” The WPs that are to be generated for safety evidence in
this clause are software verification report, software verification specification and SVP

- **Part 6 Clause 10: Software integration and testing**
  The objectives are “to integrate the software units and generate the embedded software and to verify that the embedded software is as per the software architectural design and does not contain undesired behaviour.” The WPs that are to be generated for safety evidence in this clause are Software verification report, software verification specification, embedded software and software verification plan.

- **Part 6 Clause 11: Verification of Software safety requirements**
  The main objective is “to verify that the developed embedded software satisfies all the software safety requirements defined in part 6, clause 6. The clause 6 in part 6 of the standard is out of scope of the thesis (i.e the left hand side of the software V-model).” The WPs that are to be generated for safety evidence in this clause are Software verification report, software verification specification and software verification plan.

2.1.1.2 Safety Case: Traceable Safety Evidence

Safety case is the collection of traceable safety evidence of the fulfilment of the requirements of ISO 26262 [31]. Currently, for road vehicles above 3.5 tonnes, there is no formal requirement to produce an explicit safety case. But, by 2018, such a production would become mandatory. ISO 26262 Part 2, clause 6.4.6.2, states that “The safety case should progressively compile all WPs” [32]. As already mentioned in Section 1.1, the standard requires both process and product based safety evidence for the safety case (see Figure 2.3). Process-based arguments show that the life-cycle activities defined in the standard have been adopted. Product-based arguments show that the WPs related to the life-cycle of safety-critical systems under examination, constitute evidence that the system behaves in an acceptably safe manner. The software testing and verification results are needed to be available as arguments that show product based safety evidence (see nomenclature) for the safety case [33]. In the end it’s the WPs that are the result of realising the safety lifecycle activities and processes recommended by ISO 26262. These WP’s must be shown to ensure that the ECUs used in the end vehicle perform in a safe manner. Such a safe functioning is critical for usage by customers as well as for suppliers of the ECU systems. Hence, the focus in the thesis is on product based safety evidence in form of traceable WP types. A safety case is needed to be developed for every item (see nomenclature) that has a safety requirement with an assigned ASIL of B and above. In order to reduce the time to market and decrease the development effort, support for generating reusable safety case has to be adopted. Thus, OSLC, that supports traceability
through interoperability across the application lifecycle of a software can be used to achieve this [23][32].

ISO 26262 provides automotive specific safety integrity levels, called ASIL for classifying the risks based on a hazard analysis (see nomenclature) [22]. For the hazard analysis, an initial identification of various hazards as either safe, dangerous or dangerous and detected is performed. Then the residual risk that would be left after performing the required verification activities for the detected dangerous risks is identified. This result is used to set the ASIL [15]. ASIL is the factor of controllability, exposure and severity. The ASIL classification is used as the basis to describe necessary methods to be used for reducing the risk to a tolerant value. Level D is the maximum ASIL Level and requires more effort to reduce risk. From ASIL D the effort decreases till ASIL A, which requires the least extent of risk reduction activities [34].

To understand how ASIL is associated with the software testing WP types in ISO 26262, an overview of the context of ASIL inheritance is presented in Figure 2.4. Hazard analysis is performed on the item to define the ASIL. Then, ASIL inheritance occurs from the safety requirements to architecture design specifications and to their eventual implementation. This implementation is then tested and verified for confirmance of safety [20].

Figure 2.3: Safety Evidence in a Safety Case [4][5]

ISO 26262 provides automotive specific safety integrity levels, called ASIL for classifying the risks based on a hazard analysis (see nomenclature) [22]. For the hazard analysis, an initial identification of various hazards as either safe, dangerous or dangerous and detected is performed. Then the residual risk that would be left after performing the required verification activities for the detected dangerous risks is identified. This result is used to set the ASIL [15]. ASIL is the factor of controllability, exposure and severity. The ASIL classification is used as the basis to describe necessary methods to be used for reducing the risk to a tolerant value. Level D is the maximum ASIL Level and requires more effort to reduce risk. From ASIL D the effort decreases till ASIL A, which requires the least extent of risk reduction activities [34].

To understand how ASIL is associated with the software testing WP types in ISO 26262, an overview of the context of ASIL inheritance is presented in Figure 2.4. Hazard analysis is performed on the item to define the ASIL. Then, ASIL inheritance occurs from the safety requirements to architecture design specifications and to their eventual implementation. This implementation is then tested and verified for confirmance of safety [20].
2.1.2 Traceability and Interoperability

Traceability between the application life cycle WPs is necessary and must be maintained for ISO 26262 standard compliance [3][22]. “In general, traceability is about understanding a design right through from the origin of the requirement to its implementation, test and maintenance” [35]. It is the ability to follow through all phases of development, from the specification to development, to testing and deployment. It helps ensure the quality of a software developed. In OSLC, traceability is the ability to link work artifacts produced by one tool to another [36]. The lack of traceability between WP types can lead to problems being faced by validation teams for appropriate and necessary validation. Yet, it is seen that software development enterprises rarely maintain such traceability, where there is an explicit link between the WP types developed. A safety case, for software testing and verification should include evidence that all WP types required by ISO 26262 standard are produced. Thus, there is a need to incorporate traceability that links the involved WP types for software testing [35]. Interoperability as defined in Section 1.1, requires that traces to be defined in order to understand which WPs must be linked with each other for tools to interoperate. Thus, it is a way for realising and achieving the traceability between the WP types. Interoperability can be achieved by defining the WPs (i.e resources) in RDF, which is the machine understandable code that tools can exchange [36]. RDF for a resource consists of links with other resources in the same OSLC domain specification as well as with resources in other OSLC domain specifications.

2.1.3 Open Services Life cycle Collaboration (OSLC)

OSLC standardizes a set of data expectations and protocols that can be used for easing the data exchange between tools required for achieving interoperability [36] [28]. OSLC specifies the way in which resources (i.e WPs) of a particular tool can be accessed and browsed. It specifies a minimum amount of protocols to allow two different tools to work together seamlessly [16]. In OSLC, each resource type such as a requirement, test case, architecture specification defined in OSLC domains, can be manipulated using the standard Hyper Text Transfer Protocols (HTTP) like Get, Post, Delete, Put [19]. These types of resources in an OSLC domain specifications can be extended when necessary. The OSLC Core is the basis for all domain specifications and services described in OSLC. It consists of standard rules and patterns for using HTTP protocols on RDF that all the domain specifications must adopt. The specifications and services that OSLC core supports for the domain specifications is depicted in Figure 2.5.

2.1.3.1 OSLC Domain Specifications

The OSLC domain specifications are defined for different phases of ALM such as Quality Management (QM), Requirement Management (RM), Architecture
Chapter 2. Background and Related Work

Management (AM) etc [19]. These domain specifications must describe the types of resources and the rules for creating RDF representations for the resources relevant to the phase of ALM for which the domain is defined for. The purpose of the domain specifications is to enable integration between tools that generate the resources. Thus integration is achieved by defining the interoperability between the data (i.e. resource) in one OSLC domain and with resources in other OSLC domains [19]. In the thesis, we look at interoperability between the resources required for ISO 26262 based software testing and verification. By providing ISO 26262 specific resource types as extensions to existing OSLC domains, the services OSLC provides for the domain specifications can be used for generating the safety case in more efficient manner and thus help in achieving compliance to ISO 26262 [37].

The OSLC domain specification for verification and testing phase in ALM is OSLC-Quality Management (QM) [12]. The OSLC-QM domain already has existing resources types of test plan, test result, test script, test execution record and test case. These resource types have properties like description, contributor, title, creator among others. The resource types also have relationship properties with other resource types within the same domain and with resource types in other domains. For ex: In OSLC-QM domain, the test case has relationship properties with test plan. Also, the test case has relationship properties with architecture resource type in another OSLC domain called architecture management [38]. Thus, OSLC domains are used for realizing interoperability across various phases of ALM [36].

2.1.3.2 OSLC Resources: Resource Description Framework

OSLC resource is the shared data managed by an OSLC Service in each domain specification. OSLC is industry driven approach with the goal of building specifications based on the principles of Linked data [39]. Linked data refers to interlinking of data as published in world wide web. Linked data is published in a machine readable format like XML, RDF, JSON among others [39]. OSLC uses a
simple shared data model called RDF, for representing the resources based on the linked data initiative [40]. RDF defines the resources and their property values in a consistent and a machine readable format for tools to exchange. Each RDF describes the properties that describe the resource and the relationship properties that link it with other resources in the same or in other OSLC domains (ex: in OSLC-AM, OSLC-QM etc). OSLC uses HTTP protocols on the RDF to achieve interoperability of the data [41]. In order to define RDF, first, the structure for the resources (i.e WPs) called the resource types (i.e WPs types and their traceability links) have to be defined. These resource types are depicted in conceptual meta model must be instantiated to depict an OSLC resource in RDF. Thus, RDF is the representation of the OSLC resources based on the structure of the resource type [7].

RDF provides a standard representation for data in form of triples that facilitates the linking of resources [42]. The triples are part of the core structure of linked data. The RDF triples consist of subject, object and predicate (refer to Figure 2.6). A set of such triples is called an RDF graph [43].

Figure 2.6: RDF Triple based on [7]

2.1.4 Conceptual Meta Model

“Conceptual models are used to help us know, understand, or simulate the subject matter they represent” [44]. “Meta-Models consist of concepts, relationships, and properties with defined semantics” [28].

The concepts and elements for depicting the WP types, their attributes and relationships in the conceptual meta model by using UML class diagram is presented in this section.

In the thesis, the conceptual meta model depicts the attributes and resulting WP types for different work flows. The concepts in the model are abstractions of ISO 26262 specific WPs types, which are depicted as classes. The links between such concepts was defined using the relationship types between the classes. Thus, for facilitating in being defined as OSLC resource types, the model must be defined, so as to not only incorporate the traceability but also the abstraction of data that must be exposed for aiding in carrying out the work flows.

UML Class Diagrams

UML Class Diagrams capture the static data structure for objects. In OSLC this is the RDF structure required for OSLC resources. UML class models were
used to define the static structure for data of the ISO 26262 WP types. These WP types when instantiated are the ISO 26262 specific product based safety evidence that utilise the OSLC services for achieving interoperability [45]. In the thesis, modelling is for the work products that can be used by tools to interoperate. The class models define the interface for the tools that exchange the work products. These interfaces are defined as relationship links. The relationships between of work products help tools to understand what is to be shared for ISO 26262 compliance. The UML class diagram profile has been chosen to define ISO 26262 based concepts that represent the OSLC resource types.

The attributes for the classes in the meta model have data types which are either:

- Extensive Markup Language (XML) Literals that are used for interfacing with data on the world wide web (i.e linked data) [46]. The XML literals are of the type, integer, string, boolean, date time, double, float and decimal. XML literals have been used as they are supported by OSLC resources [47], or

- ISO specific defined enumerations. An enumeration is an user defined data type with pre-determined literal values. It is an UML element, which is used to define a list of literal values as enumeration constants [10].

The concepts in the meta model are connected with relations that contribute to the semantic value of the specifications in the meta model [44]. The various relationship types [10] used are:

- Aggregation: This is a relationship type used when the child class can exist independent of the parent class. Example: Software verification plan (parent) and Tool (child). Delete the Software verification Plan and the Tool can still be defined and used.

- Composition: This is a relationship type used when the child class cannot exist independent of the parent class. Example: Test case (parent) and test data (child). Test data cannot be used and defined on its own. It is composed in a test case.

- Dependency: This is a relationship type used when a class uses another class by a certain parameter or return type. Dependency is a form of an association relationship. Dependency relationships have been used to depict traceability links between the WP types.

Other UML concepts used:
Packages and Frames are UML concepts that are used to group elements in the class diagram under a common name, and for providing a context and outline for
the content respectively. Packages are used for hierarchical organisation of the classes. A dependency between the packages represents a dependency between the package contents. Frames are used for abstracting a name for the diagrams based on the context and extent for which the models are defined [10].

Figure 2.7: UML Class Diagram Concepts used in Conceptual Meta Model

Enterprise Architect (EA) by Sparx systems had been used for developing the UML class diagram. EA is not an open source software. Hence, the original UML class diagrams have been replicated using an open source tool and presented in the thesis report. The screen shots of the original UML class diagrams are presented in Appendix H.

2.2 Related Work

The process of defining conceptual meta models to represent knowledge to be used for achieving tool interoperability using OSLC specifications is a rather new area of research, where there is not much published work. Some of the related work in this research area has been presented below.

In [35], the authors developed a Process Assessment Model (PAM) based on ISO 15504, using Unified Modelling Language (UML) diagrams. These diagrams depict the traceability between the risk management and change management processes of the software development life cycle for medical device systems. This PAM defined the traceability required for generating the safety case for medical device software certification. The reason for defining the PAM, is to fulfill the traceability requirements for the processes used for developing the software used in medical devices. The lack of such traceability between WPs can lead to shortcomings being faced by validation teams for appropriate and necessary validation. Here, the initiative has been for using OSLC to effectively show and achieve the defined traceability relationships between the processes used for software development of a medical device through tool interoperability specifications. Expert interviews as an empirical method was used to validate the model. A trial of the usage of the model was also performed for two companies. One, a small medical device software company and another a product development and design engineering company.

In [40], the various data sources used throughout the development lifecycle activities of systems engineering have been integrated and traced. Such tracing
would ease the tasks of identifying interoperating data for quality management. To achieve this, the authors chose the OSLC initiative for defining the shared data model (i.e., RDF), using the linked data approach. A case study, using the data for a product scheme classification activity in systems engineering, was used for defining the shared data model based on the structure defined using RHSP model. RHSP is a systems engineering industry practice for information modelling and is used for knowledge management. The authors used RHSP model for representing the artefact types used in systems engineering development in a structure required for defining the OSLC-based shared data model.

In [48], User Centered Design (UCD) approach based on UML has been used for meta-modelling interoperable OSLC tool chains. Here, three tools at Scania have been considered for modelling. Using Eclipse Modelling Framework (EMF) framework code generation from the defined meta models was performed. EMF is a modelling framework used for building tools and other applications based on a structured data model (i.e., meta model). This implemented prototype was then verified by the authors using usability tests.

[45] is a master thesis dissertation that investigates the ways to depict OSLC resources using UML class diagrams. A comparative analysis was performed where alternatives of EMF, class modelling, and Ecore modelling have been investigated for the purpose of defining OSLC resource types. Although, EMF was considered easier to use, class modelling was observed to define a more correct structure for resources in form of OSLC resource types. Thus, ways of representing OSLC resource types was investigated resulting in the recommendation of UML class diagrams.

In [49], the class modelling of the artifacts and roles defined in International Electrotechnical Commission (IEC) 61131, which are used and produced by different tools, has been presented. The artifacts, which are the elements shared between the tools, and the roles which are used to investigate the various integration scenarios, have been depicted as traceable OSLC resources using UML class modelling. The case of an embedded systems development of a control wing turbine system was used to depict the artifacts and roles. This depiction in form of UML class diagrams was implemented using Simulink models, for which OSLC adapters have been proposed.

Meta Object Facility is a standard by the Object Management Group (OMG) for defining modeling languages like UML. In [50], it is used for modelling tools with other lifecycle tools to achieve tool interoperability and integration. Here, OSLC tool chain specifications are meta modelled and implemented using Eclipse Modelling Framework (EMF) for a tool called Design Manger. The implementation was done by developing an integrated developing environment using EMF. Thus, the purpose for meta-modelling was to define a basis for implementation, which was used to achieve tool integration and interoperability.

Though, it is the tools that interoperate, it is the data (i.e., resources) that is exchanged between the OSLC domains that needs to be defined to achieve such
Chapter 2. Background and Related Work

interoperability. Thus, in the current thesis, the focus was on meta modelling the OSLC resource that are exchanged between the tool and not modelling of the tool chain specification as done in [50].

The above papers, are some of the related work to the topic of study. Meta modelling for OSLC has been used for depicting the knowledge in domains like systems engineering, medical devises, IEC 61131 and ISO 15504. Yet, none have been used for depicting knowledge for a functional safety standard such as ISO 26262 standard. Also, modelling has been performed for depicting varied types of subjects in these domains such as, tool chains, processes, data sources for development activities, artifacts etc. The processes in ISO 26262 have been presented as a conceptual model using Systems Process Engineering Model (SPEM) UML modelling profile in [4]. Yet, this knowledge representation of the ISO 26262 processes has not been performed to be used for OSLC. It can also be noted that, no standardized meta modelling approach has been followed in any of the related works.

Meta Modelling using UML profiles has been popularly used in the embedded software systems domain. UML modeling profiles like Systems Modeling (SysML) were used for systems engineering modelling, SPEM have been used for defining the process flows [4], MARTE for modelling embedded real time systems, EAST-ADL that combines features of both MARTE and SysML for architecture modelling of safety related embedded systems [51]. All of them have been used for model based development but not for defining a meta model depicting a structure for OSLC resources (i.e data). Thus, UML Class diagrams used for modelling data in [44],[50],[52],[45],[48],[53] has been chosen as the meta modelling approach.

According to [54], which is one of the largest literature studies in safety evidence, encompassing 218 peer reviewed articles between 1990-2012, states the need for more practitioner oriented and industry driven studies in the area of automotive safety critical system certification. This thesis is one such initiative in the direction for safety critical system certification, achieved by proposing ISO 26262 specific WP types as an extension to OSLC domain resource types. This proposal has been presented conceptually in form of a meta model as suggested in [18],[48],[50],[40].
This section defines the research method used to achieve the aims and objectives of the study. The case study methodology was used for the study. The motivation for choosing from the research methodologies defined in [8] is explained below:

*Experiment* has a major prerequisite of an initial hypothesis. There is no initial hypothesis in the current study. Further, it requires performing evaluations in a laboratory or simulated environment. Thus, controlled experiment as a research method has not been chosen [55].

*Surveys* are suitable for targeting a population to answer specific and precise questions to generalize the results to that selected population. Thus, it is not chosen for the current study.

*Action Research* as the research methodology was not considered since the researcher is not part of the change process that involves solving of a real world problem. The biggest challenge for action research is its immaturity as an empirical method. Although frameworks for evaluating action research have been proposed, they tend to be vague or subjective. Also, the role of theory in solving the problems is not well elaborated in action research (as well as in Technical Action Research (TAR)). Thus, it has not been chosen for the current study [56].

*Case Study* is an empirical investigation of a contemporary phenomenon in its real-life context conducted especially when the boundaries between phenomenon and context are not clearly evident. Exploratory case studies are used to explore theories to generate and build new ideas and theories. Explanatory case studies are used to investigate and explain existing theories or ideas. Thus, case study, which has a more flexible research design was chosen as the research methodology [8].

### 3.1 Research Design

The research design is explained in the Figure 3.1 below.
Chapter 3. Research Method

Figure 3.1: Research Design

The research is carried out in three steps. The case study with two cycles of
data collection and data analysis and the validation of the case study results.

3.1.1 Case Description

Scania is a Swedish automotive industry manufacturer of commercial automotive vehicles like trucks and buses, which are over the gross weight of 16 tones. Scania manufactures the safety critical ECU systems used in the trucks and buses. It is an example of a typical case of a company [57], that would be impacted by the 2018 version of the ISO 26262 standard [13]. The thesis has an embedded case study [8], with different units of analysis for the two cycles of the case study. The different units of analysis are defined under their respective cycles. Refer to Figure 3.2 that depicts this. For the second cycle, a separate case description is provided to understand the context of the units of analysis.

![Figure 3.2: Embedded Case Study based on [8]](image)

3.2 First Cycle: Data Collection and Analysis

The protocols for data collection and data analysis used in the first cycle are explained in this section.

3.2.1 First Data Collection Cycle

The sources of data for the first cycle are from third degree data sources of documents at the company [58]. The latest published version of the entire ISO 26262 standard available at the company has been considered. The sources of data also include an archived document, which is an internal classified power point presentation on software testing and verification part of the ISO 26262 available at the company. Further, the OSLC Quality Management official web page [38] was also used for identifying requirements for describing resource types as required by OSLC. The three data sources used in this cycle are qualitative and pertaining to the ISO 26262 standard. The first cycle is an exploratory case study used for generating the hypothesis [8].

Selection of Documents
Chapter 3. Research Method

Documents used for analysis can be any electronic or printed material like presentations, diaries, charts, books, manuscripts, newspapers among others [59]. Convenience sampling [60] was used to retrieve all available documents on ISO 26262 for software testing. The assistance of the external supervisor at the company was used to retrieve the complete set of data sources. Case studies [8] often include using confidential information in an organization. The ISO 26262 standard is not open source and thus is not presented fully in the report. It is yet to be published and made mandatory for road vehicles over 3.5 tones. The published version of the standard, for road vehicles within 3.5 tones has been used for the analysis.

3.2.2 First Data Analysis Cycle

Qualitative Inductive content analysis as suggested by Satu Elo and Helvi Kyng in [61] has been chosen as the analysis method. Qualitative inductive content analysis is mainly used to describe a phenomenon or theory in form models, maps, or systems. It involves systematic coding and categorizing to determine patterns in texts in order to define the relationship between them. The outcome of the analysis is categories and their relationships describing the theory. The aim of the first data analysis cycle was for defining the work product types and their relationships in form of a conceptual meta model. Thus, qualitative inductive content analysis was used for generating the hypothesis in the first cycle of the case study [62][63][61]. The qualitative inductive content analysis method consist of three main phases: Preparation, Organising and Reporting.

3.2.2.1 Phase 1: Preparation

Preparation phase involves choosing the unit of analysis, interpreting and making sense of the data in it based on the research aims and objectives. The aim and objective while reading the unit of analysis were,

- To identify the WPs that are required to be produced for ISO 26262 based software testing and verification of an ECU system ,
- To identify the text that describes information that must be contained in the WPs in order to define them as WP types ,
- To identify the text that describes association or interactions that inter relate and link the WPs.

Finally, define and represent the above identified aspects in form of a conceptual meta model.
Figure 3.3: Qualitative Content Analysis: Inductive Content Analysis

**Unit of Analysis**
The unit of analysis was focused on the clauses 9, 10 and 11 pertaining to software testing and verification in Part 6 of ISO 26262 standard and in the power point presentation. The specifications in the software testing and verification clauses involve information from ISO 26262 Part 8, clause 9, which was also taken into consideration. The official OSLC quality management web page [38] specifying resource properties that are normative for defining OSLC resources types have also been taken into consideration.

### 3.2.2.2 Phase 2: Organising

This phase involves organising of the data for open coding, grouping, categorising and abstracting as presented in the following sections.

**Open Coding**
The data in the documents was read through several times to identify and define the tagged text that encompasses all work product based aspects of the data. These tagged text were then open coded for depicting the data from the perspective of the research aims [61]. This process of open coding involves defining codes that describe the WP types and codes that describe the relationships between the WP types. Such codes that relate WP types with each other are called connector codes. Connector codes were defined based on a linguistic approach in [64] that looks for words and phrases that define relationships between text. For example, consider the text “In software integration testing, to evaluate the
completeness of test cases and to obtain confidence that there is no unintended functionality, the structural coverage shall be measured using metrics, which are recommended based on the ASIL. If the achieved structural coverage is considered insufficient, additional test cases shall be specified.\textsuperscript{[3]} Here, the text that was tagged based on the research aims was “metrics for structural coverage for evaluating the test cases”, “test cases”, “software integration testing”, “software architectural level”, “coverage metric based on ASIL”. These tagged text defines the codes of ‘test case’, ‘structural coverage’, ‘software architecture’, ‘test level’, ‘method name for structural coverage’, ‘ASIL’. It defines the connector code ‘used for verifying’ as a traceability link between the ‘test case’ and ‘structural coverage’. This open coding process continues till no new codes can be identified. This was called saturation where as more and more text is analysed, no new codes can be extracted. This was the exit criteria for open coding \textsuperscript{[65][66]}.

\textbf{Grouping}

This stage involves grouping of codes into higher order headings. It is a way of identifying the patterns in the open codes derived from the previous stage. A common and a more generalised heading depicting the pattern in the codes was determined. For example, the codes, ‘method name for structural coverage’, ‘ASIL’, ‘structural coverage’ and ‘test level’, all describe some attribute or property of structural coverage and were grouped together into a higher order heading called ‘structural coverage’. At this point the two codes of ‘title’ and ‘identifier’, which are the properties required to be described for every OSLC resource types were grouped along with the other codes used for defining the higher order headings. Although, OSLC specifies a number of properties for describing resource types, title and identifier are the only two which are mandatory \textsuperscript{[38]}.

\textbf{Categorizing}

The aim of grouping data was to reduce the number of codes by collapsing those that are similar or dissimilar into broader higher order headings. These higher order headings are called categories in inductive content analysis. Thus, from the previous stage, the grouped codes defined the higher order heading ‘structural coverage’, which was an example of a category. The categories were depicted as classes in the conceptual meta model. The difference between grouping of codes and categorizing stage was that categorizing involves further identification of patterns in the categories to define main categories. A category refers to a level of description of the text, where the category can be used to express the content of the text \textsuperscript{[62]}. When categorizing by inductive content analysis, the researcher comes to a decision, through interpretation, as to which categories put together define a main category \textsuperscript{[61]}. For example, the categories, ‘tool’, ‘test object’ and ‘test environment’ can be defined into the main category ‘software verification plan’. These categories were all required for planning the software testing and
verification and thus can be expressed within the context of ‘software verification plan’.

**Abstraction**  Abstraction means formulating a general description through categorizing and exposing information required for a particular work flow [61]. As seen, each category was named based on the common characteristic of its content. Similarly, these categories were further abstracted into main categories. This process occurs till all similar categories were collapsed together and no more main categories could be abstracted. An example was where the main categories of, Software Verification Plan, Software Verification Specification and Software Verification Report, consisting of all the software testing and verification WP types were abstracted as ‘ISO26262 quality management resource types’. Here, the connector codes between the main categories of SVP, SVS and SVR are used to define the traces that expose the interacting resource types for software testing work flow. Similarly for the other categories the abstraction is defined.

### 3.2.2.3 Phase 3: Reporting of Results

This was the last phase in the analysis process where the results of the analysis were defined. The various categories and main categories defined were correlated and conceptualized. This description was in form of a conceptual meta model that depicts the WP types(i.e categories), their attributes (i.e codes) and the their relationships (i.e connector codes) in a structure required for OSLC domain resource types. Thus, the final outcome of this cycle was the RDF based conceptual meta model for software testing and verification clauses of ISO 26262.

### 3.3 Second Cycle: Data Collection and Analysis

The protocols for data collection and data analysis used in the second cycle are explained in this section.

The hypothesis formulated in the first cycle of data analysis was tested, refined and extended in the second cycle data analysis using the qualitative deductive content analysis. According to Satu Elo and Helvi Kynga in [61], the context used for hypothesis generation must differ from the context for testing the hypothesis. The meta model has concepts that define the software testing and verification of ECU systems according to ISO 26262 and its related presentation at the company. Thus, it was tested for its usability in the context of software testing and verification performed for a truck ECU system at the same company. In [67], a similar approach of qualitative content analysis was performed, where a model was generated from theory related to competencies in the informatics domain. Next, the
model was tested and refined to define additional competencies by conducting interviews.

3.3.1 Case Description

Main (M) is a family of ECU systems, which along with other families of ECU systems is used for realizing the Fuel Level Display System (FLDS) functionality for heavy vehicles. Currently, the Main family has only one ECU in it, Main1 (M1), as depicted in Figure 3.4. There are 4 variants (see nomenclature) of FLDS used in trucks and buses. Here, M1 is used in variant 1, which is truck with liquid fuel. The FLDS for variant 1 consists of two main functions: displaying the estimated fuel level and the low fuel level warning. These two main functionalities are assigned to logical software elements called Allocation Element (AE) at the company. Here, AE1 and AE2 are assigned the two main functions stated above, respectively. The M1 ECU system software is used for realizing both AE1 and AE2 (see Figure 3.4 below for reference).

Within M1 ECU system there is a software module, M1: Fuel, that is used for managing the application software of the M1. It is for this application software, that the WPs involved during software unit testing were identified and defined. The M1 ECU also consists of software modules in the basic middle ware layer such as Real Time Database (RTDB) variables and other signals. This middleware software is common for all ECUs developed for trucks at Scania. Thus the WPs involved during the testing and verification of integrated software modules in M1 ECU (ie. the M1:Fuel application software with the basic software) were determined as software integration testing based work products. The M1 ECU also comprises of Comp and other hardware components (see Figure 3.5), which were out of scope for the thesis [68][69]. The WPs involved for software testing of M1 ECU system was the focus of study in the second cycle. The second cycle
Chapter 3. Research Method

is an explanatory case study used for testing the generated hypothesis from the first cycle [8].

Figure 3.5: M1 ECU System

M1 ECU system used in FLDS was an example of a typical case of the way software testing and verification was performed for large number of ECUs developed at the company. Thus, M1 ECU been selected as it a representative case [57]. Further, the FLDS is a published functionality at the company.

3.3.2 Second Data Collection Cycle

First and third degree data sources like interviews, documents [58][70] along with observations [8] were used in this cycle. Thus, triangulation of the various data sources has been done to improve the reliability of the data collected and analysed. It helps to reduce bias and improve credibility [59]. The second data collection cycle takes place after the conceptual meta model was defined from the analysis of the first cycle. The focus of the data collection here was to firstly, identify and define data for populating the concepts in the meta model with respect to software testing and verification of M1 ECU system. Secondly, for extending and refining the concepts in meta model with respect to software testing and verification of M1 ECU system [63].

3.3.2.1 Observations and Documents

Observations were used for discovering the behavior and understanding of the participants involved in performing software testing and verification for M1 ECU system. The participants were asked questions such as “What is your plan for verifying ECU softwares?”, “What are the smallest software parts you test for?” etc. Such questions were used to make the participants to think aloud their thoughts as suggested in [58]. Thus, observations were made through think aloud sessions, without the researcher getting involved in the activity being observed.
Observations aided in formulating the questions for the interviews. Further, an internal master thesis dissertation [9] on ISO 26262 gap analysis for Scania, was also used for mapping the terms between ISO 26262 and Scania. This was needed for formulating questions with terms familiar to the interviewees in order avoid misunderstandings. Look at Table 3.1 for the mapping.

<table>
<thead>
<tr>
<th>Scania Term</th>
<th>ISO 26262</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td>M1 ECU system</td>
</tr>
<tr>
<td>Element</td>
<td>Any part of the M1 ECU (i.e software unit, component etc)</td>
</tr>
<tr>
<td>Module Testing</td>
<td>Software Unit Testing</td>
</tr>
<tr>
<td>Module Integration Testing</td>
<td>Software Integration and Testing</td>
</tr>
<tr>
<td>Assumption</td>
<td>Software Safety Requirements (SSR)</td>
</tr>
</tbody>
</table>

The other documents considered were the reports generated for the WPs of test cases, test reports, tool execution logs, calibration data, allocation element specification [69], configuration data specifications, requirement specification [68] and design specification involved for software testing and verification of M1 ECU system. A total of 8 documents that were available for the software testing and verification related WP produced for M1 ECU system were used. Complemented by interviews, these documents have helped the researcher in identifying and defining the data codes for the attributes of the WP types. Access to these documents have been requested from the concerned interviewee or have been obtained through the official document management system at the company. Due to confidentiality not all documents have been referenced.

3.3.2.2 Interviews

Both closed ended and open ended questions were formulated for the interviews. Close-ended questions (see Appendix B), were aimed at identifying the WPs and their attributes involved during software testing and verification of M1 ECU system. Semi-structured interviews [71] with open ended questions (See Appendix C) were used for defining the WPs and the data values for their attributes. Both open and closed ended questions were formulated based on the WP types and attributes defined in the conceptual meta model.

The interviews have been designed as:

- Invitation to participate in an interview meeting.
- General information of the interviewee participants.
Chapter 3. Research Method

- Introduction of the topic and the description of the WP types in the conceptual meta model that were the basis for formulating the questions.

- Data collection from open and closed ended questions using audio recordings and field notes.

Selection of Interviewees
To ensure completeness of the data collected, all subjects involved at each software test level of M1 ECU system were chosen as the sample population. Thus it included software testers, developers and test engineers working at three software test levels of software unit testing, software integration testing and SSR verification testing. Convenience sampling [60] with the assistance of the external supervisor at the company was used to identify these involved stakeholders at the three test levels of M1 ECU system. Further, the company online search portal was useful in searching and finding the contact information of the relevant people. A meeting invitation (Appendix A) was sent to the involved subjects stating the goal of the study. The anonymity of all the interview participants was preserved.

Formulation of Questionnaire
The categories and codes defined in the conceptual meta model were used for formulating the questionnaires. General information related to the interviewees was initially collected to understand their background. This was also used to aid in formulating and asking the questions. The questions have been listed in an interview guide (see Appendix B and C). The interview guide was used to make sure that all relevant questions were asked to the interviewees [72].

Frame/Re-frame Questions for Interview
The researcher formulates the questions for the interview from the data, as mentioned in the previous step. Observations further helped in refining the questions to be more understandable to the interviewee under consideration. The direction of the information given by the interviewees was assessed by the researcher to decide the order in which the questions were to be asked. As three different software test levels were considered, some of the open ended questions were asked specific to the involved stakeholders.

Transcription of Data
Transcription for the interview data was key for identifying the complete set of codes. Transcription were done manually, word by word. For convenience and to reduce the chances of missing information, interviewees have been audio recorded along with taking field notes. Once each interview was conducted, it was immediately transcribed [73] from the audio recordings using the open source software, Express Scribe (See Appendix E for example). Interviews that could not be audio recorded have been transcribed soon after the interview.
3.3.3 Second Data Analysis Cycle

This section explains the WPs involved for software testing and verification for the case of M1 ECU system. The second cycle was an embedded case within the case of Scania as depicted in Figure 3.2.

Qualitative Deductive content analysis is used for testing existing models, concepts or categories. This analysis involves a methodological controlled assignment of the categories and their respective codes to a passage of interview transcript or document data. Thus, in the second cycle, interview transcriptions and document data were brought in connection with the categories and codes defined in the conceptual meta model from the first cycle. This was done in order to instantiate the meta model with software testing and verification WPs for M1 ECU system. These WPs were the OSLC resources, which are represented in RDF. Thus, the results of the deductive content analysis was for instantiating for testing and extending the concepts in the meta model. Deductive content analysis method consist of three main phases: Preparation, Organizing and Reporting.

![Diagram of Qualitative Content Analysis: Deductive Content Analysis](image.png)

**Figure 3.6: Qualitative Content Analysis: Deductive Content Analysis**

**Selection of Data Analysis Method**

Qualitative content analysis involving both inductive and deductive content analysis has been chosen for the thesis. This was because study requires defining a
conceptual meta model and then testing its usage to define the OSLC resources (i.e RDF). Qualitative Content analysis is suitable for a lower level of data interpretation as required in the study, than the grounded theory analysis method. Grounded theory involves an in depth and higher level of data interpretation [62][63]. Although content and thematic analysis are similar in terms of the results derived, the reason for opting for qualitative content analysis was because it defines the data based on the emerging categories. Thus, it can be used to generalize and abstract the defined categories as required for generating the conceptual meta model.

3.3.3.1 Phase 1: Preparation

This phase involves the choosing of the unit of analysis, and making sense of the data in it based on the research aims and objectives [61].

Unit of Analysis
The interviews comprised of both closed ended and open ended questions for collecting data. Documents related to the various WPs used or produced during software testing and verification of M1 ECU were also used as a data sources. All these data sources were triangulated during analysis [74]. The number of interviewees for closed ended questions (i.e. the data points) were not high enough for performing statistical analysis. Thus, upon researcher consultation with the internal supervisor, the data collected from the closed ended questions was analysed using a table [75]. Open ended questions were formulated specific to the involved stakeholder (i.e interviewee) at the different software test levels under consideration. Thus, the unit analysis were the interview transcripts and documents for the WPs produced or used for software testing and verification M1 ECU system.

3.3.3.2 Phase 2: Organising

This phase of deductive content analysis involves the stages presented below.

Development of Structured Analysis Matrix
In deductive content analysis a structured analysis matrix is developed based on previously defined theories, models or mind maps [61]. In the thesis, the conceptual meta model defined from the first cycle was used for defining the structured analysis matrix. Based on the research aim to instantiate and test the meta model, the structured analysis matrix was used as a basis for extracting data codes from the interview transcripts and documents. The matrix was depicted using cross tabulation [75], in form of a table (see Figure 3.7), consisting of the concepts defined in the meta model. Thus, the codes (i.e attributes) and categories (i.e classes) were depicted on the X-axis and the data coded from the
Chapter 3. Research Method

interviews and documents for the three software test levels, on the Y-axis of the table.

<table>
<thead>
<tr>
<th>Meta Model Concepts</th>
<th>Data coded from the Units of Analysis in Second Data Collection Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categories (classes)</td>
<td>6-9: Software Unit Testing</td>
</tr>
<tr>
<td>Software Verification Plan (SVP)</td>
<td>6-10: Software Integration Testing</td>
</tr>
<tr>
<td>Tool</td>
<td>6-11: Verification of Software Safety Requirements</td>
</tr>
<tr>
<td>Test Environment</td>
<td>Title:</td>
</tr>
<tr>
<td>Test Object</td>
<td>Pass criteria:</td>
</tr>
<tr>
<td>Structural Coverage</td>
<td>Exception handling action:</td>
</tr>
<tr>
<td>Test Case</td>
<td>Regression Strategy:</td>
</tr>
<tr>
<td>Test Data</td>
<td>identifier:</td>
</tr>
<tr>
<td>Test Method</td>
<td></td>
</tr>
<tr>
<td>Configuration Data</td>
<td></td>
</tr>
<tr>
<td>Software Verification Report (SVP)</td>
<td></td>
</tr>
<tr>
<td>Calibration Data</td>
<td></td>
</tr>
<tr>
<td>Architecture design</td>
<td></td>
</tr>
<tr>
<td>Specification (ADS)</td>
<td></td>
</tr>
<tr>
<td>Software Safety Requirements (SSR)</td>
<td></td>
</tr>
<tr>
<td>Software verification</td>
<td></td>
</tr>
<tr>
<td>Specification (SVS)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.7: Structured Analysis Matrix: Categories and codes

An example of the attributes (i.e corresponding codes) for the category, SVP, is presented in the structured analysis matrix in Figure 3.7. Similarly, codes for the other categories in the matrix have been depicted in the populated matrix of Table 4.4 presented in the Section 4.3.

Data Coding According to Structured Analysis Matrix

After defining the structured analysis matrix, the text from the interview transcripts and documents was coded into data that fits into the categories and codes of the matrix. Although, a structured analysis matrix was used as the basis for data coding, it is possible to inductively define new codes when the data does not fit for into the categories and codes in the matrix. Thus, the new aspects in the data, that are relevant to the research aims, can be coded and categorised based on the principles of inductive content analysis described in Section 3.2.2. For example, consider the category tool, which consist of codes, ‘title’, ‘guidelines used’ and ‘identifier’. From the interview transcripts and documents, data related to these codes was identified and coded. In the interview transcripts, new aspects of whether the tool is built at the company or commercially bought was identified. This data could be important for tool qualification requirements of ISO 26262 and thus be used for achieving or showing product based safety evidence. Hence, this data was defined as new code and grouped with other codes under the category of ‘Tool’.
Hypothesis Testing
The usability of the conceptual meta model for providing a structure for software testing and verification of an ECU system is tested by instantiating it with data for M1 ECU system. The categories and their codes defined in the meta model are tested by extracting the data defined for the same intent from the unit of analysis. Further, any additional data that can be used for showing WP based safety evidence have been inductively coded and categorised as extensions to the existing concepts in the meta model.

3.3.3.3 Phase 3: Reporting of Results
This is the last phase in the analysis process for the second cycle. The various categories defined were tested, extended and refined. The classes and their attributes described in the meta model were instantiated and defined in RDF. This was to test the usability of the WP types in conceptual meta model in defining OSLC resource types. The outcome of this phase is the extended and tested conceptual meta model depicting OSLC domain resource types. The outcome for the second cycle was also the definition of the OSLC resources (i.e RDF) based on the instantiated meta model concepts.

3.4 Validation of the Conceptual Meta Model
In order to evaluate the effectivess of the conceptual meta model, the purpose along with the situation of its applicability needs to be understood. Applicability refers to practical usability of the model. The deductive content analysis in the second cycle has tested the usability of the conceptual meta model for instantiating data that can be defined in RDF. Now the content of the meta model has to be validated. The criteria of confirmability, traceability, interoperability and abstraction are proposed to evaluate the content of the conceptual meta model. These criteria are to be met by the conceptual meta model to achieve its purpose. The purpose is for defining traceable ISO 26262 specific WP types for software testing and verification of ECU system, that can be proposed as resource types for extending OSLC domains [15].

Since the concepts in the meta model were tested and extended based on the case of M1 ECU system developed at Scania, the validation is performed at the company itself to maintain relevance. The chosen criteria are formulated as closed ended questions to be answered by subjects knowledgeable in ISO 26262 standard. The answers to this questionnaire would be evaluated to understand the extent to which the conceptual meta model is effective in complying with ISO 26262 [76][8].
3.4.1 Criteria for validation

The motivation for choosing the criteria for validation is presented below:

1. **Confirmability**: Confirmability is the property of the model by the virtue of which it can be verified to conform to the specifications for which it has been defined. Confirmability for the conceptual meta model have been defined in two contexts, which were:
   - Context 1: With respect to ISO 26262, the confirmability was for the specifications in the ISO 26262 Part 6, clauses 9, 10 and 11 (i.e Right hand side of the ISO 26262 software V-model) and the Part 8, clause 9.
   - Context 2: In OSLC, each exchanged resource has to adhere to the resource specification. This specification explicitly defines the structure for the resources, called the resource type. The resource types must specify the resource properties and relationship properties. The resource specification states that each resource should have a subject, and set of properties, data types for the properties which are either XML literals or another resource [16]. Therefore, the WP types in the conceptual meta model must confirm to this structure for them to be represented as resource types in OSLC domains. This has been validated by defining the RDF for the resources of M1 ECU software testing. This was followed by validating the defined RDF syntax using the RDF validator of the World Wide Web Consortium (W3C) [43]. The W3C is the main working body that defines RDF (See appendix F for an example).

2. **Traceability**: In OSLC, traceability [35] among WP types can be depicted using the relationship properties of the resource types. Thus, the WP types in the conceptual meta model require efficient traceability links that help to understand inter related WP types according to ISO 26262.

3. **Interoperability**: An efficient solution for interoperability as defined in Section 1.1 supports traceability. Thus, in the conceptual meta model, it is the traceability links defined that aid in understanding the WP types that must be interoperated among the tools, for performing software testing. Interoperability requires that the WPs be exchanged. Thus this criteria was realised by defining the instantiated data for the WP types in RDF. The RDF is the data that tools can exchange in OSLC for interoperability [28].

4. **Abstraction**: A model is the abstraction from a certain viewpoint. It is the depiction of information at a certain level of detail based on the viewpoint [77]. For the conceptual meta model, WP types required for different software testing work flows of ISO 26262 was the viewpoint under consideration. For ex: software testing as per ISO 26262 requires a flow between the WP types of SVP, SVR and SVS. It also requires a flow between the software testing WP types and non
Chapter 3. Research Method

software testing WPs types against which testing is being performed according to ISO 26262 software V-model (see Figure 1.1). Thus, the depiction of the meta model must be abstracted in way as to identify the work flows.
Chapter 4

Results

4.1 Summary of Results for First Cycle

The summary of results for the process of inductive content analysis for the first cycle is presented in this section.

4.1.1 Open Coding

A total of 43 unique codes have been identified from the data. Unique codes have been extracted by identifying the text that pertains to describing the WP types in the unit of analysis. Connector codes have been extracted by identifying text pertaining to the way these WP types are associated with each other. All codes have been extracted by manually analysing the documents in the unit of analysis.

Table 4.1: Open coding: Extracted Open Codes

<table>
<thead>
<tr>
<th>Unique Codes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Test level</td>
<td>Pass result</td>
</tr>
<tr>
<td>Methods for structural coverage metrics</td>
<td>Methods for test case derivation</td>
</tr>
<tr>
<td>Methods for testing</td>
<td>Guidelines used for tools</td>
</tr>
<tr>
<td>Tool</td>
<td>Test case</td>
</tr>
<tr>
<td>Structural coverage</td>
<td>Test method</td>
</tr>
<tr>
<td>Method names for target environment</td>
<td>Tailoring</td>
</tr>
<tr>
<td>Test environment</td>
<td>Pass criteria</td>
</tr>
<tr>
<td>Valid values</td>
<td>Input data</td>
</tr>
<tr>
<td>Expected Output</td>
<td>Dependencies</td>
</tr>
<tr>
<td>Intent</td>
<td>Calibration data</td>
</tr>
<tr>
<td>Test data</td>
<td>Configuration data</td>
</tr>
<tr>
<td>Architectural Design Specification (ADS)</td>
<td>Software safety Requirement</td>
</tr>
</tbody>
</table>
### 4.1.2 Conceptual Meta Model

The defined codes and connector codes have been inductively analysed to be presented as classes, their attributes and traceability links in the conceptual meta model. The process of inductive content analysis followed has been presented in the Chapter 5. The conceptual meta model as the result of this analysis has been presented and explained in this section.

The description of the concepts in the meta model based on the ISO 26262 standard [3] presented in this section are:

#### The Classes in the Conceptual Meta Model

- **Software Verification Plan (SVP):** The plan must consists of all the WP types that are required for carrying out the software testing and verification as specified in ISO 26262. These WP types are tool, test environment, structural coverage, test method and test object. The plan must encompass the criteria for evaluating the tests and the way in which the exceptions or changes must be handled.

- **Tool:** The tool class describes the various tools and frameworks used for verification and testing of the software at the three test levels. It is the means used for coding programs that test the functionality of the software units, software components or an embedded software.

- **Test Object:** The test object is the realisation of the ADS or requirements specifications. The test object must fulfil the WP type against which the verification or testing is performed. The test object differs for each test level. It could be model or code, which is the software unit. The software unit must fulfil the
software unit design specification at the software unit testing level. It could be a software component, which must fulfil the software architecture design at software integration test level. It is the embedded software, which must be verified for fulfilling the SSRs at the SSR verification test level.

**Test Environment:** An environment for verification can be a test or simulation environment. This is the target environment in which verification and testing is to be performed. The target environment must be as close as possible to the actual environment in which the software is used.

**Test Method:** The test method describes the way in which verification or testing is performed to check for compliance with the corresponding ADS or SSR. The test method is for showing confidence that unintended functions are not present.

**Structural Coverage:** Structural Coverage is used to ensure that all the required functionalities to be implemented by the test object are covered in the test cases. The completeness of the test cases is assessed by the structural coverage metrics. This is to ensure the completeness of the testing process.

**Software Verification Specification (SVS):** The SVS must consist of the methods used for testing, the test objects used at the various software testing levels, the test cases written for the test objects and the environment in which the test cases are executed.

**Configuration Data:** This is the data that is specified for the software build and is used to control the software build process. It is used for defining values that enable controlled changes in the behaviour of the software used for different applications. The calibration is dependent on the configuration of the software. The configuration data is included in the test case.

**Test Data:** This is the data or script that consists of programming code related to the test object being tested. This test data or code is composed and a part of the test case.

**Test Case:** Test cases must consist of the test data, the data pertaining to the configuration of the test object and the environment in which the tests are executed. Further, they must also be associated with the structural coverage metrics which are used for verifying their completeness.

**Calibration Data:** The calibration data specifies parameters that encompass a wide range of input values to test the software. It is to ensure that the configured software performs as expected and in a correct way. Thus, calibration is done to ensure correct operation of the configured software under various conditions.
Chapter 4. Results

Architectural Design Specification (ADS): These are the specifications against which the testing of the implemented software (i.e., the test object) is performed in clause 9 and 10 of ISO 26262 Part 6. For Clause 9, the software unit test level, the software unit is tested against the software unit design specification. For clause 9, which is the software integration test level, the software component is tested against the software architecture design specification. Since, these are the WPs types on the left side of the V-model (Clauses 7, 8), which is not the focus of the thesis, they have not been further detailed.

Software Verification Report (SVR): The SVR must consist of the evaluation and execution performed at each software test level. It must include the result of the tests, the version of the test report generated and any tailoring rules, if applicable. The report must consist of the environment for test execution, the tools and the data used for calibrating the configured software.

Software Safety Requirement (SSR): These are the requirements specified in ISO 26262 Part 6, clause 6 (left side of the V-model). These SSR must be verified to have been realised in the embedded software. The SSR is not detailed further as it is not a WP type described in clauses 9, 10 and 11 (i.e., right side of the V-model). At the concerned department in the company, requirements are not specified explicitly for the software. They are yet to be classified as safety related or not. The software part of the Allocation Element Requirements (AERs) has been considered. The AERs are implementation specific requirements specified at the company. AERs compose of specifications for both the software and the hardware. Thus, the software specific AER specified at the company are assumed as the SSRs in the thesis.

The Attributes in the Conceptual Meta Model

- Attributes with XML literals as data types

  Tailoring: This is the description of the rationale or justification in case of failures in any of the tests. It can also be used to describe suggestions for changes to avoid such failures.

  Regression strategy: A regression strategy specifies how testing is repeated after changes have been made to the item or element (see nomenclature). Testing can be repeated fully or partially and can include other items or elements (see nomenclature) that might affect the results of the testing.
Chapter 4. Results

Figure 4.1: Conceptual Meta Model
Pass Result: This is an unambiguous statement that describes if the testing or verification has passed or failed.

Guidelines Used: For all the tools used for testing, unambiguous and clear steps on the way the tools must be used are specified as guidelines.

Dependencies: This is the description of the various inter dependencies that exist between the configured software and the calibration data. These dependencies are mostly functional dependencies.

Intent: This is the description of the context or application of usage of the configured or calibrated data, while testing.

Pass Criteria: This is the description used as the basis for declaring the tests to have passed or failed.

Exception handling action: This is description of the actions to be taken if anomalies like errors or exceptions are detected.

Expected Output: This describes the expected behaviour which can include acceptable ranges of output values. This expected output is determined based on the configuration software of test object. It is included as part of the test data defined in the test case.

Input values: This is the description of the the input data used for testing. Such input data can be signal values, sensor values, variable values etc.

Valid Values: This is used to describe the correct and appropriate ranges of values using scales and units. The scales and units are used for supporting the functional application for configuration or calibration.

Identifier: It is an ambiguous reference to the WP under consideration. Identifier is a common and mandatory property required to be specified for an OSLC domain resource type [78]. This property value is used for uniquely identifying a resource in a data source by the tools. It is an implementation specific property used for performing OSLC core services (see Figure 2.5) on the OSLC resources [48]. Thus, this property would be populated only when the proposed extension of OSLC QM resource types is implemented.

Title: This is the name of the resource or WP. This attribute is a common property defined for all OSLC resource types. This attribute value must be mentioned for all OSLC resources and has thus been included [78].
Version: This is the reference to the version of the test object being verified or tested and its corresponding version of the software verification report. Different versions of the test object could be tested to generate corresponding version of SVRs. Thus, these various versions need to be accounted for.

Unique identifier: This is an unambiguous and unique reference to the test cases developed for software testing and verification. This attribute is different from the attribute identifier, as it a requirement from ISO 26262 for test cases. It is an attribute required to be defined for test cases in ISO 26262 standard.

- Attributes with ISO 26262 Specific Enumerations as Data Types:

  **Automotive Safety Integrity Level (ASIL)**, It is one of the four levels to specify a systems needed requirements of ISO 26262 safety measures (methods). These methods are to be applied for avoiding an unreasonable residual risk. Level D requires the most stringent measures and A, the least stringent methods. The standard specifies the values A,B,C and D for ASIL.

  **Test level**, This attribute is used to indicate the software test level under consideration. There are three different software test levels according to ISO 26262 part 6, clause 9, 10 and 11 (i.e right hand side of the software V-model). The values for the test levels are software unit testing, software integration testing and verification of SSRs testing.

  **Method Name**, This attribute is used to specify the name of the method recommended based on the ASIL. There are various ranges of methods defined for various activities. These methods are for activities of software testing, test case derivation, for structural coverage and for the types of target environments used. These range of methods recommended by the standard are defined in their respective ISO 26262 specific enumerations (see Figure 4.2) as elaborated below.

  The recommended test methods stated in the ISO 26262 standard are requirements based test, interface test, fault injection test, resource usage test and back-to-back comparison of model and code for model based development.

  - Requirements based test: The requirements are the basis for what is to be tested and how to perform the tests.
  - Interface test: These tests are used to communicate with other software or hardware through an interface. Interface is a simulated software that consists of sets of messages, commands, signals, and other features that allow communication.
• Fault injection test: These tests involve introducing arbitrary faults to test the safety mechanisms. For ex: corrupting variables, introducing mutations, corrupting CPU registers, signals, corrupting software etc.

• Resource usage test: These tests are used to evaluate the fulfilment of the functionality based on values of average and maximum processor performance, minimum or maximum execution times etc. These tests mostly involve an emulator which acts as the target processor on which the test object is tested.

• Back-to-back comparison test between model and code: Here, the test requires the model that simulates the functionality of the software units or components. Then the model and the code are simulated in the same way and the results are compared.

The recommended test case derivation methods stated in the ISO 26262 standard are analysis of requirements, generation and analysis of equivalence classes, error guessing and boundary value analysis.

• Analysis of requirements: Here, the test cases are derived based on requirements written in natural languages or using formal notations.

• Generation and analysis of equivalence classes: Equivalence classes is where the inputs and outputs are divided into classes, so as to select the representative test value from each of the divided classes.

• Error guessing: In this method, test case are written based on personal judgement and experience.

• Boundary value analysis: The test cases are derived based on the various range of values obtained from the boundaries of the equivalence classes.

The recommendation for structural coverage metrics stated in the ISO 26262 standard are statement coverage, call coverage, function coverage, branch coverage and Modified Condition/ Decision Coverage (MC/DC)

• Statement coverage: This method is used to make sure that every statement in the code is executed at least once.

• Call coverage: The method refers to the percentage of software function calls covered in the implemented software code.

• Function coverage: This refers to the percentage of executed software functions. This evidence can be achieved by appropriate software integration.

• Branch coverage: This method is used to verify that each branch or output of a decision is tested. It is the percentage of branches of the control flow that have been executed.
Chapter 4. Results

- Modified Condition/Decision Coverage (MC/DC): This type of coverage is highly recommended for the most critical software parts (i.e. for ASIL D).

The recommended methods for target environment stated in the ISO 26262 standard are software in loop, hardware in loop, ECU emulation and vehicle tests. These environments indicate the way in which the test conditions are simulated for testing.

- Hardware in Loop (HIL): In simulation tests for embedded system, hardware in the loop simulation deals with reproducing the environment where the embedded system will run on.

- Software in Loop (SIL): In simulation tests for embedded system, Software in loop testing comes before HIL, where you test the software code/application and not the entire system. This is done by simulating the software environment.

- ECU emulation: This involves test benches that partially or fully integrate the electrical systems of the vehicle.

- Vehicle tests: This is the testing performed in the real environment (i.e. in the trucks).

Figure 4.2: ISO 26262 specific Enumerations

Abstraction: Packages and Frames

Packages were used to represent the main categories and the frames for the abstractions respectively. The main categories were defined as parent classes in the UML class diagrams. Here, the three main categories defined for software testing and verification were:

Software Verification Plan (SVP)

Software Verification Specification (SVS)
Chapter 4. Results

and Software Verification Report (SVR)

All software testing and verification based WP types with their attributes and relationships have been grouped together into one of these main categories. Thus, these main categories are the parent classes. Finally, the abstraction takes place where all these main categories, within the context of the software testing and verification are framed under ‘ISO 26262 Quality Management’. Similarly the ADS WP type was abstracted as ‘ISO 26262 Architecture Management (AM)’ and the SSR WP type was abstracted as ‘ISO 26262 Requirement Management (RM)’.

ISO 26262 QM, QM domain in OSLC has testing related resource types [79]. In the class diagrams, the packages SVS, SVP and SVR consist of ISO 26262 based software testing WP types. Thus, they were abstracted as ISO 26262 QM as a proposed extension to OSLC QM resource types. It was depicted using a UML frame element as they are all WP types within the context of quality management phase of ALM.

ISO 26262 AM, AM domain in OSLC has architecture or design related resource types [80]. In the class diagrams, the category, architecture design specification is an ISO 26262 specific WP type. Thus, it was abstracted as ISO 26262 AM as a proposed extension to OSLC AM domain resource types. It was depicted using a frame as ADS WP type falls within the context of AM phase of ALM.

ISO 26262 RM, RM domain in OSLC has requirements related resource types [81]. In the class diagrams, the category, SSR is an ISO 26262 specific WP type. Thus, it was abstracted as ISO 26262 RM as an proposed extension to OSLC RM domain resource types. It was depicted using a frame as SSR WP type falls within the context of RM phase of ALM.

The Relationships Types in the Conceptual Meta Model
The name of the connector code used for grouping of the categories and for depicting the traceability links between the categories is presented in the Table 4.2. The connector codes are depicted using either the dependency relationship type for traceability link, or the aggregation and composition relationship type for grouping of categories in the UML class diagram.

Table 4.2: Relationships in the Conceptual Meta Model

<table>
<thead>
<tr>
<th>Category 1</th>
<th>Connector Code (i.e the name of the relationship link)</th>
<th>Category 2</th>
<th>Relationship Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software verification specification</td>
<td>is based on</td>
<td>Software verification plan</td>
<td>Traceability link</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------</td>
<td>---------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Software verification report</td>
<td>is generated based on</td>
<td>Software verification specification</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Software verification report</td>
<td>is generated based on</td>
<td>Software verification plan</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Software verification plan</td>
<td>is for verifying against</td>
<td>Architecture design specifications</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Software verification plan</td>
<td>is for verifying against</td>
<td>Software safety requirements</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Software verification specification</td>
<td>is for verifying based on</td>
<td>Architecture design specifications</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Software verification specification</td>
<td>is for verifying based on</td>
<td>Software safety requirements</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Test object</td>
<td>must fulfill</td>
<td>Architecture design specifications</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Test case</td>
<td>is defined for</td>
<td>Test object</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Structural metrics</td>
<td>used to verify</td>
<td>Test cases</td>
<td>Traceability link</td>
</tr>
<tr>
<td>Software verification specification</td>
<td>has</td>
<td>Test case, test method and test object</td>
<td>Grouping of categories</td>
</tr>
<tr>
<td>Software verification plan</td>
<td>has</td>
<td>Test environment, structural coverage, test method, test object, tool</td>
<td>Grouping of categories</td>
</tr>
<tr>
<td>Software verification report</td>
<td>has</td>
<td>Tool, test environment, calibration data</td>
<td>Grouping of categories</td>
</tr>
<tr>
<td>Test case</td>
<td>has</td>
<td>test environment, configuration data</td>
<td>Grouping of categories</td>
</tr>
<tr>
<td>Test case</td>
<td>is composed of</td>
<td>test data</td>
<td>Grouping of categories</td>
</tr>
</tbody>
</table>

4.2 Summary of Results for Second Data Collection Cycle

Summary of the Interviewees
7 interviewees involved across the three software test levels for the M1 ECU system have been invited. Of the total of 9 interviews conducted with the 7 interviewees, 7 have been audio recorded. Field notes were taken for the other 2 interviews,
which were immediately transcribed. The general information pertaining to the background of the interviewees and the characteristics of the interviews is presented in the Table 4.3.

As seen in Table 4.3, some subjects were involved across more than one software test level. There were 4 interviewees for the software unit testing, 4 for software integration testing and 2 for verification of SSR testing. The interviewees had different levels of experience in working with embedded software systems. The experience range helps to encompass a wider understanding on the WPs being produced for software testing and verification of M1 ECU system. The actual test execution was performed by one or two subject at each of the three test levels. Yet, there were other individuals involved in producing WPs that are used or required for testing as depicted in the conceptual meta model. Thus, all individuals involved have been consulted and interviewed. The involvement of a subject with a particular WP type, was used as the basis for deciding the open ended question to be asked.

Table 4.3: General Information of the Interviews

<table>
<thead>
<tr>
<th>Interviewee</th>
<th>Job Role</th>
<th>Experience (years)</th>
<th>Software Test Level</th>
<th>Number of round of interviews</th>
<th>Time Spent (approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I 1</td>
<td>Software developer and tester</td>
<td>&gt; 1</td>
<td>Software Unit testing, Software integration testing</td>
<td>2</td>
<td>75 mins</td>
</tr>
<tr>
<td>I 2</td>
<td>Test Engineer</td>
<td>3</td>
<td>SSR verification testing</td>
<td>2</td>
<td>60 mins</td>
</tr>
<tr>
<td>I 3</td>
<td>Development Engineer</td>
<td>8</td>
<td>Software Integration testing</td>
<td>1</td>
<td>40 mins</td>
</tr>
<tr>
<td>I 4</td>
<td>Development Engineer</td>
<td>3</td>
<td>Software Unit testing</td>
<td>1</td>
<td>40 mins</td>
</tr>
<tr>
<td>I 5</td>
<td>Development Engineer</td>
<td>3</td>
<td>Software Unit testing</td>
<td>1</td>
<td>15 mins</td>
</tr>
</tbody>
</table>
4.3 Summary of Results for Second Data Analysis Cycle

The results of the data coding for the structured analysis matrix is presented in this section. In order to extract data for filling the structured analysis matrix, the steps presented below have been followed:

- Understanding the WP types which were produced for M1 ECU software testing. This was done by asking closed ended questions (see Appendix B) to the interviewees, with answers as Yes (Y), No (N) or Not Sure (NS). These questions were asked to all interviewees. Figure 4.3 depicts results of this step.

Results of closed ended questions helped to determine the WPs that were produced for the M1 ECU at the three software test levels. If all the interviewees at each test level answered with an unanimous Yes or No, it was concluded that the category (i.e WP type) exists or not, respectively. In case, there was a non unanimous response from all the interviewees for a particular test level, it has been concluded as Not Sure for that test level. In such a case, the answers to the open ended question were used for clarifying and determining whether the WPs were produced or not.

The Not Sure response could either be due of lack of knowledge or due to misinterpretation of the questions asked. To reduce the risk of misinterpretation, the meaning of the category for which the question was formulated, was explained (as described in Section 4.1.2), prior to asking of the question. Certain cells of the table depicted in Figure 4.3 were left blank, as the categories were not relevant to the test level under consideration (for ex: structural coverage is not relevant or applicable for SSR verification test level). This was because the meta model categories are a compilation of all
the WP types required by clauses 9, 10 and 11. Thus some WP types may be required at a certain test level and some WP types may not be required.

- Based on the results of the previous step, open ended questions (see Appendix C) pertaining to the WP types produced for M1 ECU were asked. The responses were used for extracting data for attributes that define the WPs. Open ended questions were asked right after the corresponding closed ended question (for ex: *Do you have a plan for testing modules for M1 ECU system?* was followed by *What does your plan consist of?*). Further, the documents for the work products produced were also used for extracting data codes apart from being a proof of evidence. The data codes for the attributes and classes of the meta model for M1 ECU software testing and verification is depicted in the Table 4.4.

<table>
<thead>
<tr>
<th>Meta Model Concepts</th>
<th>6-9: Software Unit Testing Number of interviewees</th>
<th>6-10: Software Integration Testing Number of interviewees</th>
<th>6-11: Verification of Safety Requirements Number of interviewees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categories</td>
<td>Y N NS Result</td>
<td>Y N NS Result</td>
<td>Y N NS Result</td>
</tr>
<tr>
<td>Software Verification Plan (Do you have a plan for testing the software modules/integrated modules/verifying the requirements for CMS ECU system?)</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Tool</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Test Environment</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Test Object</td>
<td>4 0 0 No</td>
<td>4 0 0 No</td>
<td>- - -</td>
</tr>
<tr>
<td>Structural Coverage</td>
<td>4 0 0 No</td>
<td>4 0 0 No</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Test Case</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Test Data</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Test Method</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Configuration Data</td>
<td>4 0 0 No</td>
<td>4 0 0 No</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Calibration Data</td>
<td>0 3 1 NS</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Architecture design Specification</td>
<td>4 0 0 Y</td>
<td>2 0 2 NS</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Software Safety Requirements (Assumed software specific AER)</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Software verification specification</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
<tr>
<td>Software Verification Report</td>
<td>4 0 0 Y</td>
<td>4 0 0 Y</td>
<td>2 0 0 Y</td>
</tr>
</tbody>
</table>

Figure 4.3: WP available for M1 ECU System: Results of Closed Ended Questions

The extraction of these data codes from the interview transcripts and WP documents is explained in the data analysis section.

### 4.3.1 Inductively Defined Extension to the Conceptual Meta Model

From the interview transcripts and documents, new data that do not fit into codes and categories of the matrix, relevant to the objective of the research have been identified. Thus data that defines a type of WP or its attributes which can be used for showing safety evidence have been inductively analysed. The process
### 4.4: Data coding according Structured Analysis Matrix - Work Products and their Attributes

<table>
<thead>
<tr>
<th>Conceptual Meta Model Concepts</th>
<th>Testing</th>
<th>Integration Testing</th>
<th>Software Safety Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Verification Plan (SVP)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Not tailoring</td>
<td>Not tailoring</td>
<td>Not tailoring</td>
</tr>
<tr>
<td>Test Environment</td>
<td>Not tailoring</td>
<td>Not tailoring</td>
<td>Not tailoring</td>
</tr>
<tr>
<td>Test Object</td>
<td>Not tailoring</td>
<td>Not tailoring</td>
<td>Not tailoring</td>
</tr>
<tr>
<td>Test Case</td>
<td>Not tailoring</td>
<td>Not tailoring</td>
<td>Not tailoring</td>
</tr>
<tr>
<td>Test Method</td>
<td>Not Tailoring</td>
<td>Not Tailoring</td>
<td>Not Tailoring</td>
</tr>
<tr>
<td>Test Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calibration Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Verification Report (SVP)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Design specifications (AD)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety Requirement specification</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4. Results

of inductive content analysis process explained in Section 3.2.2, was used to define open codes, categories, main categories and abstractions for newly identified data.

The newly defined open codes are: requirement coverage, tested requirements, not testable requirements, percentage, test time, in house developed, test execution record and total requirements.

They have been grouped together into categories as:

- In house developed was grouped with other codes for the category, Tool.
- Title, identifier, requirement coverage, tested requirements, not testable requirements, percentage and total requirements were grouped together into the category, requirement coverage.
- Test time, test execution record were grouped together into a category, Test execution record.

The newly defined connector codes were: SVR ‘includes’ the requirement coverage, Tool ‘generates’ Test execution record.

Categorization and abstraction:

- The category, requirement coverage was further grouped with other categories into the main category of SVR using the aggregation relationship type in the conceptual meta model. The requirement coverage is also a type of test report and thus is included as part of the main category, SVR.
- The category, test execution record is traced to the tool that generates it by the dependency relationship type in the conceptual meta model. Since the test execution record is a WP type that is the result of testing, it is grouped as part of the main category, SVR.

The description of the additional new open codes and categories defined in the extended and refined conceptual meta model were:

Requirement coverage: This is the report generated as the result of verifying the software specific AERs against the embedded software of M1 ECU system. This differs from the parent class, SVR, in terms of the attributes (i.e codes) that are grouped to define it. Since this a type of verification report, it is included as part of the main category, SVR.

Non testable requirements: These are the software specific AERs which cannot be verified for being realized by the embedded software in M1 ECU. This was described as part of the report for requirement coverage.

Tested requirements: These are the software specific AERs that are verified to be realized in the embedded software of M1 ECU system. This was specified as part of the report for requirement coverage.
Total requirements: These are the total number of software specific AERs assigned to AE1 and AE2, which are realised by the M1 ECU software. This was described as part of the report for requirement coverage.

Percentage: This is the percentage of the number of software specific AERs tested to the total number of software specific AERs defined to be realised by the embedded software in M1 ECU. This was specified as part of the report for requirement coverage.

In house developed: This is to indicate if the tool is bought commercially or manufactured in house. This attribute is of importance for the tool developer, to decide if they have to perform tool qualification based on the ISO 26262 requirements. Such results of tool qualification can be used to show safety evidence. Thus, it is required to know if the tool is built in house or not in order to decide if tool qualification requirements are to be applied.

Test execution Record: This is the record generated by the testing tool during the execution of the test.

Test time: This is the amount of time the tool or the tool framework takes to execute the test in the given test environment, for the given test object. This is of importance for showing safety as the tests must be executed within a given time in the simulated test environment to ensure their safe functioning.

Thus, the conceptual meta model defined in the first cycle is extended and tested. This testing was for the usability of the WP types defined in the conceptual meta model. The concepts in the meta model were instantiated with WPs to be defined in RDF. Hence, it was tested that the meta model WP types were defined in the RDF structure (i.e as resource types) for extending OSLC domains. From the testing of usability, it was seen that the WP types and their attributes in the conceptual meta model could be largely populated for software testing and verification of an ECU system like M1 at the company. Further, such an empirical investigation has lead to refinements and extensions to the concepts in the meta model with respect an ECU system at the company. The population of the structured analysis matrix with the data for M1 ECU system helps to define traceability that must exist between the WPs for compliance to ISO 26262.

4.3.2 OSLC Resources: Resource Description Framework (RDF)

The purpose of this section is to illustrate the way in which the WPs defined for software testing and verification of M1 ECU system were translated into OSLC resources (i.e RDF).

The data codes in the structured analysis matrix, which are the data for the classes in the conceptual meta model were used for defining the RDF. RDF, is used to represent data in the structure of subject, predicate and object. This is
called the RDF triple. A grouping of such RDF triples is called an RDF graph [43]. A graph is a set of triples with a unique namespace (Universal resource Locator). One simple way to generate the graph is to map it to the model[18]. This model in the current context were the instantiated classes of the meta model.

- the subject, which is the WP type (i.e. name of the class)
- the predicate, which is either a property, resource or relationship property (i.e. attribute or another WP type or traceability link)
- the object, which is the value of the property or relationship property (i.e. either an attribute value or another WP)

The subject, predicate and object when implemented, were referred using an Universal Resource Identifier (URI). Since, there was no implementation, no such URIs exist. Thus dummy URIs of the form “http://open-services/ISO 26262-extension-name” for defining the domain extension of OSLC for ISO 26262 and, “http://myserver/Work Product Type/Identifier” for identifying the resource have been used as depicted in Figure 4.5.

Consider the data codes for the instantiated parent class of SVP at the software unit test level. This parent class has its attributes and the child classes included in it. Thus, the subject was SVP and the predicates were the attributes of SVP, the WP types (i.e. child classes) grouped to define SVP, and the traceability links defined for SVP. Thus, the attributes were title, pass criteria, regression strategy, exception handling action and test level. The WP types were tool, test environment, test object and the traceability links were between SVP and ADS (since we consider software unit testing level, no link with SSR). The objects were the values for all above mentioned predicates. Look at Figure 4.4 for this depiction.

Figure 4.4: SVP Class Instantiation for M1 ECU system: Software unit test level
Chapter 4. Results

Figure 4.5: Corresponding RDF graph for SVP Work Product

As seen in the Figure 4.5, the WPs that are predicates for the subject, SVP were referenced using “http://myserver/WorkProductType/Identifier”. This is the URI that consists of the attribute values for the particular WP. The above mentioned URI is the subject of another RDF graph that describes the predicates and objects of the subject under consideration. Also it can be noted that the attribute, identifier, is for the objects of the classes in the conceptual meta model. The identifier is instantiated when an URI is actually created.

A similar process was followed for the other instantiated classes in the conceptual meta model to define their RDF graphs. The syntax for the RDF graphs for the three main WPs of SVP, SVR and SVS at the three software test levels is presented in Appendix G. Similar RDF graphs can be defined for all other instantiated WPs for the data codes presented in the Section 5.2.1.

4.3.3 Refined and Tested Conceptual Meta Model

The refined, extended and tested conceptual meta model, based on the empirical investigation with respect to software testing and verification of M1 ECU system is depicted in Figure 4.6.
Figure 4.6: Refined and Tested Conceptual Meta Model
4.4 Validation

The validation of the extended and refined conceptual meta model is presented in this section. Since a questionnaire was used for validation, the use of face-to-face interviews or electronic mail have been contemplated as modes of data collection [82]. The validation requires particular contexts defined in the conceptual meta model to be focused on for each of the questions in the questionnaire. For this purpose claims for each question based on the conceptual meta model have been made to make the questionnaire as self contained as possible. Such claims aid the respondent in understanding the context of the question. It ensures minimal interaction with the respondent. A mock trial of data collection using electronic mail was performed by mailing the questionnaire and conceptual meta model to one respondent. From the outcome it was identified that creator of such a model could be required in case of any ambiguity or confusion. Thus, face to face interviews were chosen as the means of data collection for the validation. Also, face to face interviews reduce the cognitive burden on the respondent along with ensuring higher response rates [83].

4.4.1 Summary of Interviewees

The data has been collected from 10 interviewees. A short questionnaire was used to collect the background information of the interviewees (See Appendix D). The anonymity of all the interviewees was preserved. Interviewee experience with respect to embedded systems/EU systems and their expertise with respect to the ISO 26262 standard have been taken into consideration. Judgement Sampling, a non-probabilistic sampling method has been chosen for selecting the most productive sample of respondents for the validation [60]. The next step in the validation approach was to collect empirical responses for the questions. The questions were formulated based on the criteria defined in Section 3.4.1. The questionnaire was answered by individuals with varied knowledge level in ISO 26262, including experts. This varied experience and expertise level of the subjects helped to adjudge the effectiveness of the conceptual meta model by encompassing for a broader viewpoint.

4.4.2 Results of Validation

Likert scale [84] was used for preparing the questions for the validation questionnaire (See Appendix D). It is an ordinal scale with ordered ranks assigned to each response. Based on the response chosen by the interviewees the corresponding rank was used for quantifying the responses [85]. The purpose of the validation was to determine the effectiveness of the model in representing the knowledge on the right hand side of the ISO 26262 software V-model. Thus, the quantified ordinal data was statistically analysed using measures for determining
Chapter 4. Results

the central tendency in the data. There are three such measures of central tendency, named, median, mean and mode. The arithmetical manipulations required to calculate the mean are inappropriate for ordinal data [86]. Thus median and mode were chosen to determine a single value that describes the effectiveness of each criteria being validated.

Most of the interviewees have rated that model satisfies the criteria of abstraction, confirmability and traceability to a great extent. Thus, from the histogram with the results for the validation (See Appendix D), the effectiveness of the criteria, confirm ability in the conceptual meta model has the mode 4 and median 4. The effectiveness of the criteria, traceability in the conceptual meta model has the mode 4 and median 4. The effectiveness of the criteria, abstraction in the conceptual meta model has the mode 4 and median 4.

Table 4.5: Interviewee Profile for Validation

<table>
<thead>
<tr>
<th>Interviewee (I)</th>
<th>Experience in Years (in testing or with safety critical systems)</th>
<th>Job Role</th>
<th>ISO 26262 Expertise Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>&lt; 1 year</td>
<td>Student</td>
<td>Beginner</td>
</tr>
<tr>
<td>I2</td>
<td>17 years</td>
<td>Test engineer</td>
<td>Advanced</td>
</tr>
<tr>
<td>I3</td>
<td>1 year</td>
<td>Software developer and tester</td>
<td>Fundamentally aware</td>
</tr>
<tr>
<td>I4</td>
<td>15 years</td>
<td>Safety</td>
<td>Beginner</td>
</tr>
<tr>
<td>I5</td>
<td>15 years</td>
<td>Embedded systems development and ISO 26262 researcher</td>
<td>Expert</td>
</tr>
<tr>
<td>I6</td>
<td>3 years</td>
<td>Integration tester</td>
<td>Intermediate</td>
</tr>
<tr>
<td>I7</td>
<td>3 years</td>
<td>Test Engineer</td>
<td>Fundamentally Aware</td>
</tr>
<tr>
<td>I8</td>
<td>3 years</td>
<td>Software Tester</td>
<td>Intermediate</td>
</tr>
<tr>
<td>I9</td>
<td>20 years</td>
<td>Senior System Architect</td>
<td>Intermediate</td>
</tr>
<tr>
<td>I10</td>
<td>5 years</td>
<td>Test Developer</td>
<td>Advanced</td>
</tr>
</tbody>
</table>
Chapter 5

Data Analysis

5.1 Data Analysis for the First Cycle

The process of inductive content analysis in the first cycle is presented in this section. The data pertaining software testing and verification in ISO 26262 standard and company presentation on the standard were the two data sources used as units of analysis in the first cycle. Each Clause in the ISO 26262 standard has sub clauses for objectives, general, inputs for the clause, requirements and recommendation and WPs. Thus, there is already a sub clause for WPs. The WPs required by this sub clause in the unit of analysis are: SVS, SVP and SVR. The same WP types are required for all the clauses in the unit of analysis. These WP types must be produced and refined through the Clauses of the unit of analysis (i.e SVP defined in Clause 9 is used and refined for defining the SVP in Clause 10). The attributes and other WP types that are used to describe these three WP types were defined by analysing the requirements and recommendations sub clause, in the units of analysis. Further, the WP types against which testing or verification must be performed according to ISO 26262 was stated in the sub clause, objectives. The objective sub clause needs to be analysed to understand which WPs must be linked for tools to interoperate and exchange data. Thus, apart from understanding the data exchange within the software testing and verification WPs, the data exchange with WPs in other phases of ALM also needs to be identified for showing interoperability across the ALM.

Hence, the sub clauses of objectives, requirements and recommendation, and WPs in the unit of analysis of the first cycle were analysed for extracting data codes. Since the standard is not an open source document, the process of inductive content analysis has been depicted for one main WP type, SVP. The similar process was followed for the other two WP types of SVS and SVR.

note: all the sentences under quotation have been paraphrased from the ISO 26262 standard.
5.1.1 Clause 9: Software Unit Testing

- The sub clause 9.1 states the objective of software unit testing as “To demonstrate that the software units fulfil the software unit design specifications and do not contain undesired functionality.”

- Since we are considering the work products, the sub clause 9.5 for WPs was analysed. The part of the sub clause 9.5 pertaining to SVP states that “Software verification plan must result from requirements 9.4.2 and 9.4.6”

- 9.4.2 states that “Software unit testing shall be planned, specified and executed according to ISO 26262 Part 8, Clause 9.”

Clause 9 in Part 8 of ISO 26262 standard has the same clause structure as the other clauses in the ISO 26262 standard. Hence, the requirements and recommendations sub clause of 9.4 is analysed. Sub clause 9.4 states what verification and testing must consist of.

Sub clause 9.4 is once again divided into 9.4.1, 9.4.2 and 9.4.3 for verification planning, verification specification and verification execution and evaluation respectively.

9.4.1, which is for verification planning is further split into 9.4.1.1 and 9.4.1.2. The requirements on the way the planning should be done are specified in 9.4.1.2. While, the requirements on what the plan should consist of are specified in 9.4.1.1. Since the focus is only on defining what the work products are, the 9.4.1.1 was analysed. This sub clause states that “The verification plan should be carried out for each phase and sub-phase of the safety life cycle and must address and consist of the following: a) the content of the WPs to be verified, b) methods used for verification, c) the pass and fail criteria for the verification, d) the verification environment, if applicable, e) the tools used for verification and its application guidelines, if applicable, f) the actions to be taken if anomalies are detected, and g) the regression strategy.”

- The SVP must result also from the requirements in 9.4.6, which states that “The test environment for software unit testing shall correspond as closely as possible to the target environment. If the software unit testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analysed in order to specify additional tests in the target environment during the subsequent test phases. The software unit testing can be executed in different test environments like, Software in loop, hardware in loop, ECU emulator tests”.

- From the slides in the presentation [87] on the Clause 9 state that: “The test planning must consist of the object to be tested, the WPs to be verified, the pass and fail criteria, the environment for testing, the tool used, the
The text tagged for SVP at software unit testing level were: object to be tested, WP to be verified, tools used for verification, guidelines for using the tools, actions for anomalies detection, regression strategy, pass or fail criteria, methods used for verification, object code to be verified for software unit testing, target environment for software units, software unit, software unit design specification, software unit fulfils software unit design specification, software verification plan ‘shall have’ all the text tagged so far, software verification plan ‘resulting from’ all the text tagged so far.

5.1.2 Clause 10: Software Integration and Testing

- The sub-clause 10.1 in clause 10 states that “The objective of this sub-phase is to demonstrate that the software architectural design is realized by the embedded software.”

- Since we are considering the work products, the sub clause 10.5 for WPs was analysed. The part of the 10.5 pertaining to SVP states that “Software verification plan from sub clause 9.5 must be refined using results from requirements 10.4.1, 10.4.6 and 10.4.8”

- 10.4.1 states that “The planning of the software integration shall describe the steps for integrating the individual software units hierarchically ...”. Thus this sub clause is pertaining to describing the steps of the process of defining SVP. Since the focus is on product based safety evidence, this requirements is not analysed for tagging.

- 10.4.6 states that “This subclause applies to ASIL A, B, C and D for the associated safety goal, which is used to define the safety requirements and implementation fulfilling these safety requirements. To evaluate the completeness of test cases and to obtain confidence that there is no unintended functionality, the structural coverage shall be measured in accordance with the metrics listed. These recommended structural coverage metrics based on the ASIL for software integration testing are call coverage, function coverage, statement coverage, branch coverage, modified condition/decision coverage (MC/DC). If the achieved structural coverage is considered insufficient, either additional test cases shall be specified or a rationale shall be provided.”.
• 10.4.8 states that “The test environment for software integration testing shall correspond as closely as possible to the target environment. If the software integration testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment shall be analysed in order to specify additional tests in the target environment. The software integration testing can be executed in different test environments based on the ASIL such as software in loop, hardware in loop, ECU emulator tests”.

• The slides in the presentation [87] on the Clause 10 state that “test object is the software component which in the current context is the embedded software. The test environment is based on ASIL recommended target environment types. The goal is to show compliance to the software architecture design. The methods used are for testing, for deriving test cases and for the structural coverage metrics.”

The text tagged for SVP at software integration testing level were: Associated ASIL for implementation being tested (i.e software component, software unit or embedded software), call coverage, function coverage , ASIL recommended methods for structural coverage, methods for structural coverage metrics, test cases, structural coverage ‘to evaluate for’ test cases, target environment, object code, test environment, embedded software ‘realises’ software architecture design, methods for target environment, ASIL recommended methods for target environment, Software in loop, hardware in loop, ECU emulator tests, software verification plan ‘shall have’ all the tagged text, software verification plan ‘resulting from’ all the tagged text.

5.1.3 Clause 11: Verification of Software Safety Requirements Testing

• The sub clause 11.1 states that “The objective of this sub-phase is to demonstrate that the embedded software fulfils the SSRs”

• Since we are considering the work products, the sub clause 11.5 was analysed. The part of the 11.5 pertaining to SVP states that “Software verification plan from sub clause 10.5 must be refined using results from requirements 11.4.1 and 11.4.3”

• 11.4.1 states that “The verification of the SSRs shall be planned, specified and executed in accordance with ISO 26262:2011, Clause 9.” This clause is already analysed and is the same as in the requirement 9.4.2.

• 11.4.3 states that “The testing of the implementation of the SSRs shall be executed on the target environment. There are recommended methods for the
Chapter 5. Data Analysis

choice of target environment based on ASIL such as software in loop, hardware in loop, vehicle tests and ECU emulator tests”.

- The slides in the presentation [87] on the Clause 11 state that “Test object is the embedded software. The test environment is based on ASIL recommended target environment types largely being the ECU emulator or the vehicle, the goal is to show compliance to the SSRs”.

The text tagged for SVP at software safety requirement verification test level is: software safety requirements, target environment, embedded software, WP to be verified, tools used for verification, actions for anomalies detection, regression strategy, pass or fail criteria, methods used for verification, embedded software ‘fulfills’ software safety requirements, ASIL recommended method for target environment, software in loop, hardware in loop, vehicle tests, ECU emulator tests, software verification plan ‘shall have’ all the tagged text, software verification plan ‘resulting from’ all the tagged text.

Explanation

The process of coding from the text tagged for SVP in the Clauses 9, 10 and 11 is explained here.

Tagged text having the same meaning or repeated frequently in different contexts are placed in the rows of the Table 5.1. The constant comparison of such tagged text is used for deriving the codes. Constant comparison [72] is where text is compared based on the aim to define patterns in the text, that can be extracted as codes. These codes can then constantly be compared to be merged, divided or modified. For example, if we consider in row 1, object to be tested, implementation to be tested, embedded software, software units and object code to be verified all have the same meaning of a test object and thus the code test object has been extracted. The tagged text, software unit testing, integration testing, software safety requirement verification tests are repeated in the different contexts of a test object, structural coverage, test environment and has thus been extracted as the code ‘test level’ to define the three software test levels.

Thus, constant comparison was done to check if text was frequently tagged in different contexts or if text having the same or similar was tagged, in order to extract codes. In Table 5.2, all text that describes the way in which codes are linked were extracted as a connector codes [64]. Then, using the inductive content analysis process explained in Section 3.2.2, the classes are defined.
<table>
<thead>
<tr>
<th>Tagged Text</th>
<th>Open codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object to be tested, associated ASIL for implementation being tested, Object code to be verified for software unit testing, WP to be verified, Object code to be verified for software integration testing, Software units, Embedded software</td>
<td>Test object, ASIL, Test level</td>
</tr>
<tr>
<td>Verification environment, Test environment for software integration, Test environment for software units, Target environment for software units, Target environment for software integration, Target environment for SSRs verification, Software in loop, Hardware in loop, ECU emulator, Vehicle tests</td>
<td>Test environment, Method names for target environment, Test level, SSR</td>
</tr>
<tr>
<td>Pass or fail criteria, Pass or fail criteria for verification</td>
<td>Pass Criteria</td>
</tr>
<tr>
<td>Tools used for verification, Guidelines for using tools</td>
<td>Tool, Guidelines used</td>
</tr>
<tr>
<td>Regression strategy</td>
<td>Regression strategy</td>
</tr>
<tr>
<td>methods used for verification, methods used for testing</td>
<td>Test Method</td>
</tr>
<tr>
<td>Plan for software integration, Plan for software unit testing, Plan for verifying SSRs</td>
<td>Software verification plan, Test level, SSR</td>
</tr>
<tr>
<td>ASIL Recommended methods for structural coverage, methods for structural coverage metrics, ASIL Recommended methods for target environment, based on Automotive Safety Integrity Level</td>
<td>ASIL, methods for structural coverage metrics, methods for target environment</td>
</tr>
<tr>
<td>Actions for anomalies detection</td>
<td>Exception handling action</td>
</tr>
<tr>
<td>Structural coverage for software unit test cases, structural coverage metrics for unit test cases, Structural coverage for software component test cases, Structural coverage metrics for integration test cases, Call coverage, Function coverage</td>
<td>Structural coverage, Structural coverage metrics, test level, Test case</td>
</tr>
<tr>
<td>Test cases for software integrated components, Test case for software units</td>
<td>Test case, Test level</td>
</tr>
<tr>
<td>WP to be verified at software unit level, Software unit design specification, Software architecture design</td>
<td>Architecture design specification</td>
</tr>
</tbody>
</table>

Table 5.1: Open Codes- Software Verification Plan
Chapter 5. Data Analysis

<table>
<thead>
<tr>
<th>Tagged Text</th>
<th>Open Connector codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>structural coverage ‘to evaluate for’ test cases</td>
<td>Structural coverage ‘Is used for verifying’ Test Cases</td>
</tr>
<tr>
<td>Software units ‘fulfills’ software unit design specification, Integrated software ‘fulfills’ software architecture design, Embedded software ‘fulfills’ software safety requirements</td>
<td>Test object ‘must fulfill’ Architectural design specification</td>
</tr>
<tr>
<td>Verification plan for the software units ‘realize’ software unit design specification, Verification plan for the embedded software ‘realize’ software safety requirements, Verification plan for the integrated software must ‘realize’, software architecture design</td>
<td>Verification plan ‘is for verifying’ Architecture design specification, Verification plan ‘is for verifying’ Software Safety Requirements</td>
</tr>
<tr>
<td>software verification plan ‘shall have’ all the identified tagged text, software verification plan ‘resulting from’ all the identified tagged text</td>
<td>Software verification plan ‘has’ all codes defined in the Table 5.1</td>
</tr>
</tbody>
</table>

Table 5.2: Open Connector Codes- Software Verification Plan

The codes that have been defined, depict either a type of WP, their attributes, or the traceability link between them. Next, these codes were grouped and categorised. They were grouped together to describe the classes and parent classes. Parts of the conceptual model depict an aspect of the model for better understanding purpose. Thus, here the part depicting the parent classes of SVP, SVS and SVR in the conceptual meta model are explained.

5.1.4 Software Verification Plan in Conceptual Meta model

The codes regression strategy, pass criteria, exception handling action, test level have been extracted for describing the SVP (see Figure 5.1). These codes do not have other codes that can be grouped to define them as classes. Thus, these codes are the attributes of the class, SVP. But, the classes, tool, test object, test method, test environment and structural coverage had other codes that could be grouped to define them as classes. Thus, these are the child classes that were also grouped to describe SVP. The attributes along with the child classes define the main category, SVP. The SVP is the parent class, which consists of child classes related through the aggregation relationship as depicted in Figure 5.1.
Additionally, codes ‘title’ and ‘identifier’ required for describing OSLC resource types were also grouped for defining the classes. The class, SVP has traceability links with class, ADS and class, SSR as depicted using the dependency relationship type. The traceability link specifies that SVP is used for verifying against the ADS or SSR depending on the software test level under consideration. Also, the class, Test object has a traceability link with ADS, which it must fulfill to meet safety specifications in the ADS.

In a similar way, tagged text have been coded, grouped and categorised for the other two WP types of SVS and SVR.

### 5.1.5 Software Verification Specification in Conceptual Meta model

The Figure 5.2 depicts SVS as defined by the codes of test level, title and identifier. These are the attributes of SVS. SVS is the main category resulting from the grouping of the classes, test object, test case and test method using the aggregation relationship type (i.e using connector code ‘has’). The test case is another main category, which groups the classes, test environment, configuration data and test data using the connector codes ‘has’ (i.e aggregation relationship) and ‘is composed of’ (i.e composition relationship) respectively. But, the test case was again grouped with other classes to define SVS. Thus, the final main category defined here was SVS.
Figure 5.2: Categorization for Main Category- Software Verification Specification

SVS had traceability links with ADS class, with SSR class and with the class, SVP. The class, test case has traceability link with structural coverage which is grouped to be a child class of the main category, SVP. Also, the class, Test object has a traceability link with ADS, which it must fulfill to meet safety specifications in the ADS. In this way the links depict the way WP types can be traced to define a flow among them.

5.1.6 Software Verification Report in Conceptual Meta model

Figure 5.3: Categorization for Main Category- Software Verification Report

The Figure 5.3 depicts the main category, SVP defined by the codes of tailoring, pass result, test level, version, title and identifier. It also depicts that SVR
as the main category/parent class resulting from the grouping of classes, tool, test environment and calibration data using the aggregation relationship type (i.e. using connector code ‘has’). Further, the SVR has traceability links with other two main categories of SVP and SVS.

After defining the main categories, abstraction was performed. Abstraction was for depicting the defined classes to be more relevant for OSLC domains. Here, ADS, SSR and software testing and verification based WP types have been abstracted using frames in the UML class diagrams. These frames correspond to AM, RM and QM domains of OSLC respectively. Since, ISO 26262 specific resource types have been defined as extensions to these OSLC domains, the extensions were named as ISO 26262 QM, ISO 26262 RM and ISO 26262 AM.

The process of coding, grouping, categorising and abstracting into common higher order headings (i.e classes and parent classes) in inductive content analysis helps to resolve the complexity involved in defining conceptual models. Thus, links for traceability could be efficiently depicted in a minimalistic way, by linking the main categories/parent classes that compose the child classes. Finally, all the three main categories of SVP, SVR and SVS have been combined and represented using UML packages in the conceptual meta model. The ISO 26262 specific OSLC domain abstractions have been represented using the UML frames in the conceptual meta model as presented in Figure 4.1.

5.2 Data Analysis for the Second Cycle

The process of deductive content analysis for extracting codes from the interview transcripts and WP documents is presented in this section. The documents used were the technical data reports of the WPs. A total 8 technical data reports related to software testing and verification of the M1 ECU available have been used. The data from these documents was triangulated with interview transcripts for extracting the data codes.

5.2.1 Data Codes for the Structured Analysis matrix

The data codes for the WP types instantiated with respect to M1 ECU system at the three software test levels is presented in this section. The data for these WP type attributes (i.e codes) can be used to identify, define and link the WPs based on the traces defined in the conceptual meta model. This data is used to define the RDF to represent WPs as OSLC resources. The RDF defined for the data codes of SVP, SVS and SVR WPs at the three test levels is presented in Appendix G.
5.2.1.1 Data Codes for Software Unit Testing

The answers to the open ended questions were transcribed and the relevant content from the transcriptions and documents for the WPs have been paraphrased in the quotations. This paraphrased data was used as the basis to derive the data codes defined below.

**Class: Test Object**
**Title:** Simulink models for M1: Fuel Application Software, **Asil:** Not available, **Version:** Daily updated versions

The smallest software part tested in M1 ECU was identified as Fuel application software. This is the software unit under test. From the related interview transcript, “We create simulink models for the fuel application software, which is the smallest part of the ECU that we test”. From another interviewee, “I would say a module is equivalent to the application software, M1: Fuel. It is the only application software in M1 used for AE1 and AE2 of FLDS.”. ISO 26262 standard also has notes for interpreting the standard for model based development, where the test object can be a model. Currently, ASIL classification is not performed for the requirements and thus the subsequent test object that realises these requirements also does not have an ASIL assigned. An interviewee replied “No, we don’t have any ASIL levels assigned”. From the technical data report [88], it was confirmed that no ASIL have been assigned. The report was also useful for gathering data codes for the version of the fuel application software considered.

**Class: Tool**
**Title:** Test Engine tool, **Guidelines used:** None, **In House Developed:** Yes

Testing or verification is largely performed using tools and tool frameworks. It has been identified that Test engine was used for testing the simulink models of Fuel application software. The transcripts from the interviews were “the simulink models produced are run in the test engine. Its a tool framework in which you give the input data and run the simulink model to check for the output. It’s used for model testing”. An interviewee involved with developing the test engine tool replied “It simulates the model functionality. Its automated, so when you give the input it runs the tests and generates the log with the output for the test. Currently, there is not much development as it is a legacy tool, which may get replaced in the future”. Also, it was identified that there were no guidelines available for the usage of the tool. “There are no written guidelines that tell you how to use it”. Here, the output of the test engine for the M1: Fuel application software simulink models was also viewed to identify and confirm the data.

From the data, new text could be identified and tagged. This tagged text is ‘developed at the company’, ‘generates log with output for test’, ‘test time for execution’. This data does not fit into the existing codes of the matrix and thus are inductively coded according the process described in Section 3.2.2. Here, the
Chapter 5. Data Analysis

Tagged text have been coded as: In house Developed, Test Execution Log, Test Time. Test engine is a tool which is built in house. Thus it is required to know if any tool qualification requirements from ISO 26262 are to be met. The execution record with the time can be a source of safety evidence that the test has been executed within time. Thus, these text have tagged and coded as they show or are required to show product based safety evidence for WPs. Thus, the class of test execution record, which consist of the codes, test time, title and identifier (i.e mandatory for all OSLC resources) was defined. The code of in house developed is grouped along with already defined codes for the category, Tool.

**Class: Test Execution Record**

Test time: 10.6 seconds (within time), Title: Test Engine result for Fuel application software simulink models

**Class: Test Environment**

Title: C++ programming language based simulation, Method name: Software in loop, Asil: Not Available

The target environment where the modules are tested is simulated. Test engine simulates the functionality of the model to test it for various input data. The environment on which the modules are tested is a C++ programming language based simulation. As seen from the transcripts “Test engine simulates the model functionality on the computer using C++ programming code ...”. Thus the method for the target environment used is concluded as software in loop, based on the description in section 4.1.2. Since no ASIL classification has been performed for the test object, there is no associated ASIL available for recommending the method for target environment.

**Class: Software Verification Plan**

Title: Plan for testing Fuel level calculated by M1:Fuel Simulink Models, Pass criteria: The output functionality is as expected, Exception handling action: Report error or bug in Jira tool, Regression strategy: Iterative testing cycles that incorporate for reported changes

The development practices at the company follow agile principles and as such there is lack of focus on documenting. For planning, the concerned team discuss and come to a consensus on the way to perform testing and verification for the modules. From the related interview transcripts, “In M1 we have only one application software, Fuel, that is used for both AE1 and AE2. The basic way of testing it is to run the models in the Test Engine and check if correct output signal are produced for all the tests. If so, then the tests have passed. In case if certain tests don’t pass, then we report it in Jira, which are then prioritised and resolved”. As a supplement a subject replied that “if any changes occur to the application software then we retest the changed parts in the next iteration...”.
Class: Test Case
Title: Test cases for testing the low fuel level and fuel level estimation functionality, Method name: Error guessing, Asil: Not Available, Unique identifier: TC-AE1 and TC-AE2
The test cases are developed in test suites which are executed using the respective testing tool. The test case files for the simulink models were provided, from which the data codes for unique identifiers was extracted. The TS and TC were the abbreviation used for test suites and test cases. The test suites are for the entire M1 ECU software and it comprises of test cases for the fuel application software and for other basic middle ware software. The related interview transcripts were “Test cases are written based on on personal experience and judgement”. Thus the method for deriving the test cases was identified as error guessing based on the ISO 26262 based enumeration constants. “The test case consists of data that is required to verify the functionality of the fuel application software ......”. The fuel application software realises AE1 and AE2 which are for estimating the fuel level and the low fuel level warning functionalities respectively. Since no ASIL classification is performed for the requirements, none is inherited by the design and its subsequent implementation (i.e test object).

Class: Test Data
Title: C programming language based test scripts for test cases, Input values: RTDB variable values with fuel level from middleware layer, Expected output: Post K1 filter expected fuel level to middleware layer
The test cases are written in C code and consist of data related to the fuel values given by the RTDB variables coming from the middleware software. The data in the test case files have been observed along with the transcripts from interviews such as “The RTDB input signals to the test engine are taken up by the K1 filter part of the model to calculate the fuel level. If it meets the specification of not deviating more than 4 percent of input fuel level, then it is reported out to the middleware software”.

Class: Calibration Data
Title: variants based calibration , Intent: To parametrize the interfaces of M1:Fuel application software, Dependencies: The dependencies with the signal and RTDB variables from middleware, Valid values: Setting the parameters within the simulink model to different values
From the relevant interview transcripts “I have calibration in my applications for some variants. The way I test the different variants is by setting the parameters within the model to different values depending on interfaces with middleware.” “The calibration is based on the RTDB variables and signals coming from the middleware software for testing”.

Class: Test Method
Title: Functional testing for Fuel application software, Method name: Interface test,
Asil: Not available
Simulink models needs to implement the functionality of the fuel application software. From the interview transcripts “Testing is against the requirements, it’s functional testing.” “For module test we simulate all the layers below the fuel application. These layers have modules which are common for all ECUs. So, only the application software changes for other ECUs.” As the middle layer is simulated, it becomes the interface for the application software with the hardware (See Figure 3.4). This method of testing is called, interface test.

Class: Software Verification Specification
Title: Specifications for testing the simulink models for fuel application software
The verification specification consists of the test cases and the test method used for testing the simulink models of the fuel application software. From the paraphrased interview transcripts “The specifications for testing are made in the test cases for the simulink models and are executed to check if the functionality of the interface to the middleware software works correctly”.

Class: Software Verification Report
Title: Test engine report for Fuel application software Simulink model, Pass result: All tests have passed successfully, Tailoring: Not applicable, Version: Based on the test suite for the 4th version of the test object
The screenshot of the test engine report was viewed to identify codes for the SVR. From the interview transcripts “Apart from the result generated by the tool, a formal report is written which consists of details of the test. In case of any negative results, the causes are presented in these reports and reported to Jira. But for the current version fuel simulink models all the tests have passed successfully”. From another interview transcript “The test report has all the tests as passed for the fuel simulink models”. Here, the version of the report is based on the version of the test object. The data from the formal report for testing and the test engine report indicated that the tests have passed along with the time it took to execute these tests.

Class: Architecture Design Specification
Title: Simulink models for M1: Fuel application software
Model based development is used for the M1 ECU software. Thus, the simulink models based on the requirements, depict the design specification as well as the design implementation. From the relevant transcripts in the interviews, “I read the specifications from the AERs for AE1 and AE2 to generate the simulink models. The models contain the implementation specifications for fuel application software”. Thus, the software unit design specification for fuel application software for M1 ECU are the the simulink models.
5.2.1.2 Data Codes for Software Integration Testing

Class: Test Object
Title: Target file for M1, Asil: Not available, Version: fourth release in 2016
The simulink models for fuel application software are executed to generate the .c and .h files which are compiled together with the basic software in the middleware to generate the target file. The target file is the implementation of all the software used in M1 ECU. Thus, the software component here is the integration of all the software modules, which is the embedded software of M1 ECU system. This was based on the relevant transcripts from the interviews of, “We generate the compiled code for fuel and then integrate it with the common middleware software to generate a target file for the entire M1 software which is then tested for ....”. Since no ASIL is assigned for the software units, there is no ASIL inherited for the test objects at this test level. Further, field notes were used to understand the software component as depicted in Appendix E.

Class: Tool
Title: Itest tool framework, Guidelines Used: Emulator tests usage guidelines, In House Developed: Yes
From the relevant interviews transcripts , “We take the C code of the target file and compile it into .dll files. Itest runs these .dll files in the simulated environment by stubbing off the lower level hardware... ” “Guidelines for usage are there in an emulator tests guide available in the company web portal”. “Currently, the Itest tool is being developed to incorporate for a wider range of input signals and data ....”. Thus, it indicates that the tool framework is being developed in house at the company.

Class: Test Environment
Title: C++ programming language based simulation, Method name: software in loop, Asil: Not Available
The target environment where the integration testing occurs is simulated. From the relevant transcripts in the interviews, “The environment in the Itest for running the .dll file is simulated using C++ programming code ”. “Itest uses a software in the loop type of a target environment”. Thus, here the method name for the target environment is software in loop. Since no ASIL classification has been performed for the test object, there is no ASIL inherited for recommending a particular method for simulating the target environment.

Class: Software Verification Plan
Title: Plan for testing target file of M1 ECU integrated software, Pass criteria: The output for test behaves as expected, Exception handling action: Report exception using Jira tool, Regression strategy: changes are implemented and updated into perforce database
From the relevant transcripts in the interviews, “The plan is for knowing what to use and how to perform testing. So in case, any errors occur then I know I can use Jira to report them and in case of changes I keep track of them using perforce”. As a supplement a subject responded “We plan the testing using Itest framework to check if the output fuel value from the built target file is as expected”. Thus, the codes as defined above have been extracted.

Class: Test Case
Title: Test cases for testing the low fuel level and fuel level estimation functionality
Method name: error guessing, Asil: Not Available, Unique identifier: TS-AE1, TS-AE2
The test cases are for testing the functionality of AE1 and AE2 realised by M1 embedded software. Thus as already stated by the subjects, the test cases are written in test suites, for the entire M1 ECU software, which consist of test cases that pertain to individual modules and for the integrated modules. From relevant interview transcripts, “The test cases for the M1 ECU system are written in test suite AE1 and AE2...”. Based the previous interview transcripts for test case at software unit testing, all test cases are written using personal experience and judgement and thus the method used for deriving the test cases is error guessing.

Class: Test Data
Title: C programming based test scripts for test cases, Input values: fuel sensor input and CAN bus inputs, Expected output: CAN bus outputs and signals with post fuel level
The test cases are written in C code and consist of data related to the fuel level values given by the various sensors and CAN buses (see Nomenclature) coming from other ECUs. The test data in the test cases was observed to verify that it is C programming based. From the relevant interview transcripts, “Itest is used for testing CAN stimuli, where data is logged. The test cases are written with CAN stimuli and the expected output fuel value the CAN bus should report is based on the C1 calibration parameters”.

Class: Configuration Data
Title: Build Cubl used for configuring the target file., Intent: To build the target file for M1 integrated software, Dependencies: used for identifying the related C1 calibration parameters, Valid values: values in Build Cubl
Configuration is done during run time for building the code for the target file that is being tested. Such a configuration is required to determine the data to be used to calibrate, when executing the tests. With different configurations of the software code, the calibration data when executing tests also differs. From the interview transcripts “I compile the software for the integrated models using Build Cubl and then create the built target file. Then this built software file is calibrated...”.
Class: Calibration Data
Title: parameters for Build target file, Intent: used for parametrising the build target file, Dependencies: dependent on various diagnostic data values of the CAN inputs and outputs pertained to the built target file, Code valid values: automatically generated values
From the relevant transcript in the interviews “The output fuel value the CAN bus should report for the executed test cases is based on the calibration data. The calibration is provided using parameters that are pre determined with a wide range of data values for a specific configuration”. Thus, the calibration data is provided while executing the test cases.

Class: Test Method
Title: Functional testing of M1 integrated software, Method name: Interface based test, Asil: Not available
From the interview transcripts “It is essentially black box testing where the test methods are mirroring the embedded software functions. Due to resource scarcity, we develop tests that can be run on our computers. Thus we simulate for load, the voltages, the processor speeds, the bandwidth for communication etc. coming from the middleware software by using the Itest framework”. The method used is for depicting the functional behaviour of the M1 software by sending input to and output out of the Itest framework. Since it tests the interface with the middleware, interface based tests was concluded as the test method adopted.

Class: Software Verification Specification
Title: Specifications for testing the M1 target file
The software verification specification consists of the test cases and the test method used for testing the target file from integrating the M1 software modules. Codes are derived from the interview transcripts of “The specifications for testing is the same as for module testing. We specify all data required for executing the test cases written for the target file ”.

Class: Software Verification Report
Title: Result of testing the software target file for M1, Pass result: All the tests have passed successfully, Tailoring: None, Version: fourth version
The codes have been derived from the related interview transcripts of “The test report for Itest is in from of an console log in the command prompt of the computer. For the fourth version of the target file, there are no exceptions or errors and all the tests have passed”. This data was also identified and confirmed based on a formal technical report generated for the test by the interviewee.

Class: Architectural Design specification
Title: Functional Allocation description (FAD)
From the interview transcripts “We look at the FAD. Its design document that assigns the functionality to a group of AEs, the variables, the CAN bases, the registers, pins...,and their interaction”. Thus, the ADS WP type at this test level should be the software architecture design specification according the ISO 26262 standard. But, FAD produced at the company is used as the design specification.

### 5.2.1.3 Data Codes for Verification of Software Safety Requirement Testing

**Class: Test Object**

**Title:** M1 embedded software, Asil: not available, Version: fourth version

The M1 embedded software is the test object to be verified for realising the allocation element requirements. From the interview transcripts “We test to check if the M1 embedded software satisfies all the AERs specifications for AE1 and AE2” . From the technical data report [68] it was observed and confirmed that no ASIL have been assigned for the software specific AERs. In the previous test level, it was already identified that the test object was the fourth version of the integrated software for M1 (i.e embedded software from integration of all software modules in M1 ECU system).

**Class: Tool**

**Title:** Tag coverage tool, Guidelines Used: none, In house developed: Yes

Testing and verification is mostly performed using tools and tool frameworks. Tag coverage, an in house developed tool, has been used for verifying the AER coverage in M1. From the interviews, “...then tag coverage runs the emulator tests to generate the coverage report for M1....The way tag coverage does this is by running the test cases pointed and stored in Jenkins. I only determine the software to be tested and request for the corresponding test cases in Jenkins tool. Tag coverage then automatically runs the tests”. Thus, here no guidelines for usage have been identified as tag coverage is an automated tool that run the test cases pointed by the Jenkins tool.

**Class: Test Environment**

**Title:** Python programming language based target environment, Method name: ECU emulator test, Asil: Not Available

The hardware of the M1 ECU on which the embedded software works is emulated. This emulated environment is used for conducting the tests. From the relevant interview transcripts, “For the M1 ECU software, we stub of the lower level hardware and emulate it...” Since, there is no ASIL classification for the SSR, there is no ASIL available for recommending a particular method for the type of target environment.
**Chapter 5. Data Analysis**

**Class: Software Verification Plan**
Title: Plan for verifying the AER for AE1 and AE2 in M1 embedded software, Regression strategy: Regression suite handled by Jira, Pass criteria: The functionality of all the AERs must be satisfied, Exception handling action: Raising an issue in Jira tool

From the related interview transcripts “The plan is to check if all the AERs are satisfied in the M1 embedded software by running the test cases in Jenkins using Tag Coverage. Tag coverage emulates the test environment...”. From another interviewee “The verification is successful if all the AER specifications are completely fulfilled in the M1. In case they are not, an exception is raised in Jira which is reported to the concerned.....Any changes made to the requirements are reported and verification is once again done for the affected parts of M1 embedded software”.

**Class: Test Case**
Title: Test suite for AERs verification, Method name: Analysis of requirements, Asil: Not Available, Unique identifier: TS-fuel

From the interview transcripts “The test suites are managed in Jenkins, which helps to point to the relevant test cases to be run using tag coverage. Test suites are for checking if the AERs for FLDS function are fulfilled in M1 embedded software”. The test suites in Jenkins were viewed to determine the unique identifier as TS-fuel. Here, the test cases are based on specification of the AERs. Thus the method used to derive test cases is analysis of requirements from the description provided in Section 4.1.2. Since, no ASIL classification is performed, verification of whether a particular test case derivation method was used cannot be done.

**Class: Test Data**
Title: Python programming language based test scripts, Input data: Based on the functionality specified in the requirements, Expected output: Valid range of values for the functionality in the requirements

From the interview transcripts “we give data based on the requirements in AERs such as sensor signal values, start-up and shut down voltages.....and the resultant values for these are based on the specification in the AER for the functionality under consideration. So, if the test results don’t fall within the expected range then the AER is considered to not have been realised correctly in the software”. Since, there are a lot of input values, not all the values are stated. Rather the data code extracted and presented above represents all such kind of input values.

**Class: Configuration Data**
Title: M1 configuration for SOP, Intent: Build the M1 embedded software for variant 1 of FLDS, Dependencies: Is basis for the C1 Calibration parameters to be used, Valid values: Values from SOP

The related interview transcripts are “Based on the current SOP (see nomencla-
Chapter 5. Data Analysis

ture), we configure the M1. The SOP is for liquid fuel variant of trucks and is used to realise FLDS functionality.” “The SOP for fourth version of M1 software is used with its corresponding C1 calibration parameters for testing”. From the technical data report on FLDS [68], it was noted that truck with liquid fuel is variant 1 of FLDS.

**Class: Calibration Data**

*Title: C1 Calibration parameters, Intent: Parametrise the embedded software for execution, Dependencies: Dependent on the run time configuration of the embedded software, Valid values: Automatically generated values*

From the relevant transcript in the interviews “The data in the test cases relies on the execution of the calibration data, which provides for a wide range of data values so as to account for all situations. The C1 Calibration parameters are predetermined and are executed automatically to calibrate the SOP for M1. It has incorporated diagnostic as well as RTDB test signals”. Thus, the calibration data is provided while executing the test cases. From the Test report generated it was noted that the calibration data was specified automatically using C1 Calibration parameters for the considered M1 SOP configuration.

**Class: Test Method**

*Title: Black box testing of M1 embedded software, Method name: Requirements based test, Asil: Not available*

The verification of the M1 embedded software involves checking if the functionality of the AERs are met. From the related interview transcripts, “Its black box testing where the behaviour of the software is tested to meet all required specifications”. Since no ASIL classification is performed, no verification whether a particular test method recommended by the standard is being followed was performed. Since the tests were based on verifying requirements, it can be stated that the method used is requirements based tests.

**Class: Software Verification Specification**

*Title: Specifications for AERs coverage in M1 embedded software*

The specification for verifying the M1 embedded software mainly comprises of the test cases and its related data. “The specifications for testing the M1 embedded software are the test cases written for black box testing for generating the requirement coverage report”.

**Class: Software Safety Requirement**

*Title: Software Specific Allocation Element Requirements for AE1 and AE2*

Requirements are defined for the allocation elements. The AEs are realised by the software. Since there is no ASIL classification, requirements have not yet been classified as safety related or not. From the interviewees “There are no explicit requirements written for software implementation, but some of the AERs
are composed of specifications pertaining only for the functionality of the software used in ECU systems”.

Class: Software Verification Report
Title: Requirement coverage report generated by tag coverage, Pass result: All the software specific AERs are covered in M1 embedded software, Tailoring: None required, Version: fourth version
“Tag coverage is used to generate the AER coverage report for the SOP of the fourth version of the M1 embedded software”. Tag coverage generates a coverage report that consist of the the requirement profile for the tests conducted. From the transcripts of an interview, “I perform the coverage test for the 38 AERs used in M1. Currently all the software specific AERs for the M1 used in FLDS are testable. The coverage is a 100 percent”. This requirement coverage report differs from the verification report in terms of the attributes that describe it. This can be a source of safety evidence and thus has been inductively coded and categorised into a new class as depicted below.

Class: Requirement Coverage
Total requirements: 38, Number tested: 38, Number not testable: 0, Percentage: 100
Chapter 6

Discussion and Limitations

The ISO 26262 specific resource types for extending the OSLC domain have been proposed in the conceptual meta model. A total of 16 resource types depicted as classes in the meta model are the proposed extensions to OSLC domains. Out of which 14 resource types are proposed extension to OSLC-QM domain that pertains to the testing phase of the ALM. The traceability between the WP types was defined to understand the flow among them. By realising the traceability requirements of ISO 26262 in the conceptual meta model, the WP types that must be exchanged between the tools for software testing and verification have been identified and defined. This exchange is for realising the work product flow among the WP types of SVP, SVS, SVR and with ADS or SSR, depending on the software test level under consideration. By defining traceability among SVP, SVR, SVS we realise traceability between the product based safety evidence that were grouped to compose these three WP types (i.e traces between the parent class transfers to traces between the child classes that compose these parent classes).

As seen from the analysis of Clause 9, 10 and 11 presented in section 5.1, all the three main WP types of SVP, SVR and the SVS can be composed and refined incrementally across the three software test levels.

From investigating the software testing and verification performed for M1 ECU system, gaps with respect to WPs and their attribute values have been identified. The ASIL classification was yet to be performed. Thus there could not a be verification whether the recommended methods suggested in ISO 26262 are being followed. Metrics have not been used to verify the structural coverage of the functionality being tested by the test cases at software unit and software integration test levels. Also, configuration data has not been used for software unit testing. According to ISO 26262, the testing of the implementation must be performed against the ADS defined at the different test levels. These ADS are in turn based on the requirement specifications. But for M1 ECU, functional or behavioural tests were performed, where the implementation was based on the requirement specifications. Thus, instead of a software architecture design, a Functional allocation description was identified as the ADS at software integration test level. Hence, there were certain WPs and attribute values that still need to be
produced in order to have all the required products as safety evidence for software testing. The implications of conducting interviews and instantiating with data in the second cycle was to confirm the usability of the meta model with respect to real data for an ECU software testing. Further, it had helped to inductively analyse and define new data as extensions to the conceptual meta model. By defining the RDF, the usability of the conceptual meta model could be tested for providing a structure for OSLC resource types. Further, RDF validator was used to confirm that the definition of the RDF was syntactically correct in terms of its structure.

From the validation results, it could be concluded that the conceptual meta model has been effective to a great extent in achieving compliance to ISO 26262. This has been concluded from the statistical analysis of the data collected for the validation questionnaire. The conceptual meta model could be extended and refined in the second cycle. Most of the effort involved was for defining the conceptual meta model in the first cycle of the case study. Once defined, any required extensions or additions that were identified in the second cycle of the case study were relatively easier and faster to update, extend and refine in the conceptual meta model. Such extensions or refinements can be made with relatively less effort in the model for any additional WP types or attributes required for ECU systems. This is due to the validated confirmability, traceability and abstraction in the structure of the conceptual meta model. The conceptual meta model can be used to distinguish the context, traces and extent of the involved concepts (i.e. product based safety evidence) using UML elements of packages, frames, relationship types. Any additional extensions, updates and refinements can incorporated within the relevant package, frame or by using a suitable relationship type. Thus, the effort that could be required to make modifications to the conceptual meta model would be based on the experience with embedded systems and expertise in ISO 26262 of the person performing the modification. Such changes would be required to be made for new revisions of the ISO 26262 standard.

When new versions of the software used in the safety critical systems are released and tested, the data in the WPs meta modelled as WP types in the meta model would also be required to be updated. For additions or change in traces, the impact analysis of the update in the conceptual meta model, on the existing traces would need to performed to check if any existing traces would need to be updated, changed or deleted. For ex: an addition or changes in traceability links in the meta model would require an impact analysis of the change on the WPs and their traceability links managed by the interoperating tools. The corresponding effort such an impact would require could also be investigated. Similarly, changes to the WP types in meta model would require an estimation of the effort that would be required for making the changes in the actual resources (i.e WPs) managed by the interoperating tools. This could be a probable scope for future work, where the impact analysis of the updates to the conceptual meta model on the work products and their traceability links maintained by the interoperating tools can
be investigated. Thus, the RDF based conceptual meta model once implemented can be used by tools to make updates, changes, additions and deletions to the work products based on the estimated effort. These updates can then be made and shown in a time and cost effective way by using OSLC services for compiling the traceable WPs as safety evidence for software testing and verification in ISO 26262.

6.1 Threats to Validity

Every research is subject to certain limitations which are a threat to validity. Some of the threats to validity for the case study based on the classification provided by Yin in [66] have been presented in this section.

6.1.1 Construct Validity

Construct validity is the extent to which measures used and analysis performed by the researcher reflects the subject of study as stated in the research questions [66]. The data analysis method as suggested in [61] was followed to the best of knowledge to assure a structured and valid process for defining the conceptual meta model and its instances. Triangulation of multiple data sources in the two cycles of the case study helped to further improve the construct validity. The data from the ISO 26262 standard along with the company specific presentation helped in identifying reliable codes for defining the conceptual meta model in the first cycle of case study. Use of multiple sources of data such as documents, observations and interviews helped to improve the validity of the results for the second cycle of the case study.

6.1.2 Internal Validity

Internal validity is concerned with investigating causal relations to verify that there are no unknown factors that could affect the results of the study. If such factors exists, then there is a threat to internal validity [66]. As a novice researcher was performing the research, there were bound to be a few threats to the internal validity. The guidance of the supervisor (i.e experienced researcher) helped to make the research process more reliable. But for the validation, the questionnaire was performed by face to face interviews which leads to the threat of interviewer bias. To mitigate this threat, claims were added for the each question to make the questionnaire self contained. This aids in understandability and reduces interaction to minimal. Thus, it helps to reduce the interviewer bias. But, it leads to another threat of researcher bias. This is a threat that could not be mitigated. Further, to decrease the interviewer bias, anonymity of the respondents was maintained [83]. In the second cycle of the case study, software
testing for M1 ECU system used in FLDS system was a representative case at the company. Thus, software testing performed for most other ECU systems at the company would involve similar WPs. Though, it should be noted that minor differences or changes could be existing for a diverse case [57] of software testing performed for ECUs at the company. The data collection in the second cycle of the case study was performed by maintaining the anonymity of the interviewees to obtain unbiased and honest responses.

6.1.3 External Validity

External validity is the extent to which the results can be generalized [66]. The conceptual meta model defined in the first cycle of the case study can be used and extended with respect to other ECU systems used in heavy automobiles like trucks. Thus, the results of the first cycle of the case study could be generalized to most of the ECU software developed for heavy automotive vehicles. A general misconception of the case study research is that it is not generalizable. This misconception as addressed in [89], is explained for the current context. In terms of generalizability for other heavy automotive manufacturers, an in detail context and characteristics of the case has been presented (noting down considerations of confidentiality), so that analytical generalizability can be developed. This can assure similar results under similar setup for other case studies within the heavy automotive manufacturing industry and improve the generalizability of the research findings.

6.1.4 Reliability

Reliability is the extent to which the analysis and results are dependant on the concerned researcher. For research to be reliable, the results should be valid when repeated [66]. The reliability of qualitative content analysis is often presented by using terms such as confirmability [90], which is one of the criteria used for validating the effectiveness of the conceptual meta model. The external supervisor, a researcher working on ISO 26262 at the company, has examined the codes extracted for the defining the extended conceptual meta model of the case study and judged them to increase their reliability [62]. The RDF defined from the instantiation of meta model was validated using the RDF validator of W3C official website [43]. In order to improve the reliability of the interview data in the second cycle of the case study, cross checking of the transcribed and paraphrased data was performed with the corresponding interviewees. For the literature review performed, a threat to reliability was the use of a few literature sources which were not peer reviewed. This was due to the relative recentness of the topic in the research domain. It was also observed from the fact that most of the peer reviewed work related to the topic have been published from 2010 and later.
6.1.5 General Limitations

The limitations that are general to the research presented in thesis as a whole have been specified in this section.

- For the thesis, a literature review was performed and not a systematic literature review. A systematic study of all the relevant sources of literature has not been presented in the thesis as the purpose was to get an understanding of the concepts required for the background of the study.

- A general limitation was pertaining to including only those terminologies in the nomenclature that needed further elaboration for their understanding. Any remaining terms used in the thesis which have not been included in the nomenclature were explained to the required detail in the thesis.

- Manual coding was performed for analysing the units of analysis in the first two cycles of the case study. Use of electronic qualitative analysis tools eases and speeds up the process of coding. But, coding for qualitative data is an intellectual process and still requires the deliberation of the researcher in spite of using tools. Also, given that an unmanageable number of documents or interviews were not being considered in either of the first two cycles of the case study, manual coding was used [91].

- Certain data with respect to the case of M1 ECU system have been referenced and mentioned ambiguously in the report due to confidentiality.
Chapter 7

Conclusions

7.1 Conclusions

Compiling of traceable safety evidence of software testing and verification of an ECU system for generating ISO 26262 safety case is cumbersome and challenging. The safety case is needed for ISO 26262 certification. The WP types are needed to be produced as product based safety evidence for the safety case. Thus, there was a need to identify the required WP types and link (i.e. trace) them according to ISO 26262 traceability requirements, so that their compilation can be achieved. By using OSLC based tool interoperability, the WP types (i.e. safety evidence) linked across the application lifecycle can be maintained, updated and changed. This leads to an easy, cost and time efficient compilation of the defined traceable product based safety evidence. For this the WP types must be depicted in the structure required for OSLC resources (i.e. RDF). Thus, the proposal is for extending the OSLC domain resource types targeting ISO 26262, so that they can be exchanged and used among tools that produce them. This proposed structure is defined in form of a RDF based conceptual meta model (i.e. UML class diagrams).

RQ1 and RQ2 have been answered in Section 4.1.2, by the conceptual meta model with the compilation of traceable WP types as safety evidence for ISO 26262 software testing and verification clauses. The WP types were represented using UML class diagrams to depict the structure for OSLC resources. These are the ISO 26262 specific resource types proposed as an extension to OSLC resource types. The testing, refining and extending of the conceptual meta model with respect to software testing at the unit, integration and software safety requirement verification levels, for a truck ECU system at Scania was performed as presented in Section 4.3. RQ3 was answered by the empirical investigation used for instantiating the WPs types in the conceptual meta model and proposing them as OSLC resources. This is done by defining them in RDF as presented in Section 4.3.2. Testing the practical applicability and extending of the conceptual meta model (see Section 4.3.1) was achieved as the results of the second cycle in the case study. Thus, RQ1 and RQ2 were also partially realised in the results of the second cycle of the case study. Finally, RQ4 was answered by empirically verifying the
effectiveness of the extended and tested conceptual meta model for compliance to ISO 26262. The validation resulted in the proposed meta modelled resource types achieving confirmance, abstraction, interoperability and traceability with respect to ISO 26262 standard to ‘a great extent’, as presented in Section 4.4. A defined and structured qualitative analysis method was followed for defining the conceptual meta model from the units of analysis in the first and second cycle of the case study. Thus, the model generated could achieve compliance to ISO 26262 to a great extent of effectiveness. Although, it also implies that there is still scope to refine and improvise the model based on more relevant interpretations and revisions being contemplated for the 2018 version of the ISO 26262 standard for road vehicles above 3.5 tones.

Thus, the thesis achieves a proposal that defines a RDF based conceptual meta model, to represent the ISO 26262 based traceable work product types for software testing and verification. This model can be used as the basis for continuous reassessment with respect to the work deliverables required for safety case of ISO 26262. The knowledge representation of the product based safety evidence for software testing and verification in ISO 26262 has been achieved. This representation can be used for, firstly, identifying if all WPs for software testing are being produced. Secondly, to define the traces among the WP types to link the tools that produce them, in order to compile the safety evidence. Hence, the aim with proposing ISO 26262 based OSLC resource types is aid in querying, updating, modifying and maintaining the defined WPs in the meta model, by interoperating tools. This results in an cost effective and less time consuming way of compiling the safety evidence.

The results of the research help the software testers and heavy automotive manufacturers of safety critical ECU systems to identify and define the traceable WP types that must be available in the data sources of the interoperating tools. These tools can realise the traceability for compiling the product based evidence of software testing and verification part of ISO 26262 safety case.

7.2 Future Work

As part of future work, a similar knowledge representation of the other parts and clauses of the ISO 26262 standard can be performed to compile the product based evidence for the entire safety case. Another step would be the representation of process based knowledge (i.e safety evidence) for software testing and verification in ISO 26262. The proposed model has only been tested for software testing of one ECU system of M1. Further, test of usability for diverse cases of software testing and verification for ECU systems can be performed. This could lead to further refining and extending of the conceptual meta model.

In addition to the validation criteria used in the thesis, more sophisticated performance profiles can be applied for validation. Knowledge Space Theory
for predicting answer patterns based on conceptual meta model is one such performance profile. Here, the model is validated by comparing the predicted answer patterns with empirically obtained answer patterns. Another approach for validating the content of the conceptual meta model is to compare it with other empirically defined conceptual meta models representing the same knowledge. For instance, tables can be used for comparing the models, detailing the number of propositions that are contained either in both or in only one of them. This number is used to calculate the convergence score, which is the basis for validation [76]. Further, the questionnaire based validation can be performed encompassing a larger population set to increase the credibility and applicability of the results.

Further, the different ways of defining the meta modelled concepts using RDF based constraint languages like Resource Shape (ReSh), Shape Expressions (ShEx), SPARQL Inferencing Notation (SPIN), among others, can be investigated.


Model-Driven Engineering and Software Development (MODELSWARD), IEEE, pp. 1–6, 2015.


[34] B. B. Konka, “A case study on software testing methods and tools: A pre-study on software testing requirements of ISO/DIS 26262,” University of Gothenburg, July 2012.


[74] A. Lacey and D. Luff, Qualitative data analysis, Trent Focus Sheffield, 2001.


Chapter 8

Appendix

8.1 Appendix A: Invitation Letter for Participation in Case Study

Greetings,
This is Kathyayani Padira, Master thesis student at Scania

Short Introduction:

I am pursuing my masters in software engineering, at Blekinge Institute of Technology, Sweden. I am working on my thesis project in software engineering titled “Investigation of Resource Types for OSLC domains targeting ISO 26262: Focus on Traceable safety evidence for the Right side of the ISO 26262 Software V-model”. I am performing my thesis under, Barbara Gallina and Mattia Nyberg, RESA and have their consent to approach you in this regard.

Research Objective:
My aim is to identify software testing and verification related WPs that are used or developed for M1 ECU used in the variant 1 of the FLDS. Then to understand the traceability that exists between the WPs, to understand which ones need to interoperate.

Request for a meeting:
I have a couple of questions that I would like to ask you related to my aims stated above. Could you please suggest a time and date of your convenience to schedule for a meeting interview?
Thank you for your time!
PS: Anonymity would be protected.

Note:
Anonymity of the interview participant is fully preserved.

Regards
Kathyayani Padira
Background Information of the Interviewees
1. What is your job role?
2. What is your experience working with embedded software (in years)?
3. At which software test level are you involved for M1 ECU system?
   1. Module testing
   2. Module Integration testing
   3. AER verification testing
8.2 Interview Guide

8.2.1 Appendix B: Close-ended Questionnaire for Interviews

Software Unit Testing Level

1. Do you have a plan for testing the software modules of the M1 ECU system?
2. Do you use any tools or tool frameworks for testing?
3. Do you have any plan or strategy to handle change requests (i.e. regression strategy)?
4. Is there a target environment in which you run the test cases for testing?
5. Do you an object (part or whole of M1) that you test?
6. Do you write or generate test cases for the object under test?
7. Do you assess the coverage of the test cases that you write?
8. Do you have test scripts or data that contains the data for the testing?
9. Is there any method that you use for testing?
10. Do you configure the software (i.e test object) for test cases used for testing?
11. Do you generate a report for your test result?
12. Do you calibrate the data for the generating the test result?
13. Do you have any software unit design specification used as the basis for testing or verifying against?

Software Integration Testing Level

1. Do you have a plan for testing the integrated software modules of the M1 ECU system?
2. Do you use any tools or tool frameworks for testing?
3. Do you have any plan or strategy to handle change requests (i.e. regression strategy)?
4. Is there a target environment in which you perform the testing?
5. Do you an object (part or whole of M1) that you test?
6. Do you write or generate test cases for the object under test?
7. Do you assess the coverage of the test cases that you write?
8. Do you have test scripts or data that contains the data for the testing?
9. Is there any method that you use for testing?
10. Do you configure the software (i.e test object) for testing?
11. Do you generate a report for your test result?
12. Do you calibrate the data for the generating the test result?
13. Do you have any software architecture design specification used as the basis for testing or verifying against?
Software Safety Requirement Testing Level

1. Do you have a plan for verifying the M1 ECU software for satisfying the allocation element requirements?
2. Do you use any tools or tool frameworks for testing?
3. Do you have any plan or strategy to handle change requests (i.e regression strategy)?
4. Is there a target environment in which you perform the testing?
5. Do you an object (part or whole of M1) that you test?
6. Do you write or generate test cases for the object under test?
7. Do you assess the coverage of the test cases that you write?
8. Do you have test scripts or data that contains the data for the testing?
9. Is there any method that you use for testing?
10. Do you configure the software (i.e test object) for testing?
11. Do you generate a report for your test result?
12. Do you calibrate the data for the generating the test result?
13. Do you have requirements used as the basis for the verification to be performed against?

8.2.2 Appendix C: Open Ended Questionnaire for Interviews

Software Unit Testing Level

1. What is the module for your test? Do you have any ASIL assigned for the module?
2. What data do you use to configure the module?
3. what is your method for testing? (i.e kind of testing you perform?)
4. What is the target environment in which you perform the tests?
5. What are the data and test cases written for testing the module?
6. How and what kind of data do you use to calibrate the module for testing?
7. What are the different tools used, how and for what?
8. Do you have any plan for testing the module and a criteria for assessing the test results?
9. How do you handle errors or exceptions?
10. What is the final result of the testing?
11. How do you perform software testing using the products elicited so far?

Software Integration and Testing Level

1. which modules do you integrate in M1 ECU for performing software integration testing? Do you have ASIL assigned to any of these models?
2. What data do you use to configure the Target file?
3. What is your method for testing (i.e. kind of testing you perform)?
4. What is the target environment in which you perform the tests?
5. What are the data and test cases written for testing the module?
6. How and what kind of data do you use to calibrate the build target file?
7. What are the different tools used, how and for what?
8. Do you have a plan for testing the target file and a criteria for assessing the test results?
9. How do you handle errors or exceptions?
10. What is the final result of the testing?
11. How do you perform software testing using the products elicited so far?

Verification of Software Safety Requirement Testing Level

1. What is the object being verified for realising the AER for M1 ECU?
2. What data do you use to configure the M1 embedded software?
3. What is your method for testing (i.e. kind of testing you perform)?
4. What is the target environment in which you perform the tests?
5. What are the data and test cases written for verifying that all the AERs are covered in M1 embedded software?
6. How and what kind of data do you use to calibrate the M1 embedded software?
7. What are the different tools used, how and for what?
8. Do you have any plan for verifying the AERs and a criteria for assessing the test results?
9. How do you handle errors or exceptions?
10. What is the final result of the test verification?
11. How do you perform software testing using the products elicited so far?
8.3 Appendix D: Validation of the Conceptual Meta Model

Background information of the Respondents
1. What is your job role?
2. How many years of experience do you have?
3. What is your level of expertise in the ISO 26262 standard?

Protocol for Data Collection for the Questionnaire
For each question, a claim based on the proposed conceptual meta model is formulated as the basis for answering the questionnaire. These claims elaborate the context in the defined conceptual meta model for which the question was formulated. The questionnaire is appended with the images of the conceptual meta model is shown to the subjects. Thus, all measure to ensure minimal interaction have been considered.
Validation Questionnaire

1. Do the models describe software testing related work product types based on requirements from ISO 26262?

   5. Yes  4. To a great extent  3. To a good extent  2. To a little extent  1. No

Claim: In the meta model, the work products types like SVP, test environment, tool, the object to test, structural coverage metrics etc. required by ISO 26262 are presented as classes. These classes have attributes like the ASIL and its recommended methods for various activities, the version of the object being tested etc. So, do such classes and attributes depict work product types that are required for software testing by ISO 26262?

2. Do the models depict the traceability links between the work product types required by ISO 26262?

   5. Yes  4. To a great extent  3. To a good extent  2. To a little extent  1. No

Claim: The meta model in compliance with ISO 26262 defines that the SVP must be the basis for SVS and the SVR must be generated by executing the SVS. The SVP and SVS are for verifying the architecture design specifications or requirements. In this way the various work products for software testing are linked with each other. So do these kind of relationships represent the traceability between the work products?

3. Do the models expose the required work product types for realizing the software testing work flows based on ISO 26262?

   5. Yes  4. To a great extent  3. To a good extent  2. To a little extent  1. No

Claim: The meta models uses packages that help to abstract the defined work products types into three main software testing work product types of SVP, SVS and SVR. This abstraction is for realizing the work flow required by the work products sub clause in ISO 26262.

The meta model uses frames to abstract the software testing and non software testing based work product types against which the testing is to be performed. This abstraction exposes the work flow for realizing the objective sub clause of ISO 26262. It helps to understand the interacting work product types from different phases of ALM that need to exchange data.

So, does the meta model helps to identify the work product types that must be exposed for the particular software testing workflow under consideration?

Figure 8.1: Questionnaire for Validation of the Tested and Extended Conceptual Meta Model
Validation Results

Figure 8.2: Histogram with Results of Validation
8.4 Appendix E: Screen-shot from Express Scribe and Screen Shots of Field Notes

Figure 8.3: Express Scribe- Example Screen Shot 1

Figure 8.4: Field note: Example 1 and 2
8.5 Appendix F: Screen-Shots From W3C RDF Validator

Figure 8.5: W3C RDF validator: The input RDF syntax based on the Conceptual Meta Model
Figure 8.6: W3C RDF validator: Result For Confirmity to OSLC Resource Structure
8.6 Appendix G: OSLC Resources - RDF for the Three Main WPs For Software Testing in ISO 26262

At Software Unit Test Level

![Software Verification Plan]

Figure 8.7: RDF syntax for Work Products - SVP and SVS
Figure 8.8: RDF syntax for Work Products- SVR

At Software Integration Test Level

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oslcd_iso26262q="http://open-services.net/ns/iso26262q#"
    xmlns:oslcd_iso26262qm= "http://open-services.net/ns/iso26262qm#
    rdf:about="http://iso26262qInterop.org/verificationReports/14"
    oslcd_iso26262qm:title Test engine report for fuel application software Simulink model"
    oslcd_iso26262qm:passResult All tests have passed successfully
    oslcd_iso26262qm:passTestLevel software unit test</oslcd_iso26262qm:testLevel>
    oslcd_is26262q:version based on the test suite for the 4th version of the test object</oslcd_is26262q:version>
    oslcd_is26262qm:tailoring Not applicable</oslcd_is26262qm:tailoring>
    oslcd_is26262qm:Tool rdf:resource="http://myserver/Tools/1"
    oslcd_is26262qm:calibrationDataSpecification rdf:resource="http://myserver/calibrationData/1"
    oslcd_is26262qm:generatorBasedSoftwareVerificationPlan rdf:resource="http://myserver/verificationPlans/1"
    oslcd_is26262qm:generatorBasedSoftwareVerificationSpecification rdf:resource="http://myserver/verificationSpecifications/1"
    oslcd_is26262qm:SoftwareVerificationReport>
    </rdf:RDF>
```

Figure 8.9: RDF syntax for Work Products- SVP

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oslcd_iso26262q="http://open-services.net/ns/iso26262q#"
    xmlns:oslcd_iso26262qm= "http://open-services.net/ns/iso26262qm#
    rdf:about="http://iso26262qInterop.org/verificationReports/14"
    oslcd_iso26262qm:title Test engine report for fuel application software Simulink model"
    oslcd_iso26262qm:passResult All tests have passed successfully
    oslcd_iso26262qm:passTestLevel software unit test</oslcd_iso26262qm:testLevel>
    oslcd_is26262q:version based on the test suite for the 4th version of the test object</oslcd_is26262q:version>
    oslcd_is26262qm:tailoring Not applicable</oslcd_is26262qm:tailoring>
    oslcd_is26262qm:Tool rdf:resource="http://myserver/Tools/1"
    oslcd_is26262qm:calibrationDataSpecification rdf:resource="http://myserver/calibrationData/1"
    oslcd_is26262qm:generatorBasedSoftwareVerificationPlan rdf:resource="http://myserver/verificationPlans/1"
    oslcd_is26262qm:generatorBasedSoftwareVerificationSpecification rdf:resource="http://myserver/verificationSpecifications/1"
    oslcd_is26262qm:SoftwareVerificationReport>
    </rdf:RDF>
```
Figure 8.10: RDF syntax for Work Products- SVS and SVR

At Software Safety Requirement Verification Test Level
1. NAME: Software Verification Plan

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:oslc_iso26262qm="http://open-services.net/ns/iso26262qm#">
  <oslc_iso26262qm:SoftwareVerificationPlan
    rdf:about="http://iso26262qmsupplier.com/verificationPlans/1">
    <!-- Attributes of SVP class -->
    <oslc_iso26262qm:title>Plan for testing CMSI ECU integrated software target file</oslc_iso26262qm:title>
    <oslc_iso26262qm:testLevel>Software safety requirement verification</oslc_iso26262qm:testLevel>
    <oslc_iso26262qm:passCriteria>The output functionality is as expected</oslc_iso26262qm:passCriteria>
    <oslc_iso26262qm:exceptionHandlingAction>Report exception using Jira tool</oslc_iso26262qm:exceptionHandlingAction>
    <oslc_iso26262qm:regressionStrategy>changes are implemented and updated into perfecive database</oslc_iso26262qm:regressionStrategy>
    <!-- Predicates: grouped classes to define parent class, SVP -->
    <oslc_iso26262qm:tool rdf:resource="http://myserver/tools/1"/>
    <oslc_iso26262qm:testObject rdf:resource="http://myserver/testObjects/1"/>
    <oslc_iso26262qm:testEnvironment rdf:resource="http://myserver/testEnvironments/1"/>
  </oslc_iso26262qm:SoftwareVerificationPlan>
</rdf:RDF>

Software Verification Specification

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:oslc_iso26262qm="http://open-services.net/ns/iso26262qm#">
  <oslc_iso26262qm:SoftwareVerificationSpecification
    rdf:about="http://iso26262qmsupplier.com/verificationSpecifications/1">
    <oslc_iso26262qm:title>Specifications for AERs coverage in CMSI embedded software</oslc_iso26262qm:title>
    <oslc_iso26262qm:testLevel>software safety requirement verification</oslc_iso26262qm:testLevel>
    <oslc_iso26262qm:testObject rdf:resource="http://myserver/myapp/testing/testObjects/1"/>
    <oslc_iso26262qm:testCase rdf:resource="http://myserver/myapp/testing/testCase/1"/>
    <oslc_iso26262qm:usesSoftwareVerificationPlan rdf:resource="http://iso26262qmsupplier.com/verificationPlans/1"/>
    <oslc_iso26262qm:usedForVerifyingSoftwareSafetyRequirement rdf:resource="http://iso26262qmsupplier/architectureDesignSpecifications/1"/>
  </oslc_iso26262qm:SoftwareVerificationSpecification>
</rdf:RDF>

Figure 8.11: RDF syntax for Work Products- SVP and SVS
Figure 8.12: RDF syntax for Work Products-SVR
Figure 8.13: Original Enterprise Architect UML Class Diagram
Figure 8.14: Original Enterprise Architect UML Class Diagram
Figure 8.15: Original Enterprise Architect UML Class Diagram
Figure 8.16: Original Enterprise Architect UML Class Diagram