



Università degli Studi di Napoli Federico II  
Ph.D. Program in  
Information Technology and Electrical Engineering  
XXXVII Cycle

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

# Enhancing Software Development Processes for Industrial Software Systems

by

MARCO DE LUCA

Advisor: Prof. Anna Rita Fasolino

Co-advisor: Pasquale Cimmino



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE



*Do or do not. There is no try.*



# ENHANCING SOFTWARE DEVELOPMENT PROCESSES FOR INDUSTRIAL SOFTWARE SYSTEMS

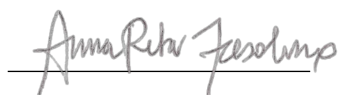
Ph.D. Thesis presented  
for the fulfillment of the Degree of Doctor of Philosophy  
in Information Technology and Electrical Engineering  
by

**MARCO DE LUCA**

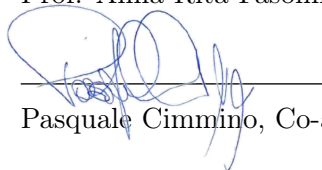
October 2024



Approved as to style and content by

A handwritten signature in black ink, reading "Anna Rita Fasolino", written over a horizontal line.

Prof. Anna Rita Fasolino, Advisor

A handwritten signature in blue ink, reading "Pasquale Cimmino", written over a horizontal line.

Pasquale Cimmino, Co-advisor

Università degli Studi di Napoli Federico II

Ph.D. Program in Information Technology and Electrical Engineering

XXXVII cycle - Chairman: Prof. Stefano Russo



<http://itee.dieti.unina.it>

## **Candidate's declaration**

I hereby declare that this thesis submitted to obtain the academic degree of Philosophiæ Doctor (Ph.D.) in Information Technology and Electrical Engineering is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.

Parts of this dissertation have been published in international journals and/or conference articles (see list of the author's publications at the end of the thesis).

Napoli, December 10, 2024

A handwritten signature in black ink, appearing to read 'Marco De Luca', is written over a horizontal line.

Marco De Luca

# Abstract

The automotive industry is transforming as software becomes more integrated into vehicles, now containing millions of lines of code and multiple Electronic Control Units (ECUs). Software is now central to vehicle function and innovation, but it also brings challenges in development, quality assurance, and compliance with safety standards.

This thesis addresses these challenges by improving the software development process in the automotive domain, focusing on software documentation and adherence to industry regulations. It presents several contributions, including a community detection methodology within developer networks to improve team formation by identifying experts with the right skills. In addition, the thesis introduces a new software architecture documentation model for safety-critical domains, compliant with ISO 26262, to improve traceability and maintainability. This template has been validated through industrial case studies, enhancing long-term software reliability. To bridge the gap between design and implementation, the thesis also proposes a software architecture recovery (SAR) tool to automate the generation of architectural documentation from code bases, improving system understanding and ensuring accurate documentation. Finally, a framework for software architecture metrics is introduced to support continuous compliance processes. By identifying suitable metrics, this framework helps integrate compliance into industrial practices, ensuring adherence to safety standards and internal policies.

In conclusion, this research improves software development tackling key challenges in team collaboration, documentation and compliance, enabling innovation and maintaining high standards of safety and reliability.

**Keywords:** Documenting Software Architecture, Software Architecture Design, Software Architecture Recovery, Community Detection, Automotive Domain





## Sintesi in lingua italiana

L'industria automobilistica si sta trasformando a causa della crescente integrazione del software nei veicoli, che ora contengono milioni di righe di codice e diverse Unità di Controllo Elettronico (ECU). Il software è sempre più centrale per il funzionamento e l'innovazione dei veicoli, ma pone sfide nello sviluppo, nella qualità e nella conformità agli standard di sicurezza.

Questa tesi affronta queste sfide migliorando il processo di sviluppo software nel settore automobilistico, concentrandosi sulla documentazione del software e sull'aderenza alle normative del settore. Presenta diversi contributi, tra cui una metodologia di rilevamento delle comunità all'interno delle reti di sviluppatori per migliorare la formazione dei team, identificando esperti con le giuste competenze. Inoltre, viene introdotto un nuovo modello di documentazione dell'architettura software per domini critici per la sicurezza, conforme alla norma ISO 26262, per migliorare la tracciabilità e la manutenibilità. Questo modello è stato validato attraverso casi di studio industriali, migliorando l'affidabilità del software. Per colmare il divario tra progettazione e implementazione, la tesi propone uno strumento di recupero dell'architettura software (SAR) per automatizzare la generazione della documentazione architetturale a partire dal codice sorgente, migliorando la comprensione del sistema e garantendo una documentazione accurata. Infine, viene introdotto un framework di metriche per l'architettura software a supporto dei processi di conformità continua. Identificando metriche adeguate, questo framework aiuta a integrare la conformità nelle pratiche industriali, garantendo l'adesione agli standard di sicurezza e alle politiche interne.

In conclusione, questa ricerca migliora lo sviluppo del software affrontando sfide cruciali nella collaborazione dei team, nella documentazione e nella conformità, permettendo l'innovazione e mantenendo elevati standard di sicurezza e affidabilità.

**Parole chiave:** Documentazione dell'architettura del software, progettazione dell'architettura del software, recupero dell'architettura del software, rilevamento della comunità, automotive

---

## Acknowledgements

The author's work has been supported by the PhD scholarship funded by Micron Semiconductor Italy, as part of the collaboration with the University of Naples "Federico II" and the DIETI department. The research activities were conducted in collaboration with the "FW Development for managed NAND" team of Massimo Iaculo, Pasquale Cimmino, Paolo Papa and Salvatore Del Prete.



# Contents

Abstract . . . . .	i
Sintesi in lingua italiana . . . . .	iii
Acknowledgements . . . . .	v
List of Acronyms . . . . .	xi
List of Figures . . . . .	xv
List of Tables . . . . .	xviii
List of Symbols . . . . .	1
<b>1 Introduction</b>	<b>1</b>
<b>2 Challenges in Automotive Software Development</b>	<b>7</b>
2.0.1 Multidisciplinary Software Development . . . . .	9
2.0.2 Software Documentation in Automotive Systems De- velopment . . . . .	11
2.0.3 Industry Standards and Frameworks . . . . .	13
<b>3 A Community Detection Approach Based on Network Rep-     resentation Learning for Repository Mining</b>	<b>21</b>
3.1 Introduction . . . . .	21
3.2 Related Work . . . . .	24
3.2.1 GitHub Information Models . . . . .	24

3.2.2	Developer Social Networks . . . . .	25
3.2.3	Community Detection on Developer Social Networks	25
3.3	Framework . . . . .	26
3.3.1	Task Definition . . . . .	27
3.3.2	Modeling GitHub as a DSN . . . . .	28
3.3.3	Community Detection process . . . . .	31
3.3.4	Running Example . . . . .	34
3.4	Experimental analysis . . . . .	36
3.4.1	Goals . . . . .	36
3.4.2	Research Questions . . . . .	37
3.4.3	Variables and Metrics . . . . .	37
3.4.4	Objects . . . . .	40
3.4.5	Design of Experiments . . . . .	41
3.4.6	Results . . . . .	42
3.4.7	Example . . . . .	49
3.4.8	Threats to Validity . . . . .	50
3.5	Conclusions . . . . .	52
<b>4</b>	<b>Documenting Software Architecture Design in Compliance with the ISO 26262: a Practical Experience in Industry</b>	<b>55</b>
4.1	Introduction . . . . .	56
4.2	Related Works . . . . .	58
4.2.1	SAD issues and challenges in industry . . . . .	58
4.2.2	SAD issues and challenges in automotive domains . .	59
4.3	Industrial survey . . . . .	60
4.4	Proposed Template . . . . .	62
4.4.1	The documentation template . . . . .	64
4.4.2	Implementation of the proposed SAD template . . .	68
4.4.3	Mapping between challenges and solutions . . . . .	73
4.5	Industrial Case Study . . . . .	74

4.6	Conclusion and Future Work . . . . .	80
<b>5</b>	<b>Automated Architecture Recovery for Embedded Software Systems: An Industrial Case Study</b>	<b>83</b>
5.1	Introduction . . . . .	84
5.2	Related Studies . . . . .	85
5.2.1	Software Architecture Recovery (SAR) . . . . .	86
5.2.2	Reverse Engineering of State Chart Diagrams . . . . .	87
5.3	The proposed reverse engineering process . . . . .	87
5.4	Implementation Details . . . . .	90
5.5	Experimental evaluation . . . . .	96
5.6	Conclusion and Future Works . . . . .	102
<b>6</b>	<b>Characterizing Software Architectural Metrics for Continuous Compliance in the Automotive Domain</b>	<b>103</b>
6.1	Introduction . . . . .	104
6.2	Background and Related Studies . . . . .	105
6.2.1	Continuous Compliance . . . . .	106
6.2.2	Metrics for Architecture . . . . .	108
6.3	Research methodology . . . . .	108
6.3.1	<b>Step 1:</b> Metrics gathering and description . . . . .	110
6.3.2	<b>Step 2:</b> Framework definition . . . . .	113
6.3.3	<b>Step 3:</b> Metrics characterization and evaluation . . . . .	118
6.4	Results and Discussion . . . . .	119
6.4.1	RQ1: <i>Which are the architectural metrics proposed in the literature that can be used in the Continuous Compliance of automotive software architectures?</i> . . . . .	119
6.4.2	RQ2: <i>How can these metrics be characterized?</i> . . . . .	121
6.5	Threats to validity . . . . .	127
6.5.1	External validity threats . . . . .	128
6.5.2	Internal validity threats . . . . .	129

6.6 Conclusion and Future Works . . . . .	129
<b>7 Conclusions</b>	<b>131</b>
<b>Bibliography</b>	<b>135</b>
<b>Author's publications</b>	<b>157</b>



# List of Acronyms

The following acronyms are used throughout the thesis.

<b>ML</b>	Machine Learning
<b>RQ</b>	Research Question
<b>SAD</b>	Software Architecture Design
<b>SAR</b>	Software Architecture Recovery
<b>ECUs</b>	Electronic Control Units
<b>AUTOSAR</b>	AUTomotive Open System ARchitecture
<b>DSNs</b>	Developer Social Networks
<b>SNA</b>	Social Network Analysis
<b>BERTO</b>	emBEdding gRaph communiTy detectiOn
<b>E/E</b>	Electric and Electronic
<b>ASIL</b>	Automotive Safety Integrity Level
<b>HARA</b>	Hazard analysis and risk assessment
<b>QM</b>	Quality Management

<b>SW</b>	Software
<b>CI/CD</b>	Continuous Integration/Continuous Deployment
<b>EA</b>	Enterprise Architect
<b>AST</b>	Abstract Syntax Trees
<b>IR</b>	Intermediate Representation
<b>OTA</b>	over-the-air
<b>CCA</b>	Continuous Compliance Assurance
<b>KPI</b>	Key Performance Indicator
<b>SPI</b>	Safety Performance Indicator

# List of Figures

- 2.1 V-Model with a distinction between the OEM and supplier  
contributions . . . . . 14
- 2.2 ISO 26262 structure [175] . . . . . 16
- 2.3 AUTOSAR architecture [1] . . . . . 19
  
- 3.1 Framework overview . . . . . 27
- 3.2 Conceptual Model of a Collaborator Software Repository . . 28
- 3.3 Developer Social Network Model . . . . . 31
- 3.4 DSN representation for the Running Example. The hetero-  
geneous graph is composed by 11 nodes and 12 edges. . . . 35
- 3.5 Running times by varying the embedding size in *BERTO* . 48
- 3.6 Running times by varying *K* value in *BERTO* . . . . . 49
  
- 4.1 Model of the Software Architecture Design Documentation  
Template . . . . . 66
- 4.2 Details on the Software Component Architecture Design  
Documentation . . . . . 67
- 4.3 Documentation Structure of the proposed template inside EA 69
- 4.4 Example of High-Level Hierarchical View . . . . . 70
- 4.5 Detailed Component Diagram . . . . . 71
- 4.6 Component-and-Connector (C&C) View . . . . . 72

4.7	High-level modeling of the interruptions flow . . . . .	72
4.8	Interruptions Level Sequence Diagram . . . . .	73
4.9	Sequence Diagram for Scheduling Properties . . . . .	74
4.10	Usage example in Enterprise Architect . . . . .	75
4.11	xml file exported from EA . . . . .	76
4.12	Box plot showing the results of the survey for answering <i>RQ2</i> . . . . .	78
5.1	The proposed SAR process . . . . .	89
5.2	Package Diagram browser view . . . . .	91
5.3	Reconstructed Package Diagram . . . . .	91
5.4	Class Diagram reconstructed by Enterprise Architect . . . . .	93
5.5	Component Diagram reconstructed by the tool . . . . .	93
5.6	C&C Diagram reconstructed by the tool . . . . .	94
5.7	Task Code Example . . . . .	96
5.8	Reconstructed Task State Chart Diagram . . . . .	96
5.9	Box plot showing the accuracy distribution of diagrams re- constructed automatically by the tool . . . . .	98
5.10	Identified categories of missing information and their rela- tive keyword . . . . .	100
5.11	Distribution of the missing information categories . . . . .	100
5.12	Pie chart on the closed-ended question: “ <i>Based on my ex- perience in using the tool, I find it useful in supporting the comprehension of the system.</i> ” . . . . .	101
5.13	Pie chart on the closed-ended question: “ <i>Which reconstructed models, whether generated automatically or manually, do you find most useful for understanding the system?</i> ” . . . . .	101
6.1	Overview of the research methodology . . . . .	109
6.2	Distribution of the selected primary studies per publication year . . . . .	113
6.3	The proposed Framework . . . . .	118

6.4	Distribution of the metrics over the Continuous Compliance Evaluation Score . . . . .	120
6.5	Distribution of the metrics over the <i>Assessment Approach</i> domain of the Framework . . . . .	122
6.6	Distribution of the metrics over the <i>Input Artifact Type</i> domain of the Framework . . . . .	123
6.7	Distribution of the metrics over the <i>Metric Description Type</i> domain of the Framework . . . . .	124
6.8	Distribution of the metrics over the <i>Experimental Setting</i> domain of the Framework . . . . .	124
6.9	Distribution of the metrics over the <i>Experimental Objects Type</i> domain of the Framework . . . . .	125
6.10	Distribution of the metrics over the <i>Measured Property</i> domain of the Framework . . . . .	126
6.11	Distribution of the metrics over the <i>Output Category</i> domain of the Framework . . . . .	127
6.12	Distribution of the metrics over the <i>Application Field</i> domain of the Framework . . . . .	128



# List of Tables

1.1	Thesis structure . . . . .	6
2.1	Methods for the verification of the software architectural design outlined in the ISO 26262 . . . . .	15
3.1	Distance matrix . . . . .	36
3.2	Dataset characterization . . . . .	41
3.3	Design of Experiments: Phase 1 involves the determination of the most effective embedding technique in terms of modularity, from among the three options available. This process is carried out for each of the three embedding techniques. Phase 2 involves fixing the embedding technique selected in Phase 1 and considering the information model as the independent variable. Phase 3 involves changing the analyzed sample while keeping the information model and clustering technique constant, and observing the variations in modularity that occur across different values of cluster and embedding size. Phase 4, the execution time is established as the dependent variable, and the impact of sample size and clustering and embedding dimensions is evaluated.	43

3.4	(RQ1) - Results of <i>BERTO</i> algorithm in terms of modularity by varying technique and embedding size . . . . .	44
3.5	(RQ2) - Comparison of the average modularity, varying <i>K</i> , for <i>BERTO</i> and <i>ABDCI</i> [90] . . . . .	44
3.6	(RQ2) - Comparison of the Median and Standard Deviation (STD) of community members, varying <i>K</i> , for <i>BERTO</i> and <i>ABDCI</i> [90] . . . . .	46
3.7	(RQ3) - Evaluation of Modularity for the four considered repository samples (Full, Python, Java, Android) . . . . .	47
4.1	Challenges in developing software architecture design emerged from the Industrial Survey and the Literature Study . . . .	63
4.2	Traceability Matrix between Challenges and Solutions . . . .	75
6.1	Selected Primary Studies . . . . .	112
6.2	Example of metrics description and evaluation according to the proposed Framework . . . . .	115
6.3	Continuous compliance Metrics Evaluation . . . . .	121



# Chapter 1

## Introduction

The automotive industry is experiencing a paradigm shift, driven by the rapid and extensive integration of software into modern vehicles. Today's vehicles rely on software not only to optimize traditional functionalities such as engine performance and fuel efficiency but also to enable cutting-edge technologies, including advanced driver assistance systems (ADAS), autonomous driving capabilities, and vehicle-to-everything (V2X) communication. These innovations mark a significant departure from the mechanical focus of traditional automotive engineering, positioning software as the core enabler of modern automotive advancements. As a result, vehicles are rapidly evolving into software-centric systems, with millions of lines of code distributed across dozens of interconnected Electronic Control Units (ECUs), sensors, and embedded systems.

This shift towards software-driven vehicles has brought with it an exponential increase in complexity, with challenges that extend beyond the technical realm. Automotive software development now requires coordination across large, multidisciplinary teams, often spread across different geographic locations and working under tight timelines. Team collaboration has become a critical factor in ensuring the success of software projects, as misalignment or inefficiencies in team structures can lead to costly delays or errors [161, 168, 67]. Furthermore, the intricate nature of modern automotive software demands meticulous documentation practices to ensure that system designs, architectural decisions, and implementation details are comprehensively captured and maintained. Such documentation is es-

essential not only for facilitating ongoing development and maintenance but also for meeting regulatory and industry standards [88, 167, 36, 174].

Safety and compliance further compound these challenges, particularly in the context of safety-critical systems such as those governed by ISO 26262 [92]. This international standard emphasizes the importance of functional safety in automotive systems, requiring rigorous adherence to processes that guarantee the reliability and safety of software components. The need to consistently align with these standards places additional challenges on developers and organizations, as they must ensure traceability, auditability, and compliance across all phases of the software development lifecycle [28].

Addressing these multifaceted challenges requires more than isolated technical solutions. A holistic, multidisciplinary approach is essential to combine innovative technical tools and methods with robust organizational strategies. On the technical side, advancements in automation, tools for software architecture recovery, and frameworks for continuous compliance can help streamline processes and reduce the likelihood of errors. On the organizational front, fostering effective collaboration among diverse teams, implementing clear documentation standards, and establishing practices for continuous improvement are equally critical. Together, these approaches can enable the automotive industry to navigate the complexity of modern software systems while maintaining the highest standards of safety, reliability, and innovation.

To tackle these challenges, this thesis is guided by the following Research Goal (RG):

**Research Goal:** *To enhance software development processes in the automotive domain by improving team formation, maintaining high-quality documentation, and ensuring continuous compliance with safety standards such as ISO 26262.*

This Research Goal is broken down into three specific Research Objectives (ROs):

- RO1. To improve team formation and collaboration in software development projects.
  - RO2. To design a standardized software architecture documentation tem-
-

plate and a tool-based Software Architecture Recovery (SAR) to support documentation practices and compliance with the ISO 26262 standard.

RO3. To establish a framework for evaluating architectural metrics that support continuous compliance with safety standards.

By addressing these objectives, this thesis seeks to bridge critical gaps in software development practices within the automotive industry, enabling better alignment between design, implementation, and compliance processes and providing valuable insights for both researchers and practitioners in this field.

## Thesis Contributions

To tackle the outlined challenges and achieve the Research Goal, this thesis presents several key contributions. Each contribution is directly mapped to a specific Research Objective, ensuring a clear and focused alignment between the research efforts and the identified objectives:

- *A community detection technique and tool for identifying developer communities within developer social networks to facilitate team formation* (RO1): this contribution focuses on addressing the challenges of forming effective software development teams. It introduces a novel community detection technique that leverages social network analysis to identify groups of developers who naturally collaborate or have complementary skills. By detecting these communities within developer social networks, the technique helps project managers and team leaders assemble teams that are more cohesive and effective.
  - *A software architectural documentation template that complies with the requirements set by ISO 26262* (RO2): this contribution involves the definition of a template for documenting software architectures, specifically tailored to meet the requirements of the ISO 26262 functional safety standard. The template provides a structured approach to document elements of software architectures in a way that aligns with safety-critical guidelines. By doing so, it facilitates clear, consistent, traceable, and compliant documentation of the software systems. This template ensures that the architectural documentation
-

not only supports the development and maintenance of safety-critical systems but also enhances traceability, auditability, and compliance with ISO 26262.

- *A tool-based Software Architecture Recovery (SAR) technique to support documentation practices in industrial settings (RO2)*: the proposed tool-based Software Architecture Recovery (SAR) technique is designed for the recovery of architectural documentation for existing industrial software systems. The SAR technique helps reverse-engineer the software architecture from the existing codebase, automatically generating updated architectural documentation. This approach aims to make easier for software teams in industrial environments to maintain accurate and up-to-date documentation, ultimately improving the quality and sustainability of complex software systems.
- *A framework to characterize architectural metrics in support of continuous compliance processes in the industrial domain with respect to safety standards and guidelines (RO3)*: this contribution introduces a comprehensive framework designed to evaluate architectural metrics that are critical for ensuring continuous compliance with safety standards and guidelines in industrial software development. The framework allows to systematically assess various architectural attributes, such as modularity, coupling, cohesion, and maintainability, which are essential for ensuring long-term system reliability and safety.

This thesis includes material from the following research papers already published in peer-reviewed journals or conferences:

- M. De Luca, A. R. Fasolino, A. Ferraro, V. Moscato, G. Sperlí, P. Tramontana. A community detection approach based on network representation learning for repository mining. *Expert Systems with Applications*, Volume 231, 2023, DOI: 10.1016/j.eswa.2023.120597. [59]
  - D. Amalfitano, M. De Luca, A. R. Fasolino. *Documenting Software Architecture Design in Compliance with the ISO 26262: a Practical*
-

---

*Experience in Industry*, IEEE 20th International Conference on Software Architecture Companion (ICSA-C), 2023, DOI: 10.1109/ICSA-C57050.2023.00022. [16]

- D. Amalfitano, M. De Luca, D. F. De Angelis, A. R. Fasolino. *Automated Architecture Recovery for Embedded Software Systems: An Industrial Case Study*, 18th European Conference on Software Architecture (ECSA), 2024, DOI: 10.1007/978-3-031-70797-1\_4. [18]
- D. Amalfitano, M. De Luca, A. R. Fasolino, P. Pelliccione and T. Santilli. *Characterizing Software Architectural Metrics for Continuous Compliance in the Automotive Domain*, IEEE 21st International Conference on Software Architecture (ICSA), 2024, DOI: 10.1109/ICSA59870.2024.00025. [15]

## Thesis Outline

Table 1.1 presents an overview of the thesis structure, showing how each proposed contribution maps to a specific Research Objective (RO), the chapter where it is detailed, and the reference to the paper on which the chapter is based.

The remainder of the thesis is organized as follows.

Chapter 2 provides an overview of the evolution of software in the automotive domain and the emerging challenges that have come with this shift. It sets the context for the rest of the thesis, highlighting how software has grown to become central to vehicle functionality and what this means for future development.

Chapter 3 introduces our proposed solution for community detection within developer social networks. This solution helps to identify the right experts within teams, which is critical for tackling the increasing complexity of automotive software.

Chapter 4 presents a new documentation template designed to improve the software documentation process, especially in cases where compliance with safety standards like ISO 26262 is required. This template addresses the need for thorough, high-quality documentation in safety-critical systems.

Chapter 5 introduces our software architecture recovery tool, which is aimed at addressing the common problem of keeping software documenta-

---

tion aligned with the actual code. In many industrial settings, documentation often falls behind due to time and budget constraints, and this tool helps ensure that documentation remains up-to-date and accurate.

Chapter 6 outlines a framework we propose for characterizing software architecture metrics in industrial settings. This framework is designed to enhance the compliance process, ensuring that software systems meet both industry standards and internal quality requirements.

**Table 1.1.** Thesis structure

Contribution	Addressed RO	Chapter	Reference
Community Detection Technique	RO1	Chapter 3	[59]
ISO 26262-Compliant Documentation Template	RO2	Chapter 4	[16]
Software Architecture Recovery (SAR) Tool	RO2	Chapter 5	[18]
Framework for Architectural Metrics	RO3	Chapter 6	[15]

---

# Challenges in Automotive Software Development

Embedded software systems play a crucial role in today's technology landscape, but their development poses unique challenges [68, 75, 12, 22]. These systems, which are tightly integrated with hardware components, are designed to perform very specific functions, often under real-time constraints. The complexity comes from the need for the software to work in hardware environments that have limited processing power, memory, and energy, making performance optimization critical. Additionally, safety and reliability are key concerns, especially in domains like automotive, aerospace, or healthcare where failure can have severe consequences [101, 171, 149].

In particular, over time in the automotive domain, software has increasingly become the core of vehicle control [82, 53, 85]. Initially introduced to optimize engine performance, software now plays a pivotal role in virtually every function of a modern vehicle, from infotainment systems to advanced driver assistance systems (ADAS). Modern cars are highly dependent on electronics, and consumers increasingly view vehicles as platforms for software-driven innovation [32]. Today's vehicles are highly dependent on software, containing over 100 million lines of code (LOC) distributed across more than 100 Electronic Control Units (ECUs). Each ECU functions as a specialized computer, responsible for a specific vehicle subsystem [177, 35]. A decade ago, the number of lines of code in a car was much

smaller, but the rapid rate of technological innovation has significantly increased both the volume and complexity of automotive software. This trend is expected to continue, with future vehicles incorporating even more complex software systems as the industry moves towards fully autonomous driving [174].

While the growing role of software presents great opportunities for innovation, it also introduces significant challenges that threaten to undercut the industry's ability to scale and adapt. Managing the complexity of modern software systems is one of the most pressing challenges. One key issue arises from the growing complexity of software development processes. Managing this complexity requires improving both the effectiveness and efficiency of these processes. However, achieving this is no simple task and involves tackling several critical issues.

One key challenge is the increasing difficulty of forming effective development teams. Assembling the right team is a complex process that demands careful attention to skills, communication, and collaboration. Poorly structured teams can result in misaligned efforts and communication breakdowns compromising the quality of the final product. Effective team formation is particularly crucial in the multidisciplinary environment of automotive software development, where mechanical, electrical, and software engineers must collaborate seamlessly.

Another critical aspect is ensuring adherence to industry standards and frameworks. Compliance with standards like ISO 26262 is essential in the automotive sector, as these guidelines ensure the safety, reliability, and interoperability of software systems. However, meeting these requirements involves maintaining high levels of traceability, consistency, and rigorous documentation practices throughout the development lifecycle. This creates additional challenges, particularly in managing the complexity of modern automotive software systems, which require the production of numerous artifacts to document compliance.

Addressing these challenges is essential for the continued advancement of the automotive industry as vehicles become increasingly software-driven. By adopting industry standards, improving team collaboration, and enhancing documentation practices, the automotive sector can not only overcome current hurdles but also unlock new opportunities for innovation. A focused effort on multidisciplinary collaboration, safety standards, and ro-

---



bust software architecture documentation will ensure that the industry can maintain the high levels of safety and reliability required while driving forward innovation in today's complex and competitive environment.

In the following sections, we will provide an analysis of these challenges, and throughout the rest of the thesis, we will discuss potential solutions to enhance the overall software development process in the automotive industry

### **2.0.1 Multidisciplinary Software Development**

Software development is inherently a team-based activity. The success and quality of the final software product depend significantly on how well the development team is structured [211, 21]. However, forming an effective team is not a straightforward task. It involves considering several factors, such as the compatibility of team members in terms of skills, communication, and collaboration. The selection of the right team members is crucial activity, especially in industrial software systems, as it directly impacts the outcome of the project [84, 81]. Despite the importance of team composition, manually selecting and arranging team members is a challenging task. This responsibility often falls on experienced project leaders or managers, who must evaluate numerous factors, including individual technical expertise, specific roles within the team, and how well the team members will collaborate.

While modern team formation tools have been developed to support collaboration, challenges still persist [161, 168]. Issues such as delays, interruptions, and the need to reopen tasks are common. These disruptions can be symptoms of an ineffective team configuration, which may result from a lack of essential skills or poor communication among team members [20]. Such problems underscore the need for a decision support system that can assist in the selection of team members, taking into account their relevant experience, technical skills, and compatibility for teamwork. A system like this would improve the ability to form high-functioning teams, thereby reducing the risks associated with incompatible team setups. Incompatible team structures, such as teams composed of individuals who have never worked together before, can lead to significant project risks. Research by Fagerholm *et al.* [67] has shown that team-related factors, including communication abilities, flexibility, and compatibility among team

---

members, play a crucial role in the success of software projects. Poor team dynamics can result in inefficiencies and even project failure.

In the context of the automotive industry, the complexity of software development has increased dramatically. As modern vehicles become more advanced, the reliance on software and electronics in automotive systems has grown. These systems are no longer built using only traditional mechanical engineering techniques. Instead, modern vehicle design requires the integration of mechanical, electrical, and software components. This shift has led to the necessity for a multidisciplinary approach, where experts from different fields, such as mechanical, electrical, and software engineering, must collaborate closely throughout the development process [155]. The increasing complexity of automotive systems requires diverse expertise, making teamwork and collaboration more critical than ever [175]. Automotive projects frequently require tight integration between these different areas, adding to the complexity of project management. This means that project managers must not only oversee the technical aspects of software development but also coordinate across various disciplines, ensuring that all elements are integrated smoothly.

Due to the complexity of these projects, new approaches are required for building teams and promoting effective collaboration. A major challenge lies in identifying experts with the appropriate skill sets to address specific problems. Finding the right group of experts for complex tasks has broad applications across various fields, including industry and education. This challenge is commonly known as "Expert Finding" or forming teams of experts within social networks [44, 108]. This process is essential for ensuring that teams have the necessary expertise to address the growing software demands in modern vehicles. Expert finding goes beyond merely identifying individuals with relevant technical skills. It also involves assessing how well these individuals can collaborate within a team [14]. Having the right mix of skills in a team is vital for solving the technical challenges associated with the increasing complexity of automotive systems.

Forming effective teams for software development in the automotive industry requires careful consideration of many factors, including technical skills, communication abilities, and compatibility among team members. As software continues to play a central role in vehicle design and operation, new methods for selecting and organizing teams will become increasingly

---

important. Addressing these challenges is crucial for improving the overall process of developing industrial software systems.

### **2.0.2 Software Documentation in Automotive Systems Development**

The development of high-quality automotive software introduces several critical challenges, particularly in terms of quality assurance and long-term maintenance. As modern vehicles increasingly rely on software to manage key functionalities, the importance of ensuring the reliability and safety of this software has grown exponentially [98, 56]. Unlike traditional mechanical systems, software can fail in ways that are difficult to detect during the design phase but can have severe consequences once deployed in real-world conditions. Software defects in automotive systems can result in significant financial costs, including recalls, warranty claims, and production delays. In more severe cases, these defects can lead to safety issues that may endanger lives [150, 165].

A prime example of the growing concerns surrounding automotive software was the 2014 incident involving Honda Motor Co., which marked a pivotal moment in the industry. For the first time, the company identified that a malfunction in the Electronic Control Units (ECUs) caused vehicles to accelerate unexpectedly without driver input, leading to dangerous situations on the road [3]. This incident underscored the critical need for rigorous software testing and validation processes, as even a small defect in the software can trigger unpredictable and hazardous behavior in the vehicle. The increasing dependence on software in vehicles has also contributed to a significant rise in vehicle recalls related to software glitches or electronic defects. Over the past decade, several major automotive manufacturers have faced high-profile recalls due to such issues [5, 24, 142]. These recalls, often prompted by software failures in key systems such as braking, steering, and engine management, highlight the growing complexity of modern automotive software.

Given these challenges, it is imperative that quality assurance (QA) practices be implemented earlier and more rigorously in the software development lifecycle. Traditional testing methods, which are often applied at the end stages of development, may no longer be sufficient to catch all potential issues. Instead, there is a growing consensus within the auto-

---

motive industry that QA must begin in the architectural design phase of software development, where the fundamental structure of the software is defined [79, 37]. By addressing potential flaws at this stage, engineers can reduce the likelihood of defects making their way into production models. Keeping accurate and up-to-date the architectural documentation can benefit the software quality of the whole system [207, 197, 80, 116]. The architectural documentation serves as the blueprint for the software system, detailing how different components interact and outlining the overall structure of the software. Maintaining the desired qualities of a software system as it evolves requires a well-defined software architecture. The architecture must be clearly understood by all developers, ensuring that any modifications to the system are aligned with its structure. A suitable architecture is not driven solely by functional requirements; it is heavily influenced by various quality attributes such as performance, security, and maintainability [180, 38]. Despite this understanding, creating an architecture that effectively supports these attributes remains a challenging task. In their work Souza *et al.* [121] identified poor or outdated documentation as the major contributor to software defects, particularly those uncovered during the testing phases of development. Software architecture documentation is a critical resource not only for the development team but also for quality assurance processes, as it enables engineers and testers to understand the software's design and potential failure points. Software architecture serves as the crucial link between an organization's business objectives and the underlying software system. Designing and selecting an architecture that meets both functional needs and quality attributes, such as reliability, security, and performance, is essential for the system's success [140].

The importance of software documentation, particularly architectural documentation, cannot be overstated in the development of high-quality automotive software systems. As vehicles become increasingly software-dependent, maintaining clear, accurate, and up-to-date architectural documentation is essential for ensuring both the functionality and long-term sustainability of the system. Proper documentation serves as the foundation for effective quality assurance processes, allowing engineers to identify potential defects early in the development lifecycle and adapt the system as new requirements arise. Ultimately, the success of automotive software depends not only on robust architectural design but also on the practices that

---

support its evolution, making documentation a critical factor in achieving reliability, security, and performance.

### 2.0.3 Industry Standards and Frameworks

In response to these challenges, the automotive industry has adopted several standards to ensure that software systems are fault-tolerant and meet stringent safety requirements. One of the most important of these is ISO 26262 [92], a functional safety standard designed specifically for automotive systems. Another widely adopted solution is AUTOSAR (Automotive Open System ARchitecture) [176], which provides a standardized software architecture for the development of automotive control units. AUTOSAR ensures the interoperability and scalability of automotive systems while supporting the integration of various software components from multiple suppliers. Similarly, the MISRA C [123] standard is widely used to enforce safe coding practices in automotive software development. These standards, along with static analysis tools and other quality assurance methods, are essential for ensuring the safety and reliability of modern automotive systems. The use of such standards and guidelines is crucial in industrial software development. These standards provide a structured framework that minimizes the risks associated with software and hardware failures. By adhering to these well-established guidelines, developers can ensure consistency, reliability, and safety across all stages of development, from design to testing and implementation.

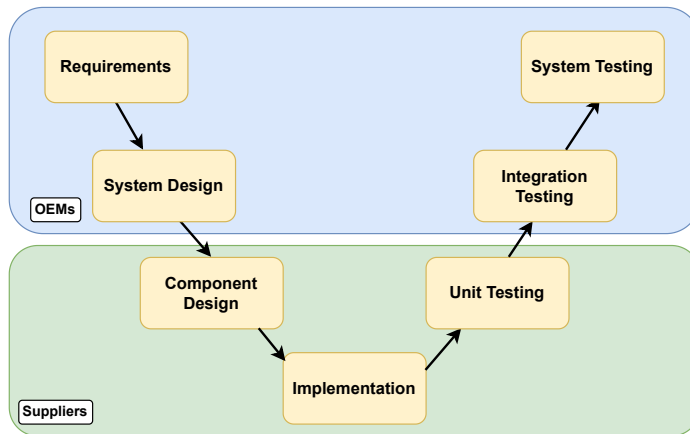
#### ISO 26262

ISO 26262 is a key safety standard designed for Electrical and Electronic (E/E) systems in the automotive domain, addressing the challenges associated with the development of safety-critical software. The standard applies to both hardware and software components, covering the entire system lifecycle. The main objective of ISO 26262 is to minimize risks associated with the failure of critical vehicle systems, particularly those that could result in injury or loss of life. Moreover, ISO 26262 aims to facilitate the reuse of both hardware and software components across different platforms and among different suppliers [69]. However, while the reuse of components can enhance efficiency, it also presents significant challenges.

---

For instance, testing becomes increasingly complex, and it is estimated that testing alone constitutes nearly 50% of the total development cost of automotive software [33].

ISO 26262 is built upon the "V" process model, a widely accepted system development methodology within the automotive industry. This model emphasizes a structured approach where system design, development, and testing are systematically aligned. In this model, OEMs are responsible for requirement specifications, system design, and final integration, while suppliers focus on actual software development for electronic control units (ECUs). Although suppliers perform initial unit testing, the responsibility for integration and acceptance testing falls to the OEMs, ensuring that the software implementation meets both functional and safety requirements. The challenge of this approach is that testing activities are often concentrated in the latter stages of development. As a result, many software defects may not be identified until late in the process, which increases the cost and complexity of addressing these issues [120].



**Figure 2.1.** V-Model with a distinction between the OEM and supplier contributions

ISO 26262 introduces the concept of Automotive Safety Integrity Levels (ASILs), which classify systems based on the level of risk they pose to vehicle safety. These levels, ranging from ASIL A (the lowest) to ASIL D

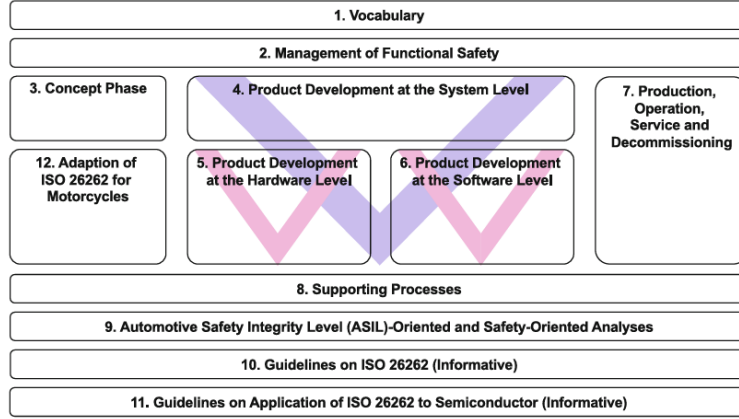
(the highest), guide the methods and techniques used in development and testing. Systems classified under higher ASILs, such as ASIL C or ASIL D, are subject to more stringent safety measures, including the use of formal verification methods to ensure software safety. An example of the methods required for the verification of the software architectural design based on their ASIL level is shown in Table 2.1.

Methods	ASIL			
	A	B	C	D
1a Walk-through of the design	+	+	o	o
1b Inspection of the design	+	+	+	+
1c Simulation of dynamic behaviour of the design	o	+	+	+
1d Prototype generation	-	o	+	+
1e Formal verification	-	-	o	+
1f Control flow analysis	-	-	o	+
1g Data flow analysis	-	-	o	+

**Table 2.1.** Methods for the verification of the software architectural design outlined in the ISO 26262

ISO 26262 is divided into 12 parts, as shown in Figure 2.2, covering various aspects of functional safety in automotive systems. This thesis focuses specifically on Part 6, which deals with software development, and more particularly, on clause 7 (ISO 26262 §6.7) regarding software architectural design (SAD). This section of the standard outlines guidelines for preventing systematic software failures by promoting best practices in software architecture. According to ISO 26262, a well-structured software architecture should possess key properties such as modularity, encapsulation, and simplicity. These characteristics are essential for ensuring that software components can be effectively tested, maintained, and updated, reducing the likelihood of introducing errors. To make sure the SAD meets these characteristics the ISO presents principles to be followed during the development of the SAD. For example, among the various recommendations of the standard we can find: a restricted size of the interfaces, strong cohesion and low coupling between components, and appropriate hierarchical structure of the software components. ISO 26262 provides a structured framework, that ensures that both OEMs and suppliers adhere to stringent safety requirements throughout the development process. While the standard helps to improve the safety and reliability of modern vehicles, it

---



**Figure 2.2.** ISO 26262 structure [175]

also presents significant challenges in terms of cost and complexity, particularly concerning software testing and safety assurance.

## AUTOSAR

In addition to functional safety standards, the automotive industry has adopted the AUTomotive Open System ARchitecture (AUTOSAR) [176] to manage the growing complexity of automotive software systems. AUTOSAR is a standardized framework designed to provide a common platform for the development and integration of electronic control units (ECUs) within vehicles. Introduced in 2003, AUTOSAR was established through a collaborative partnership of major automotive original equipment manufacturers (OEMs) and their software and hardware suppliers. Today, it includes over 200 global partners and is widely regarded as the de facto standard for automotive software development [2].

The adoption of AUTOSAR addresses the automotive industry's need for standardization in two key areas: methodology and architecture. As the responsibility for system design and verification primarily falls with original equipment manufacturers (OEMs), while implementation is often distributed across multiple suppliers, there was a clear need for a consistent and standardized methodology to guide the entire automotive software de-



velopment process. Similarly, a standardized reference architecture became essential to improve the reusability of software components developed by suppliers across different OEMs, ultimately reducing costs and improving efficiency [118]. To meet these needs, AUTOSAR established several key objectives to facilitate distributed design and development of automotive software systems:

- *Standardization of ECU Architecture*: by introducing a reference architecture, AUTOSAR enables reusable software components across different vehicle platforms, reducing redundancy, and enhancing efficiency.
- *Standardization of Development Methodology*: by providing a unified development process, AUTOSAR promotes collaboration between OEMs and suppliers, ensuring consistent ECU development and integration.
- *Standardization of Architectural Models*: AUTOSAR defines a common language for system models that facilitates seamless exchange of systems and ECU models between different modeling tools. This improves communication and efficiency across the development process, ensuring that all parties involved in the development process can work with compatible architectural representations.

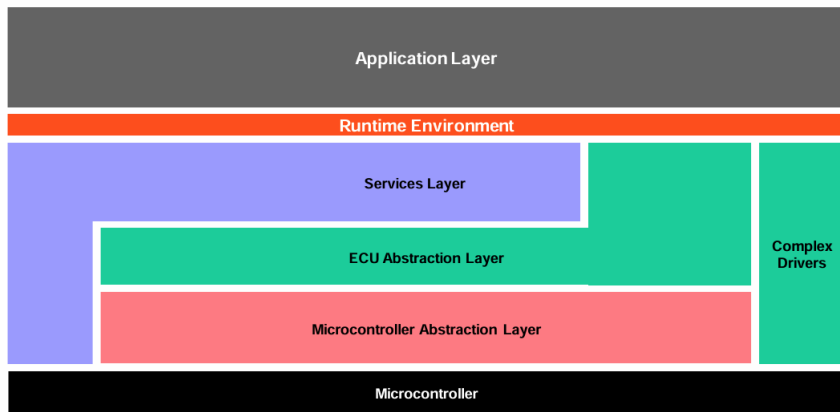
AUTOSAR introduces a reference architecture based on software components that operate at the electrical/electronic (E/E) architectural level. A software reference architecture is a generalized framework for a specific category of systems, serving as the basis for designing more specific system architectures within that category. [19]. AUTOSAR defines both the architecture and the interfaces of the software as a meta-model, as well as the file formats required for data exchange. Furthermore, the standard establishes its own development methodology, guiding the entire software development process [199].

**AUTOSAR Architecture** AUTOSAR adopts a layered software architecture that adheres to model-driven architecture (MDA) principles, an engineering method that separates software applications from the underlying hardware platform [191]. MDA focuses on the creation of mod-

---

els, including those for defining software applications, the computing platform, and the mapping of software components to computing nodes. This approach is particularly well-suited for embedded systems because it allows for easy adaptation of models as hardware and software requirements evolve. AUTOSAR provides a meta-model specifically tailored to the automotive domain to facilitate this process. To support the MDA approach in practice, AUTOSAR introduces a layered software structure for ECU development, composed of an application layer, a platform-dependent software layer (including OS services, communication drivers, etc.), and a middleware layer known as the Runtime Environment (RTE) [130]:

- *Application Layer*: This top layer contains the core controller software, which primarily consists of control algorithms within the automotive domain. The software in this layer is organized using a component-based architecture, with components modeled to communicate through defined ports and connections.
  - *Runtime Environment (RTE) Layer*: The RTE handles communication between software components as well as between software components and the basic software below. It provides a standardized interface at the application level, abstracting access to the computing platform's services. Vendors can supply different RTE implementations, offering middleware with standardized interfaces and services.
  - *Basic Software Layer*: This layer supports the fundamental functions of ECUs and is divided into three sublayers: the Service Layer, which manages essential ECU services such as the operating system, state management, diagnostics, memory, and communication; the ECU Abstraction Layer, which isolates hardware for the upper layers and provides access to hardware peripherals; and the Microcontroller Abstraction Layer (MCAL), which includes low-level drivers that interact directly with the hardware.
-



**Figure 2.3.** AUTOSAR architecture [1]

---



# A Community Detection Approach Based on Network Representation Learning for Repository Mining

**Abstract** In this paper, we propose a novel heterogeneous graph-based model for capturing and handling all the complex and strongly-correlated information of a software *Developer Social Network* (DSN) to support several analytic tasks. In particular, we challenge the problem of automatically discovering *communities* of software developers sharing interests for similar projects by relying on *Social Network Analysis* (SNA) findings. To overcome the huge graph-size issue, we leverage different graph embedding techniques. Eventually, we evaluate the proposed approach with respect to state-of-the-art approaches from an efficiency and an effectiveness point of view by carrying out an experiment involving the GitHub dataset.

## 3.1 Introduction

In the last two decades, the git control version system has been adopted by millions of software developers for efficiently managing their projects

and easily disseminating their work. GitHub<sup>1</sup> is one of the most diffused version control systems that further provides online open-access to git repositories. It has emerged as a leading choice for developers and researchers seeking to collaborate and share projects using git. GitHub provides query mechanisms and APIs for browsing, searching, and extracting relevant information from its repositories. Based on such mechanisms, several techniques and approaches have been proposed for efficiently and effectively mining data from git repositories, to satisfy different information needs and support several advanced analytics.

In this context, *Developer Social Networks* (DSNs) have recently emerged as an effective tool for the analysis of community structures and collaborations among developers in software projects and software ecosystems [87]. A DSN can be considered a Social Network in all respects and all the potentialities, and *Social Network Analysis* (SNA) facilities can be used to infer useful knowledge from these environments for different aims. The classical SNA techniques can be easily adopted on these information networks – usually modeled as graphs – to support a plethora of interesting tasks (leveraging the related topological properties, or based on statistic or bio-inspired approaches): *community detection* to find the community of developers that share interests in similar projects, *influence analysis* to discover the most important developers within a given community, *team formation* to detect the group of developers w.r.t. a given technology within several communities in order to start more quickly a new project in according to agile paradigm, *recommendation* to automatically suggest the most appropriate developers to solve an issue, etc. Thus, community detection in DSNs represents a very challenging issue and the first preliminary step in order to realize the other SNA tasks. At the best of our knowledge, most approaches proposed in the literature for community detection in DSNs employ *homogeneous* graphs that are characterized by single relevant entities as nodes (i.e., developers) and by a single type of edge and they are often used to model social networks. One example is represented by the homogeneous graph exploited in [90] that includes a single type of node (representing the developers) and a single type of edge (representing the *commit* relationship). In turn, nodes of an *heterogeneous graph* can have different types (i.e., developers, repositories, issues)

---

<sup>1</sup><https://github.com>

---

and be connected by different types of edges. Due to the multitude of data types represented, they turn out to have a more complex topological structure, which makes them both more information-rich, but also computationally more onerous. However, according to the most recent literature [96, 210, 202]) *heterogeneous graphs* seem to be the best candidates for capturing and managing the intrinsic complexity and wide range of relations that can be established among modern information networks such as DSNs.

Unfortunately, DSN-related graphs can reach enormous sizes and thus *graph-embedding* techniques may support in a more efficient manner SNA tasks [208, 203]. In particular, thanks to graph embedding, community detection problems can be effectively faced by leveraging classical clustering algorithms.

In this paper, we decided to explore the possibility of modeling the interactions that developers have within a project repository by a heterogeneous graph and to use graph embedding and community detection techniques on the embedded graph to find relevant types of communities in the global software engineering world. To this aim, we defined a novel DSN model that extracts relevant entities from the GitHub website and relevant relationships that occur among these entities. Therefore, we designed a framework that supports the population of the DSN and implements the analysis tasks by exploiting graph embedding techniques. Actually, we are interested in *non-overlapped* communities because we are focused on identifying developers who can work on the same project on the basis of similar experiences and interests and not on the basis of possessing common skills that can obviously be shared within different projects. To validate the proposed approach, we carried out an experiment where we studied the effectiveness and the performance of the proposed community detection techniques by considering different instances of DSNs and different types of graph embedding techniques.

The remainder of the paper is structured as follows. Section 3.2 reports the Related Work. Section 3.3 illustrates the framework we defined to instantiate a DSN and to implement the community detection task, relying on the social model of GitHub. Section 3.4 shows the experiment we performed to validate the proposed approach with the related results while Section 3.5 eventually reports conclusive remarks and future work.

---

## 3.2 Related Work

In this subsection, the state of the art of the literature regarding existing GitHub datasets, approaches to model Developer Social Networks, and proposed solutions for the problem of Community Detection in the context of Developer Social Networks will be presented.

### 3.2.1 GitHub Information Models

Several works in the literature have proposed information meta-models and data models for information mining from platforms such as Github.

GHTorrent represents the Github dataset that has been most frequently used by the scientific community. It was originally proposed in 2012 by [78, 77]. They extracted a dataset from GitHub making it available both in form of csv files and as a relational database. GHTorrent contains a broad spectrum of information about Developers and Repositories, including information about commits, issues and pull requests. This dataset and its subsequent updates have been the basis for many scientific works. In particular, the use of GHTorrent has been promoted by the community at the Mining Challenge at MSR 2014<sup>2</sup> from which many examples of works based on this dataset were presented (e.g., [151, 166, 119]). Unfortunately, the execution of community detection algorithms on very large relational databases or csv files is very onerous, thus different database models should be considered.

Software Heritage<sup>3</sup> is a graph database including information extracted both from Github and from other similar infrastructures such as Gitlab and Bitbucket. Software Heritage provides a web interface and a set of APIs to browse and queries its data. Software Heritage has been used as a dataset by many recent works, thanks to its efficiency and scalability (e.g., [30, 34, 146]). Software Heritage's information meta-model focuses on the history of projects making it suitable for studies about their temporal evolution.

Many other studies in the scientific literature proposed graph databases to represent the information extracted from GitHub. For example, in [112] a graph database was proposed to model Developers, Repositories and

---

<sup>2</sup><http://2014.msrconf.org/challenge.php>

<sup>3</sup><https://www.softwareheritage.org/>

---



a limited set of relationships including Commits, Comments and Watch (between a Developer and a Repository) and Forks (between Repositories). Other examples of graph based models of GitHub information have been successively proposed [41], [91], [109], [76], [212], [139], [160], [179, 183], [60]. Unfortunately, most of these graph databases are not updated or are not more available, thus in this work we propose another model including a subset of information allowing the execution of effective and efficient Community Detection algorithms.

### 3.2.2 Developer Social Networks

A large set of GitHub mining activities aim at studying developers and the relationships between them.

Many studies are focused on aspects related to the developers popularity (e.g., [27, 90, 95, 213, 55, 117, 97]). Another problem often approached by studies based on GitHub information is the Expert Finding, in which techniques for the effective search of developers with specified skills are based on the analysis of their activity on GitHub (e.g., [27, 90, 95, 213, 55, 117, 97]). The characterization of working groups of Developers sharing the participation to the same GitHub repositories is the object of several other relevant works (e.g., [23, 209, 147, 204, 119]).

The studies that are most similar to ours are those related to the modeling of Developer Social Networks and the consequent detection of community of developers. Recently, the systematic study by [87] has classified 255 scientific papers in the broader field of Developer Social Networks. In most of them, these networks have been modeled as oriented graphs in which the nodes represent the developers and the edges represent the relationships between them that can be inferred by the analysis of infrastructures such as GitHub. In particular, the authors highlight that in this specific area, there is a lack of shared datasets and that most of the works were based on the analysis of few projects (only 50 studies analyzed at least 100 projects)

### 3.2.3 Community Detection on Developer Social Networks

The specific task of Community Detection in the context of GitHub developers has been faced by several recent works in the literature. In all

---

these works a graph is built on the basis of the information extracted from GitHub, in which the nodes represent the developers and the edges between two developers represent the existence of shared repositories to which both developers have contributed in terms of commits (e.g., in [27] and [83]). In addition, some other information such as the comments to the same issues or pull requests are considered in [158] and [201]. All these works apply different metrics to weigh the edges representing the collaboration between two developers and use novel or well-known clustering algorithms to detect Developer Communities. Unfortunately, most of these works remain at a proposal level, without a wide experimentation or available material for replication purposes, thus they are not able to provide evidence of the existence of a better performing weighting model.

The work that is most similar to ours and that provides more details about its replication is the one presented in 2021 by [90]. They propose and validate Community Detection techniques applied to a graph whose nodes are represented by the developers and whose edges represent the existence of a shared repository on which both the developers commit changes. The edges are weighted on the basis of a metric called Developer Intensity Cooperation that takes into account the quantity of commits of the two Developers on shared repositories. Different clustering algorithms were proposed and compared in order to detect communities of developers. The cohesion of such clustering was evaluated by means of topological metrics. The authors validated their approach on the basis of information extracted from GitHub regarding the activity of several thousand of Developers over a two-year period, between 2017 and 2019 and on popular projects (characterized by a large quantity of 'stars' given by GitHub users)

### 3.3 Framework

Our framework aims to support community detection task within a DSN environment, which has been treated by three main phases: *Data Ingestion*, *Data Modeling* and *Community Detection*. In the first phase, we crawl data from a collaborative software repository. Successively, we process the extracted data to build the DSN graph structure, where the nodes consist of developers, repositories and issues connected by different kinds of edges. The last phase involves the community discovery process,

---

where we first embed the DSN in a lower dimensional space and then divide the multidimensional points into a set of clusters (communities). Figure 3.1 provides an overview of the three phases.

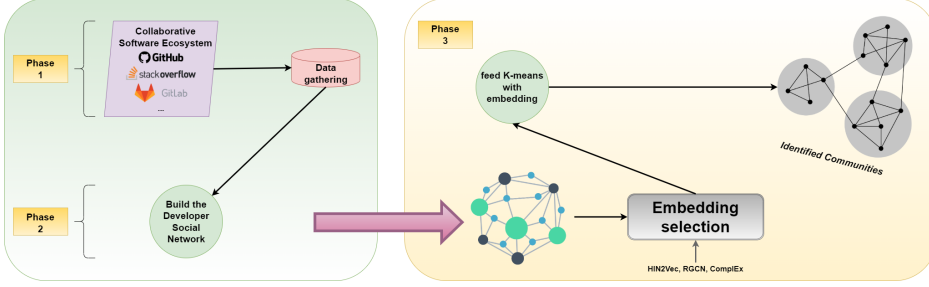


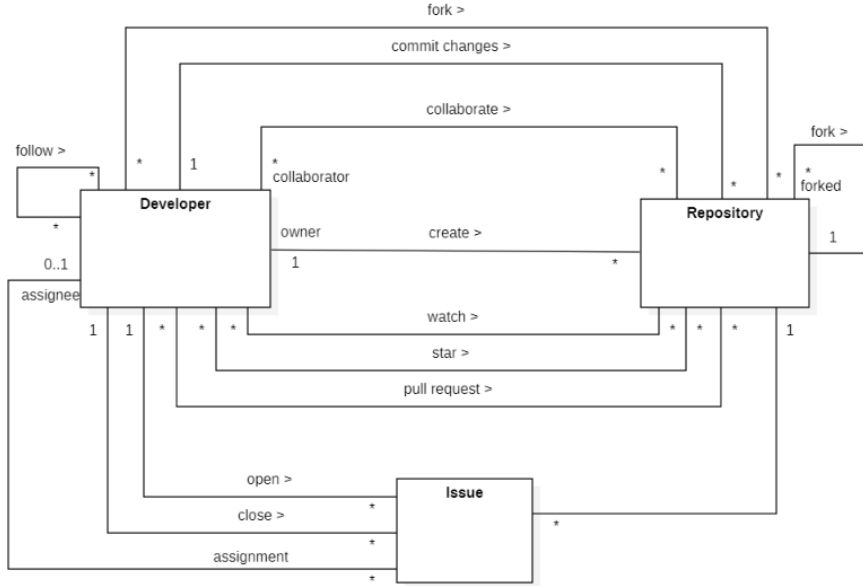
Figure 3.1. Framework overview

### 3.3.1 Task Definition

Understanding user relationships within a developer social networks can be useful for different analytic tasks (e.g., team formation or expert finding). Despite different effort in finding overlapped communities of developers sharing common skills, we are more interested in analyzing the problem of identifying developers who can work on the same project on the basis of similar interests, that is typically addressed as a non-overlapped task (see [125, 124, 90]). For this reason, we are interested in identifying developers' communities, composed of users sharing the same interests in terms of similar projects and/or applications. In more clear terms, the goal of our task is to identify a sub-set of developers in the DSN who share the same interests in terms of their interactions on given repositories (commits, stars, forks, etc.).

**Definition 1 (Task Definition)** *Our task can be modeled as a particular function  $f$  associating to each developer  $d_i$  belonging to the developers' set  $D$  ( $d_i \in D$ ) one of the  $K$  communities  $C_j \in \mathcal{P}(D)$  that can be seen as non overlapped subsets of  $D$  ( $\cup_{j=1..K} C_j = D \wedge \nexists z \neq j : C_j \cap C_z \neq \emptyset$ ):*

$$f : d_i \rightarrow C_j \quad (3.1)$$



**Figure 3.2.** Conceptual Model of a Collaborator Software Repository

The number of obtained communities depends on the number of developers and on the adopted features for their characterization.

### 3.3.2 Modeling GitHub as a DSN

Collaborative software repositories contain heterogeneous information about developers, repositories and issues, which, in our opinion, can be summarized as a list of actions that a developer can perform on an object (issue or repositories).

Figure 3.2 shows a conceptual model describing collaborative software repositories (e.g., Github) and the main interactions in that context. The model is depicted as a UML class diagram having three classes, namely *Developers*, *Repositories*, and *Issues*, and 13 associations among them.

*Developers* are registered users of a collaborative software repository system, who can create and become *owners* of new *Repositories*, i.e. collection of source code and artifacts about a given project. A new repository

can also be created by means of a *Fork* operation on an existing one, creating a clone of the original repository owned by another developer. The owner of a repository can invite and qualify other developers as *Collaborators*. A collaborator may manage the repository with almost the same rights as the owner. In particular, both the owner and every collaborator can *commit changes*, i.e. submit updates of any repository files. On the other hand, a developer, even if not directly involved in a project, may propose a contribution (for example the implementation of a new feature or the correction of a bug) via a *Pull Request*: the owner of the repository and his collaborators can approve it and apply these changes or refuse it. When a developer wants to express his interest in all the activities of another Developer he can decide to *Follow* him: in this case, he will receive notifications of all the public activities made by the developer on Github repositories. This is a common feature of many social networks and represents the unique direct connection between the two developers. If a developer is interested in the evolution of a repository (owned by another developer), he can *Watch* it: in this case, he will receive notifications of the changes of the project on his dashboard. A simpler way a developer has to express his appreciation for an existing Repository is by flagging it with a *Star*, which is a mechanism equivalent to those of many social networks (i.e. "like").

Another common interaction to be considered regards *Issues*. An issue is a message written by a developer (involved or not in the project) used to communicate the existence of a problem or a bug in a project, or the request for a new functionality, or simply a comment about the project. Unlike pull requests, no changes are associated to Issues. An issue has its own life cycle, which starts with its opening by a Developer and continues with a possible dialogue between this Developer, the Repository owner and his collaborators and possibly other external developers. During these interactions, the issue can be approved by Owners and *Assigned* to one of the developers involved in the project for its resolution. An issue may be *Closed* when it is considered resolved or not relevant.

Several meta-models have been proposed with the goal of handling the different types of entities and relationships within a collaborative software repository (for example GHTorrent [77, 78] and Software Heritage<sup>4</sup>),

---

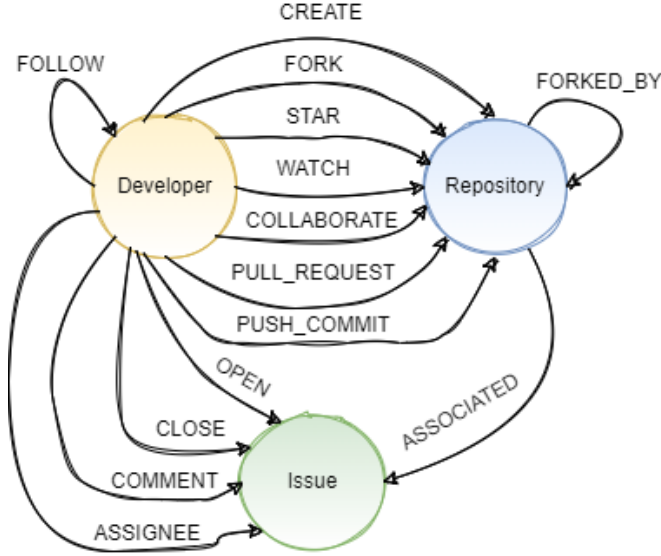
<sup>4</sup><https://www.softwareheritage.org/>

but without focusing on the relevance of the examined relationships for a specific analysis task. On the other hand, other meta-models are oversimplified or target-focused on specific aspects of open source repositories, such as the commit actions of developers on the same repositories (see [90] for more details).

Our data model shown in Figure 3.2 tries to capture and handle all the most useful information from a collaborative software repository for DSN analytics purposes. Our model considers both information demonstrating direct relationships between two developers (e.g., the follow and the collaboration relationships) and indirect relationships between developers (e.g., both participation in the activities on the same Repository) and demonstrates how they can be very useful for DSN analysis. As delineated in Section 3.1, the proposed information model relies on a heterogeneous graph structure that accommodates diverse entities, such as developers, issues, and repositories, as nodes, and allows the existence of multimodal relationships within a Developer Social Network (DSN). It is noteworthy that the suggested information model differs from the one proposed by [90], which employs a homogeneous graph structure using only developer nodes and a single edge type, i.e., collaboration intensity, to summarize information about the joint commits performed by different developers on the same repositories. To exploit the rich semantics present within our proposed information models based on heterogeneous networks, graph embedding techniques were necessary to handle analytical tasks on high-dimensional graphs [210]. We propose to leverage data detailed in Figure 3.2 by using a heterogeneous information network (our DSN). The aim is to fuse multi-typed interacting components to improve semantics related to nodes and links in order to group together developers who have shown interest in the same repositories and/or applications (*Community detection* task).

**Definition 2 (Developer Social Network)** *Let  $D$ ,  $I$  and  $R$  be respectively the sets of developers, issues and repositories, we define the Developer Social Network as an oriented graph  $G = (V, E)$ , where  $V$  is the heterogeneous set of vertices ( $V = D \cup I \cup R$ ) and  $E$  is the edge set that can be established between two nodes.*

Figure 3.3 shows at a glance our idea of DSN, which is composed by three entities: i) *Repositories*, ii) *Developers*, iii) *Issues*. The Figure also



**Figure 3.3.** Developer Social Network Model

summarizes the different types of actions or relations among developers, repositories, and issues, that we will rely on in the considered community detection tasks.

### 3.3.3 Community Detection process

On the basis of the introduced DSN, our goal is to detect the set of communities whose members are developers showing some common activities on several repositories.

Despite different community detection approaches have been proposed in the literature on heterogeneous graphs, and in particular, on those supporting DSN analysis from a social perspective, the majority of them suffers of computational and memory costs due to the sizes that such graph-based structures can reach [71].

*Representation learning* involves automatically finding and learning a latent representation of data that can be used as input features for supervised machine learning algorithms for various prediction tasks [29]. In particular, *Graph embedding* aims to represent a graph in a low dimen-

sion space, while preserving network topology, thus facing with the graph large size challenge. Therefore, we propose a community detection model, whose aim is to unveil developers' community by relying on *Graph embedding* techniques for representing members of a DSN in a low-dimensional spatial representation that preserves the network topology, satisfying the 2<sup>nd</sup> order of proximity. Thus, we performed an embedding of the DSN to a  $n$ -dimensional space through graph embedding techniques, whose points are clustered into a set of communities by using unsupervised machine learning algorithm with the aim to optimize the number of clusters. In particular, our aim is to use an algorithm, satisfying fast convergence, scalability and fit for scattered data.

In Algorithm 1 we present our community detection algorithm based on graph embedding (*BERTO*). The input of our algorithm are: i) the selection criterion used to select a subset of the developers and repositories on GitHub, ii) the chosen graph embedding technique and iii) the clustering algorithm. The output of *BERTO* will be  $\mathcal{C} = \{C_1, \dots, C_n\}$  the set of the discovered communities.

The *BERTO* algorithm is given as follows: i) a subset of relationships between Developers, Issues and Repositories are extracted from Github according to a given Selection Criterion (*SC*), ii) while there are relationships to analyze iii) an element  $r$  is fetched from the relationships set  $R$ , and iv) it is added to the DSN. v) The whole DSN is given as input to the chosen embedding *GE* technique to compute the nodes embedding that will feed vi) the clustering Technique *CT* responsible for providing in output the identified  $K$  communities. The Selection Criterion *SC* is used for choosing and categorizing the data to be retrieved from GitHub in terms of: programming language, the time frame concerning the repository creation date, and the number of stars assigned to the repositories. There are several factors that can influence the choice of clustering techniques, such as the type of data to be analyzed, available resources, and time constraints. In the case of the *BERTO* algorithm, we chose  $K$ -Means as our clustering technique because of its time-efficiency and manageable hyperparameter tuning phase, which are important for practicality and efficiency in real-world scenarios [122, 48].  $K$ -Means has a time complexity of  $O(n)$ , which is superior to other partition clustering algorithms such as *CLARANS* and *PAM*. It also outperforms clustering algorithms of other categories such



as density-based algorithms like DBSCAN and OPTICS, and hierarchical ones like Chameleon, in terms of time complexity. However, K-Means' scalability is limited as the dimensionality of the input data increases, as observed in [157]. To address this issue in the BERTO algorithm, we use a graph-embedding stage that reduces the dimensionality of the input data. Another advantage of K-Means is that it only requires the identification of the optimal number of clusters ( $K$ ), whereas other algorithms such as DBSCAN require tuning of other parameters for optimization. This tuning process can be more complex than the one used for K-Means, which can simply use the elbow rule to identify the optimal value of  $K$ .

The computational complexity achieved by our proposed algorithm *BERTO* is  $O(n \log n)$ , given from a cost of i)  $O(n)$  to explore all the relationships set ( $n = |R|$ ), ii)  $O(n)$  for the graph embedding and clustering phase, iii) and  $O(\log n)$  for the graph updating phase.

---

**Algorithm 1** *emBEDding gRaph communiTy detectiOn (BERTO)*-algorithm

---

```

1: procedure  EMBEDDING  GRAPH  COMMUNITY  DETECTION  (BERTO)-
   ALGORITHM(SC,GE,CT,K)
2:   – Input: SC (Selection Criteria)
3:   – Input: GE (Graph embedding)
4:   – Input: CT (Clustering Technique)
5:   – Input: K (Number of Clusters)
6:   – Temporary: R (Relationships Set)
7:   – Temporary: DSN (Developer Social Network)
8:   – Temporary:  $V_e$  (List of node embeddings)
9:   – Output:  $C$  (List of identified community)
10:   $R \leftarrow \text{QUERY\_GITHUB}(SC)$ 
11:  while ( $R \neq \emptyset$ ) do
12:     $r \leftarrow \text{dequeue}(R)$ 
13:     $DSN \leftarrow \text{UPDATE\_GRAPH}(DSN, r)$  ▷ DSN building
14:  end while
15:   $V_e \leftarrow \text{EMBEDDING\_GRAPH}(DSN, GE)$  ▷ Nodes list  $V_e$  with graph
   embedding  $GE$  on the DSN
16:   $C \leftarrow \text{IDENTIFICATION\_COMMUNITY}(V_e, CT)$ 
17:  return  $C$  ▷ List of identified communities
18: end procedure

```

---

### 3.3.4 Running Example

In this section, we describe a running example of how it is possible to generate a DSN from GitHub data and how it is possible to detect communities of developers by embedding the DSN graph. We suppose to rely on the following information:

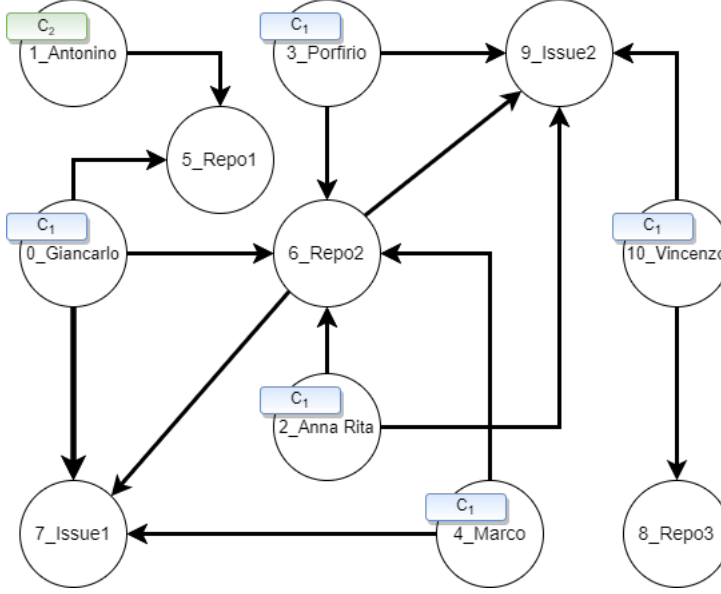
1. (*commit*, *Giancarlo*, *Repo1*)
2. (*commit*, *Antonino*, *Repo1*)
3. (*open\_by*, *AnnaRita*, *Repo2*)
4. (*pull\_request*, *Porfirio*, *Repo2*)
5. (*comment*, *Marco*, *Issue1*)
6. (*watch*, *Giancarlo*, *Repo2*)
7. (*comment*, *Giancarlo*, *Issue1*)
8. (*star*, *Marco*, *Repo2*)
9. (*comment*, *Porfirio*, *Issue2*)
10. (*open\_by*, *Vincenzo*, *Issue2*)
11. (*comment*, *AnnaRita*, *Issue2*)
12. (*create*, *Vincenzo*, *Repo3*)

Each operation is characterized by: i) a relationship (a commit, a creation of repositories and so on) ii) a source and destination object.

The obtained DSN, shown in Figure 3.4, is then fed as input to the embedding algorithm (in this example *Complex*), which provides an embedding representation of each developer node within our network. Successively, an euclidean distance between two nodes has been applied in order to build the distance matrix, shown in Table 3.1.

Finally, the vectors obtained from the embedding step are given as input to the K-Means clustering algorithm. Choosing  $K = 2$ , the community partition obtained is:

---



**Figure 3.4.** DSN representation for the Running Example. The heterogeneous graph is composed by 11 nodes and 12 edges.

$$C_1 = \{AnnaRita, Giancarlo, Marco, Porfirio, Vincenzo\}$$

$$C_2 = \{Antonino\}$$

We expected the obtained clusters because we can see from Example 1 how the developer node *Antonino* has only a unique connection to a single repository and in common with only one user, namely *Giancarlo*. In turn, the other users show more collaboration activities, as example, developer *Vincenzo* despite being connected to only two entities (*Issue2* e *Repo3*), the issue he worked on is a node of strong interest for the other developers as well, which makes *Vincenzo* also strongly connected with the other developers who worked on the same issue. In conclusion, it can be interpreted that *AnnaRita*, *Giancarlo*, *Marco*, *Porfirio* and *Vincenzo* are probably interested in the same projects, or are close collaborators (as, for example, can be inferred from their interactions on *Repo2*), while *Antonino* has no close ties with any of them; in fact, only *Giancarlo* has interaction with *Antonino* on a marginal project *Repo1*.

Table 3.1. Distance matrix

	Anna Rita	Antonino	Giancarlo	Marco	Porfirio	Vincenzo
Anna Rita	0	0.7	0.3	0.2	0.2	0.4
Antonino	0.7	0	0.6	0.6	0.7	0.8
Giancarlo	0.3	0.6	0	0.3	0.2	0.5
Marco	0.2	0.6	0.3	0	0.3	0.5
Porfirio	0.2	0.7	0.2	0.3	0	0.4
Vincenzo	0.4	0.8	0.5	0.5	0.4	0

## 3.4 Experimental analysis

### 3.4.1 Goals

The objective of this experimentation is the evaluation of the proposed Community Detection algorithm *BERTO*. We want to evaluate and compare the performance of the technique obtained by applying different graph embedding techniques and find the best-performing one. In addition, we want to compare the performance obtained by adopting the proposed information model with respect to the ones obtained by adopting the model proposed for the *ABDCI algorithm* of [90]. Our information model is based on a heterogeneous graph that takes into account not only developers but also other entities such as issues and repositories, along with their diverse types of relationships, as shown in Figure 3.2. In contrast, [90] used a homogeneous graph, defined as a *Software Ecosystem Network* (SEN) by the authors, that only represents developers connected by a single type of edge. The aim of our experiment is to determine whether the use of a heterogeneous graph, can facilitate a more distinct identification of communities compared to a homogeneous graph. Additionally, we aim to compare the time complexity of the two approaches. Furthermore, the community detection algorithm (ABDCI) [90] is based on hierarchical clustering and achieves a time complexity of  $O(n^2)$ , which is higher respect to the obtained by our proposed approach (BERTO) based on K-Means and equal to  $O(n \log n)$ . In order to assess the performance of the technique, we evaluate the quality of the detected Developers Communities, using a modularity metric to reward clustering that reveals cohesive communities of developers. Furthermore, we intent to evaluate the robustness of

the proposed technique as different samples of developers and repositories from GitHub are considered, having different sample sizes and different repository selection criteria.

Finally, we want to study the computational cost and time needed for the execution of the community detection algorithm on different information samples.

### 3.4.2 Research Questions

In order to pursue the presented goals, we have formulated four research questions to be answered by the experimentation.

- RQ1 How does the modularity of the Developers Communities produced by the *BERTO* algorithm vary depending on the adopted graph embedding technique?
- RQ2 How does the modularity of the Developers Communities produced by the *BERTO* algorithm vary with different information models?
- RQ3 How does the modularity of the Developers Communities produced by the *BERTO* algorithm vary for different Github repositories samples?
- RQ4 How does the execution time needed to evaluate the Developers Communities produced by the *BERTO* algorithm vary for different Github repositories samples?

### 3.4.3 Variables and Metrics

#### Independent Variables

Different sets of independent variables have been considered in the different phases of the experiments we carried out, i.e. the *graph embedding technique*, the *information model*, the *repository sample type*, the *repository sample size* and the *embedding and cluster size*. Comprehensive information regarding the aforementioned variables can be found in Table 3.3.

---

In the context of the research question RQ1, we considered three categories of *graph embedding technique*, based on how they capture the network topology [40] and used three different algorithms as representatives of these categories: i) Proximity Preservation methods, ii) Message-Passing methods, iii) Relation Learning methods). Specifically, an embedding algorithm was selected for each category, *Hin2Vec*, *RGCN* and *ComplEx*, respectively.

1. *HIN2Vec*: a proximity preservation method based on random walk and meta-path. Taking as input a Heterogeneous Information Network (HIN) and a set of meta-paths, the HIN2Vec framework performs multiple prediction training tasks, that are jointly based on the targeted set of relationships taken as input [72].
2. *RGCN*: based on message passing, it aims to learn node embedding by aggregating the information from the neighborhood. RGCN differs from standard link predictors because it employs node neighborhood information to learn the vector representations of the node/-graph [159, 186].
3. *ComplEx*: a relation learning method, that considers complex-valued embeddings. The use of complex embedding allows for binary relationship management, both symmetric and asymmetric. Link prediction can be solved as a binary tensor completion problem, where each slice of the tensor is an adjacency matrix of the relationships that are present in the graph [190, 189].

In the context of the research question RQ2, we considered two different *information models*, i.e. the one presented in Section 3.3 w.r.t. the one built in [89] that considered only the *joint commit* interactions. In that work, Hou et al. presented the ABDCI algorithm, which combines developer interaction information and network topology information and defined the intensity of collaboration in order to build a DSN. They assumed that developers executing commits into the same repository always have similar skills and preferences and should be in the same cluster [90]. The computational complexity of the algorithm *ABDCI* is  $o(n^2)$  which is higher than the complexity achieved by our proposed model *BERTO*.

In addition, we have varied two parameters of the *BERTO* algorithm in order to study the parameter values combinations providing the better clustering performance, i.e. the *embedding size* and the *cluster size*.

The embedding size is a critical decision: the key factors for the optimal selection are mainly related to the availability of computational resources and the choice of a trade-off value that offers complexity reduction and no information losses, so the embedding size will determine the level of information compression. A larger size offers a model with high information content, at the expense of high complexity, while a smaller size will offer higher computational performance but losing the heterogeneous information in the model. The performance of the clustering is closely related to the choice of the cluster size value  $K$ . It is important to consider values that are reasonably large, in order to reflect the characteristics of the dataset, but at the same time significantly small compared to the number of objects to be analyzed, which is why we desire clustering of the data [144].

### Dependent Variables

We have considered two dependent variables, i.e. the *modularity* of the detected developer communities and the *time* needed to execute the *BERTO* algorithm.

To evaluate how good is the quality of the division of the networks into communities, we use a quantitative and topological measure called *Modularity* [138], whose score value is an indicator of the quality of the clustering (high values corresponds to better clusterings).

Modularity  $Q$  can be defined according to equation 3.2, where  $A_{ij}$  is a single element of the adjacency matrix  $A$ . In particular,  $A_{ij} = 1$  if there is an edge that connects the nodes  $i$  and  $j$ ,  $A_{ij} = 0$  otherwise.  $g_i$  is the number of the community which node  $i$  belongs to, and  $\delta(i, j) = 1$  only if  $i = j$ , that means that only if the two nodes belong to the same community  $\delta(g_i, g_j) = 1$ .  $k_i$  indicates the degree of the node  $i$ . Lastly  $m = \frac{1}{2} \sum_{ij} A_{ij}$  is the total number of edges in the graph.

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2m}) \delta(g_i, g_j) \quad (3.2)$$

Modularity turns out to be one of the most widely used metrics in op-

timization methods to detect communities in networks [90, 127]. Its value may vary between -1 and 1: a higher value indicates a strong community structure (for more details see the paper [43]).

The recorded performances are evaluated by the aggregated execution times inherent to the embedding phase and the community identification phase.

#### 3.4.4 Objects

The objects of the experiments consisted of selected subsets of repositories from GitHub, which were analyzed in order to extract the information to be represented by graphs. In order to obtain these graphs, we have performed different queries on GitHub adopting different filtering criteria. In particular, we have varied the values of three query parameters, i.e. *the number of stars* received by the repository (that could be an indicator of the general appreciation of the corresponding project), the *creation date* of the repository (that has been used to restrict the search to a set of projects created within a specific time window) , and the *programming framework* of the source code stored in the repository.

In detail, we focused on the repositories that have at least 30 stars, were created between January 1, 2020 and January 20, 2020 and have been labeled as *Python*, *Java* or *Android* projects. Once we retrieved the repositories with these characteristics, we captured for each of them all the information related to owners, collaborators, forks, issues and any other relationships of the data model presented in Section 3.3. The decision to start our search from repositories with a certain number of stars is because we wanted to consider popular repositories that arouse a high interest from GitHub users. It is important to note that, unlike Python and Java, the "Android" category is a set of technologies (e.g., Java, XML, Kotlin, HTML5), thus we manually labeled the repositories devoted to the development of applications for the Android ecosystem.

In the context of RQ1 and RQ2 we have considered the graph built on the *Full* subset of repositories (which includes all three subsets of repositories), whereas in the context of RQ3 and RQ4, we have considered four different graphs, respectively based on the *Python*, *Java*, *Android* and *Full* subsets of repositories.

Table 3.2 reports the main characteristics of these four subsets of ex-

---



**Table 3.2.** Dataset characterization

	<b>Android</b>	<b>Java</b>	<b>Python</b>	<b>Full</b>
<b>Number of Repositories</b>	8,883	7,881	18,570	35,334
<b>Number of Developers</b>	56,231	30,262	81,612	168,105
<b>Number of Issues</b>	4,310	3,520	11,310	19,140
<b>Number of Relationships</b>	961,433	155,389	1,128,458	2,245,280

perimental objects, i.e. the number of repositories in each set, the total number of involved developers, the total number of found issues, and the total number of relationships.

### 3.4.5 Design of Experiments

In order to answer each of the four proposed research questions, an experiment organized in four phases was carried out, where each phase is characterized by different sets of objects, independent and dependent variables and factors.

The selection of graph embedding algorithms for the experiment was guided by the work of [40] which classifies embedding algorithms into three categories based on their method of capturing network topology: i) Proximity Preservation, ii) Message Passing, and iii) Relationship Learning. For each category, we selected one embedding algorithm, respectively Hin2Vec, RGCN and ComplEx. As demonstrated in the response to RQ1, we found that ComplEx outperforms the other algorithms and thus was chosen as the embedding algorithm for our experimentation.

In the first phase, to answer RQ1, the sample of repositories called *Full* and the heterogeneous graph based on the information model presented in Section 3.3 were considered. We evaluated *modularity* (dependent variable) values obtained by considering three different graph embedding techniques (independent variable): *Hin2Vec*, *RGCN*, *ComplEx*. For this purpose, the *BERTO* algorithm was executed by varying two factors, i.e. the embedding size and the cluster size. In detail, for the embedding size the values of 50, 150 and 300 were considered, while for the cluster size  $K$  the values of 50, 150 and 250 were considered in order to evaluate which of these combinations produced the best modularity values.

In the second phase of the experiment aiming at answering RQ2, we considered the same sample of repositories (*Full*) but we varied two information models as independent variables: the one that we have proposed in Section 3.3 (BERTO) and the one considering commits on shared repositories as the only relationship between developers, coherently with the *ABDCI algorithm* proposed by [90]. The *modularity* of the detected community was considered as dependent variable, and it was evaluated by also varying in this case the embedding size and the cluster size using the same set of values considered in the first phase of the experiment.

In order to answer RQ3, we considered different samples of repositories beyond the *Full* one. The other three considered samples are the ones called *Java*, *Python* and *Android*. In this phase, we considered only the heterogeneous graphs based on the information model presented in Section 3.3. The dependent variable in this phase is still the *modularity*, and the variable factors are still the embedding size and the cluster size.

Finally, in the fourth phase of the experiment, we have considered the same objects involved in the third phase but we have considered a different dependent variable, i.e. the *execution time*. The execution times were obtained via the Python library "time". This time was measured when we executed our technique in a cloud-based execution environment built on Google Colab<sup>5</sup>, involving a Python 3.6 implementation of the Community Detection algorithm *BERTO* and an execution engine composed by 2 CPU (2.2GHz), 13 GB of RAM. The *NetworkX* and *SciKit-Learn* libraries were used for analyzing graphs and using unsupervised learning algorithms. Also in this phase, we have considered embedding size and cluster size as factors to be varied, but we have considered larger samples of possible values: the clustering size varied between 50 and 500, while the cluster size varied between 50 and 300.

The characteristics of the four phases of the experiment are summarized in Table 3.3.

### 3.4.6 Results

In this Section, we report the results we obtained in each phase of the experiment and the corresponding answers to each considered RQ:

---

<sup>5</sup><https://colab.research.google.com/>

**Table 3.3.** Design of Experiments: Phase 1 involves the determination of the most effective embedding technique in terms of modularity, from among the three options available. This process is carried out for each of the three embedding techniques. Phase 2 involves fixing the embedding technique selected in Phase 1 and considering the information model as the independent variable. Phase 3 involves changing the analyzed sample while keeping the information model and clustering technique constant, and observing the variations in modularity that occur across different values of cluster and embedding size. Phase 4, the execution time is established as the dependent variable, and the impact of sample size and clustering and embedding dimensions is evaluated.

	Phase 1	Phase 2	Phase 3	Phase 4
Research Question	RQ1	RQ2	RQ3	RQ4
Objects	Full	Full	Full, Java, Python, Android	Full, Java, Python, Android
Independent Variables	Graph Embedding technique (Hin2Vec, RGCN, ComplEx)	Information Model (Berto, ABCDI Model)	-	-
Fixed Factors	Information Model (Berto)	Graph Embedding technique (ComplEx)	Information Model (Berto), Graph Embedding technique (ComplEx)	Information Model (Berto), Graph Embedding technique (ComplEx)
Varying Factors	Embedding size (50, 150, 300), Cluster size (50, 150, 250)	Embedding size (50, 150, 300), Cluster size (50, 150, 250)	Embedding size (50, 150, 300), Cluster size (50, 150, 250)	Embedding size (between 50 and 500), Cluster size (between 50 and 300)
Dependent Variables	Modularity	Modularity	Modularity	Execution Time

## RQ1

The execution of the BERTO algorithm for the three adopted graph embedding techniques and for different values of embedding size and K produced the results shown in Table 3.4. Due to computational limitations we were unable to compute the embedding vectors for RGCN with embedding size = 300. We can observe that the adoption of the ComplEx algorithm produced the greater values of modularity w.r.t. the ones obtained with the other techniques, for any value of embedding size and K, thus we can affirm that the ComplEx techniques appear the most suitable to this problem.

The poor results in terms of modularity obtained by HIN2Vec [72] can be attributed to the fact that this learning framework is based on meta-paths and thus is unable to obtain a latent representation of nodes that reflects the network topology.

In the RGCN [159] framework, the vector representation of neighboring nodes is collected and then transformed for each type of relationship separately. It turns out that the calculation of node embedding is done using only one vector from the neighborhood. Therefore, RGCN is unable to compute a representation of the network topology in the embedding

**Table 3.4.** (RQ1) - Results of *BERTO* algorithm in terms of modularity by varying technique and embedding size

<i>BERTO</i> Algorithm Modularity								
<b>K</b>	emb 50			emb 150			emb 300	
	<b>Complex</b>	<b>RGCN</b>	<b>HIN2Vec</b>	<b>Complex</b>	<b>RGCN</b>	<b>HIN2Vec</b>	<b>Complex</b>	<b>RGCN</b>
50	77,75%	4,21%	2,71%	82,64%	2,11%	2,32%	78,57%	N/A
150	67,44%	2,91%	0,82%	77,19%	1,41%	1,16%	72,49%	N/A
250	55,18%	2,17%	0,73%	68,23%	1,47%	0,91%	64,55%	N/A

**Table 3.5.** (RQ2) - Comparison of the average modularity, varying  $K$ , for *BERTO* and *ABDCI* [90]

Modularity Comparison						
<b>K</b>	emb 50		emb 150		emb 300	
	<i>BERTO</i>	<i>ABDCI</i>	<i>BERTO</i>	<i>ABDCI</i>	<i>BERTO</i>	<i>ABDCI</i>
50	77,75%	68,35%	82,64%	86,34%	78,57%	82,31%
150	67,44%	35,83%	77,19%	80,74%	72,49%	76,90%
250	55,18%	25,27%	68,23%	73,49%	64,55%	69,86%

space.

This type of issue has not shown up with Complex [190], which uses complex valued embedding. In particular, when there is only one kind of relation between entities Complex uses the real part of low-rank normal matrices to represent the relationship, allowing the framework to capture the topology of the networking inside the computed embedding vector of the nodes.

## RQ2

Answering RQ1 we concluded that ComplEx is the best performing embedding technique among the three ones we proposed. To answer RQ2 we executed *BERTO* algorithm with ComplEx embedding technique for the two different information models and for different values of embedding size and  $K$ . Table 3.5 reports the obtained *modularity values* after applying *BERTO* to our model where we consider all types of interaction, and the *modularity values* obtained using the *ABDCI algorithm* proposed by [90], which is only based on shared commits. Our model does not seem to give us a better result in terms of modularity. We therefore investigated how developers are distributed across communities for the two techniques we are comparing. Based on 30 experiments, in Table 3.6 we show median

and standard deviation of the node distribution inside the communities/-clusters. Focusing on the standard deviation, we can see that our model has smaller values than the model proposed by [90]. This means that the *BERTO* algorithm applied to our information model, despite not always obtaining better modularity values than the *ABDCI algorithm*, is able to find a more balanced community division, i.e. a more equal distribution of nodes within the various communities.

High standard deviation values indicate that the distribution is unbalanced, meaning that there will be few clusters, or in the worst case only one cluster, with most of the nodes inside, whereas all the other clusters will contain few nodes. Having all developer nodes within the same cluster is not an optimal outcome in a community detection task. Instead, our model, by obtaining a lower STD value, was able to obtain a more balanced distribution of developer nodes within clusters, meaning that it was able to better categorize the various developers in the different communities, providing us a more meaningful result than that obtained by *ABDCI algorithm*.

### RQ3

In order to have a confirmation of the validity of our algorithm with different graphs corresponding to different samples of repositories, we executed *BERTO* algorithm with ComplEx embedding technique with our Information Model for different Github repositories samples (Java, Python and Android), and for different values of embedding size and  $K$ . Table 3.7 reports the modularity achieved for each considered combination, showing that our model is robust in terms of modularity when varying the sample analyzed.

### RQ4

In Figures 3.5 and 3.6 are reported the measured execution times for the *BERTO* algorithm using ComplEx embedding techniques applied to our Information Model with the different considered samples of repositories. In particular, in Figure 3.5, we see that the execution time increases linearly as the sample size and embedding size increase. The largest times are recorded with the *Full* sample, regardless of the embedding size; this is to

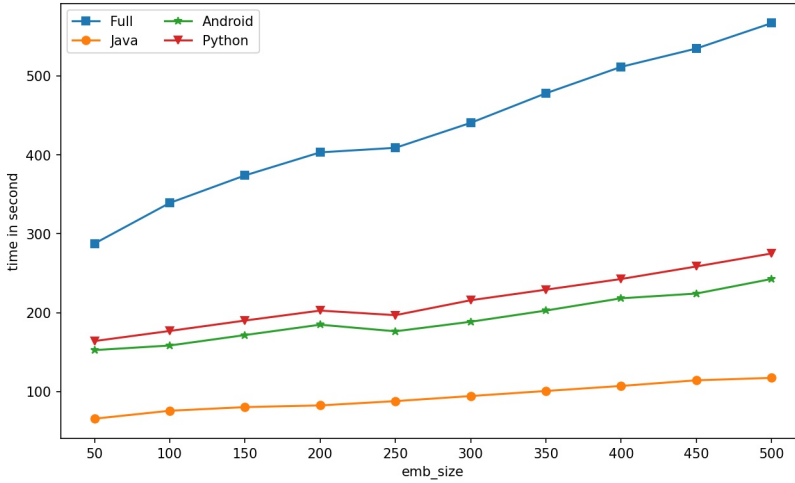
---

**Table 3.6.** (RQ2) - Comparison of the Median and Standard Deviation (STD) of community members, varying K, for *BERTO* and *ABDCI* [90]

K	Median and STD Comparison											
	emb_50				emb_150				emb_300			
	<i>BERTO</i>		<i>ABDCI</i>		<i>BERTO</i>		<i>ABDCI</i>		<i>BERTO</i>		<i>ABDCI</i>	
	Median	STD	Median	STD	Median	STD	Median	STD	Median	STD	Median	STD
50	44.12	73.97	38.10	130.79	58.08	35.41	53.04	50.48	53.03	50.99	49.60	57.93
150	14.03	26.08	16.10	24.73	18.10	12.96	19.00	15.98	19.09	14.08	18.07	17.39
250	6.53	16.78	10.02	12.57	10.01	9.37	10.00	11.36	11.03	7.50	11.04	8.42

**Table 3.7.** (RQ3) - Evaluation of Modularity for the four considered repository samples (Full, Python, Java, Android)

Modularity evaluation with different samples												
K	emb 50				emb 150				emb 300			
	Full	Python	Java	Android	Full	Python	Java	Android	Full	Python	Java	Android
50	77.75%	84.97%	78.77%	62.81%	82.64%	81.64%	69.83%	72.39%	78.57%	78.91%	64.89%	70.59%
150	67.44%	80.73%	28.81%	34.79%	77.19%	69.29%	35.62%	53.75%	72.49%	64.53%	32.63%	52.46%
250	55.18%	59.27%	14.41%	22.59%	68.23%	56.01%	19.02%	38.98%	64.55%	55.63%	17.39%	38.02%



**Figure 3.5.** Running times by varying the embedding size in *BERTO*

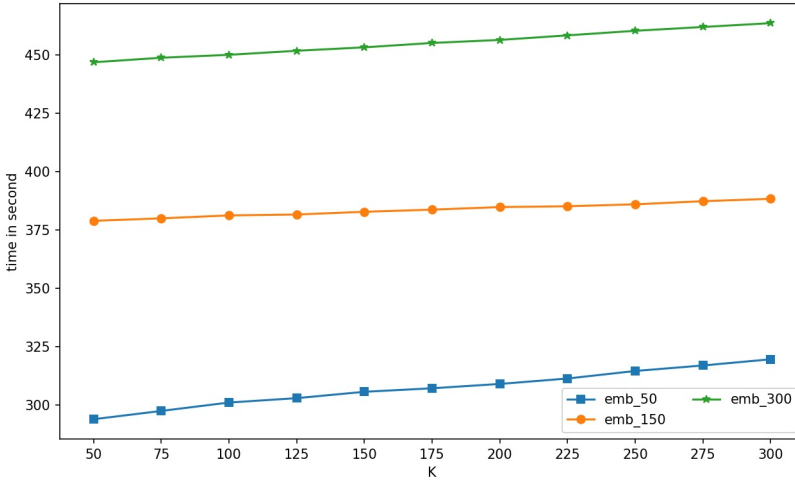
be expected because it is the larger than the others.

Another point we can observe from Fig 3.5 is that as the number of nodes within our graph increases, the execution times increase more as the embedding size increases. In fact, by placing our attention on the blue line in Fig 3.5, which represents the case where we are analyzing the graph of the *Full* sample, we can see that the execution times increase faster than in the cases where the analyzed samples are smaller (the other lines in the Figure). This means that the embedding size should be chosen on the basis of the size of the graph to support more efficiently the SNA tasks.

Instead in Figure 3.6, by varying  $K$  it is possible to observe the execution times, depending on the size of the embedding. It can be seen that the times decrease by reducing  $K$  and the embedding size, conversely they increase with a bigger  $K$  and the embedding size. Furthermore, from Figure 3.6 we can notice that varying  $K$  does not have a relevant impact on execution time.

Looking at the modularity we obtained in the previous experiments, and focusing on Figure 3.5, we can clearly say that we get the best modularity results with high values of  $K$  and embedding size. However, these two parameters impact execution times, leading us to have high execution times as their values increase. In case we have time to run our experi-





**Figure 3.6.** Running times by varying  $K$  value in BERTO

ments, it is therefore better to choose higher values of  $K$  and embedding size, since with high values of embedding size we are able to capture the topology of the network better and better, and get higher results in terms of modularity.

On the other hand, in the case where we want to be careful about execution time, there is clearly a need for a trade-off between execution time and the modularity. For example, in our experiment shown in Figure 3.5, we can see that once we pass the threshold of embedding size equals to 250, execution times begin to grow much faster as this value increases, so it is reasonable to choose an embedding size value between 200 and 250.

In practice, depending on the acquired data a parameters tuning phase is necessary to obtain a trade-off between community goodness and execution times.

### 3.4.7 Example

To evaluate the practical relevance of the communities detected by our algorithm we have selected a sample of clusters and a sample of the developers belonging to these clusters and we have analyzed in detail their activities. In the following we will present the results of the analysis involv-

ing three developers that were assigned to the same cluster. For privacy reasons, we have hidden the names of the developers and of the repositories involved in this example.

We evaluated cluster #16 from the clustering with  $K=50$  clusters having the higher Q value. 71 different developers belong to this cluster. Among them we found many developers who created or collaborated in repositories related to the development of utilities for the Android environment, in particular to mining of data during the use of Android devices. For example, we verified that the developer *Dev1* from Florida, US, created the *Rep1* repository to implement an utility for collecting stack traces generated by unexpected crashes of Android applications on rooted devices. This project was implemented in Java and Kotlin and had a good success in terms of numbers of project forks and stars (19 forks and more than 200 stars at the month of March 2023).

*Dev2* is an Italian student who developed (and also published on Google Play) an Android app for monitoring the use of the network of apps running on an Android system, within a repository *Rep2*. This app was realized in Java and C and had a greater popularity on Github, having more than 100 forks and almost 1000 stars. *Dev1* is one of the collaborators to this project who also made some commits on it.

*Dev3* from Peru is a developer who forked both *Rep1* and *Rep2* repositories and continued developing them in the context of his forks. *Dev3* is a less active developer than *Dev1* and *Dev2*, but it can be seen from his public profile that he is mainly interested in Android projects and has often made changes related to their localization. With respect to this subset of developers that the BERTO algorithm grouped together within the same cluster, we may conclude that this grouping seems significant, since these developers actually share common interests and have worked on common projects. In future, it is reasonable that these three developers may collaborate in other projects based on Android internals.

### 3.4.8 Threats to Validity

*Threats to conclusion validity.* There are threats to conclusion validity due to the selection of the sample of repositories involved in our study and to the choice of the information model.

The experiment carried out in this study involves a sample of the public

---

repositories currently on Github. They have been chosen in an arbitrary way by filtering the ones that have been created on a short period of time and that rapidly achieved a large number of stars. This selection may represent a threat to the conclusion validity because this sample could be not representative of the overall set of the public repositories on Github, but we think that our selection is able to limit possible selection biases. In order to achieve more confidence on our conclusions, in the context of the third phase of the experiment, we have evaluated the modularity of the Developers Communities produced by the BERTO algorithm for three different Github repositories samples (obtained by respectively filtering only Java, Python and Android repositories).

Our conclusions about modularity and execution times depend on the amount of information considered in our model. Of course, a model including more details about the considered interactions (for example the text associated to comment, the result of a issue or a pull request), or a modularity metric weighting in a different way the different relationships (e.g., pull requests may have a greater weight with respect to comments on issues) may produce different results. We have concluded that our model, that is richer than the one proposed by the ABCDI one [90] is able to obtain better communities in terms of modularity, but we cannot exclude that another model could be better. In future work we plan to extend our analyses on other possible models, taking also into account the correspondent increase in execution time.

*Threats to internal validity.* Since the embedding algorithms are not completely deterministic, a threats to internal validity due to randomness is present. In order to limit this threat, the community detection algorithm BERTO has been repeated 30 times and the mean, median and standard deviation values have been evaluated.

*Threats to external validity.* These threats limit the ability to generalize the results of our experiment. In community detection problems it is difficult to define a ground truth of the expected communities that should be returned, because the concept of community is often elusive: a community includes people who have similar interests and involvement in the same or similar projects, but these developers could not be aware of their belonging to a particular community. For this reason, studies related to community detection in social networks often use only topological mea-

---

asures to assess the goodness of the detected communities. We have adopted the same modularity metric used in works similar to ours [90, 127] in order to have fair comparisons between the results of the different algorithms and configurations.

### 3.5 Conclusions

In this paper, we addressed the problem of community detection inside a software repository ecosystem modeled as a particular social network and using SNA facilities. The main novelty of this work was the definition of a heterogeneous graph-based model able to capture and handle in an effective way a comprehensive set of useful, complex and strongly-correlated information about a DSN.

Then, we have challenged the problem of automatically discovering *communities* of developers sharing interests for similar projects, and we leveraged different graph embedding techniques together with clustering to overcome huge graph-size analysis issues. It should be noted that the embedding algorithms utilized in our study employ message passing techniques for information accumulation from neighboring nodes, as well as meta-path and random-walk based algorithms to compute embedding vectors representative of the initial graph [210]. Although the set of possible relationships between entities is diverse (as shown in Figure 3.2), we currently utilize embedding algorithms without estimating the weight of these relationships. However, as a future development, we may consider customized versions of the embedding algorithms that also take into account an estimation of the weights of the defined relationships. This would allow the algorithms to select random-walks and meta-paths used for embedding generation based on the weight of relationships, enabling more emphasis to be placed on relationships with higher weights during the computational process of the embedding.

In order to validate our approach, we performed an experiment where we studied the effectiveness and the performance of the proposed community detection techniques by considering different instances of DSNs and different types of graph embedding. The experiment involved data extracted from GitHub, one of the most used and popular software repository ecosystem, from which we were able to construct a heterogeneous graph

---

synthesizing relevant relationships between Developers and other entities, besides the commit relationship.

This approach has shown his capability in producing a clear division in cohesive communities.

As we model the data in our original dataset, where we considered three different kinds of nodes (i.e. Developers, Issues and Repositories) we are conscious that there are various kinds of relationships and entities that can be found inside a software ecosystem.

Because GitHub is a very rich source of information, in our future works we will focus on using all the heterogeneous information that we can gather from it and try to identify different types of communities. Furthermore, we plan to apply our model also on different software repositories and to show how the proposed community detection approach can be very useful for more specific analytic tasks such as (i) influence analysis to discover the most important developers within a given community, (ii) team formation to detect the group of developers w.r.t. a given technology within several communities in order to start more quickly a new project in according to agile paradigm, (iii) recommendation to automatically suggest the most appropriate developers to solve an issue, etc.



# Documenting Software Architecture Design in Compliance with the ISO 26262: a Practical Experience in Industry

**Abstract** Complexity of automotive systems has increased in recent years. Nowadays cars are composed by a multitude of electrical and electronic components, sensors, computer resources and so on. The ISO 26262 is a standard that deals with the functional safety of the E/E (Electric and Electronic) components of road vehicles. The standard defines a functional safety development process model that automotive manufacturing must follow and document to achieve compliance with the standard, otherwise the manufactured product will not be suitable to run in commercial vehicles.

Documenting the Software Architecture Design (SAD) is a challenging activity in industries for safety critical software systems. This is amplified when the software development must comply with the guidelines of the ISO 26262.

This paper describes the results of a practical experience we conducted in collaboration with four international companies in the automotive do-

main. In this work, we firstly performed a survey to understand the challenges that practitioners have to meet daily to develop SAD in compliance with the ISO 26262. In the subsequent step, we proposed a documentation template aiming at overcoming the challenges that emerged from the survey. The template was implemented in the Sparx Enterprise Architect modeling environment and was validated in an industrial case study that involved the same experts we enrolled in the survey. The results showed that the documentation template was judged as a valid means to produce SAD compliant with the ISO 26262 and to overcome the emerged challenges.

## 4.1 Introduction

The dominating safety Standard in the automotive domain is the ISO 26262 [92]. This Standard deals with the functional safety of the E/E (Electric and Electronic) components of a road vehicle. It is based on a system development V-process model and prescribes a series of technical requirements and recommendations. The compliance with the ISO 26262 requires the development of several types of artifact, where almost all of them can be considered as a pieces of *evidence* in the terminology of safety cases.

The Standard is wide and covers the development lifecycle of all the different subsystems (both hardware and software ones) of which an Electrical/Electronic Unit is made. In the following, we focus our work on standard Chapter 6, Clause 7 (ISO 26262§6.7), which discusses about *Software Architecture Design (SAD)* [94].

This part of the Standard defines a set of *requirements*, *recommendations* and *principles* that need to be followed during the software architecture development. A *requirement* is a criterion that must be fulfilled and is objectively verifiable. No deviation from it is permitted, otherwise compliance with the standard is not reachable. A *recommendation* is a possible suggested choice, among many others, to take in the development of a project, but without it, compliance with the standard can still be achieved. *Principles* are guidelines that need to be followed in order to achieve quality characteristics that the SAD shall address in order to avoid systematic faults. The quality *characteristics* of architecture design

---



required by the standard are: i) *comprehensibility*, ii) *consistency*, iii) *simplicity*, iv) *verifiability*, v) *modularity*, vi) *abstraction*.

Several works in the literature have described the challenges encountered in implementing different ISO Standard requirements and recommendations, and have presented possible solutions to demonstrate the compliance with ISO 26262 [58], [73], [88]. Other papers focus on solutions for implementing the ISO 26262§6 requirements regarding the SAD [64], [136], [206], [107].

Even if all these works provide valuable indications and suggestions for implementing ISO 26262 compliant SAD, it is not always easy to translate these guidelines into practical solutions. Vice-versa, industry needs well-defined and concrete procedures, artifacts, and approaches to support the guidelines defined by the Standard in a more practical manner. As an example, regarding the SAD, it may be useful for the practitioners to have a SAD documentation template that acts as a reference guide for producing artifacts having the characteristics prescribed by the Standard.

In this paper, we describe the results of a practical experience we conducted in collaboration with four software international companies in the automotive domain. For no disclosure agreement reasons we can not provide further information about them. The work was divided in three steps. In the first step, we performed a survey with industry safety experts from the automotive domain, aimed at understanding (1) which parts of the ISO 26262§6.7 are difficult to fulfill during the design of the software architecture, and (2) what are the challenges encountered by the practitioners to implement a SAD that is compliant with the ISO 26262§6.7. In the second step, on the basis of the survey results, we defined a software architectural documentation template intended to overcome the difficulties and challenges that emerged from the study. The template aims to provide a documentation model that can be instantiated in safety-related projects. We implemented an instance of the model in the Sparx Enterprise Architect (EA) <sup>1</sup> modeling environment. In the latter step, we validated the proposed template with a case study that involved the same experts we enrolled in the survey. The results showed that the proposed documentation template was accepted by practitioners as a valid means to produce SAD compliant with ISO 26262§6.7. Moreover, the template allowed to

---

<sup>1</sup><https://sparxsystems.com/>

overcome the challenges that emerged from the survey.

The remainder of the paper is organized as follows. Section 4.2 presents related works on SAD challenges in industry and in the automotive domain, while Section 4.3 presents the industrial survey we performed with experts. In Section 4.4 we present the proposed documentation template and examples of instantiating it in the EA Tool. Section 4.5 presents the study we performed to validate the proposed template and in Section 4.6 we summarize conclusions and future works.

## 4.2 Related Works

The relevance of the software documentation process to avoid defects, costly maintenance interventions, and quality degradation in software systems is well-known [46], [9], [51]. Empirical studies have shown that most of the software defects found within the testing phase can be attributed to a poor quality, outdated, or missing documentation [10], [192]. Software documentation is not getting a *passing grade*, especially inside an industrial context. Studies like [197] highlight clear gaps in software documentation practices within industrial environments. There is a strong need to turn good intentions in improving the documentation process into explicit policies and actions to be taken. This is true also for the software architecture documentation. The *software architecture documentation* that includes design and API documentation is one of the most important parts of the whole software development process.

Many works discuss the challenges and issues with software architecture documentation in the industry. In the following, we summarize related works describing such issues and challenges in industry and in particular in safety-critical domains.

### 4.2.1 SAD issues and challenges in industry

Software architectural documentation has received much attention in the research during the recent years. Low-quality documentation leads to costly revisions and maintenance, which adds to the cost and time of the entire development process [104], [115]. The work [39] reports a strong need to improve the software architecture documentation in the industrial

---

environments. The main problem identified by this study is the absence of adequate human resources and of specialized tools to support the documentation process. Aghajani *et. al* [8] present a survey from the practitioner's perspective on which are the relevant problems in the software documentation process. The most frequent issues reported in this work are: *non-complete* and *up-to-date documentation*, *inconsistency* between different documentation artifacts, *lack of tool* to support the documentation process and *maintainability of the documentation*. These problems can be solved by becoming aware that there is a need to increase the budget devoted to documentation and by the use of adequate tools and standards. In this regard the ISO 42010:2011 [6] defines a conceptual model for the software architecture description based on the system's stakeholders point of view.

#### 4.2.2 SAD issues and challenges in automotive domains

The problems related to the architecture design become more challenging when the documentation has to be developed inside a safety-critical environment under the requirements and recommendations prescribed by a safety standard [88], [167]. Safety-critical systems are systems that may lead to harm to people or the environment if they fail. Achieving an adequate level of safety means to be compliant with the most relevant safety standards like the ISO 26262 for the automotive domain or the EN 50128 for the railway domain. These standards provide guidelines in different aspects of a system, leading to a lack of common interpretation of the standard by the different stakeholder/practitioners [28]. In this context, it is recommended that the software architecture documentation shall contain models with different types of viewpoints [36]. As reported in the book [174] functional, logical and deployment views should be documented. A correct documentation under different views can contribute to the system comprehension and management [136] and can support the *abstraction* principle, one of the *characteristics* that the SAD shall address according to the ISO 26262. On the other hand, the use of different views with different levels of detail can be challenging in regard of the *consistency management* as shown in [64]. In their industrial case study, Eliasson *et al.*, report the need to use different architectures with different levels of detail and abstraction for the automotive system design. However, their

---

study highlighted problems with the consistency management among these different views, with the consequence of onerous re-work and illegal behavior of the system. Challenges in *consistency management* has also been marked by the work of Wohlrab *et al.*. In [206], the authors conducted a survey with industrial practitioners and stakeholders to analyze the inconsistencies and their consequences, between different architectural views. This study showed that the *consistency management* challenges can also be related to the interface documentation in regard to the implementation, and for wording and language between different views. According to the ISO 26262, the *software architecture design* shall be developed down to the component (unit) level. The empirical study of Land *et al.* [107], that analyzes the challenges identified in [13], highlighting the need to support the *component abstraction* and to provide a proper documentation of interfaces, including the configuration parameter. An architecture documentation of the system shall highlight the interaction between the components and describe the communication through the interfaces. Moreover, the dynamic aspect shall also address a description for timing semantics of the element, which can be useful for data consistency and system performance analysis [152]. The architecture design can impact system maintainability and testability [174]. In the automotive domain the maintainability is a key principle because its lifetime has to be considerable, unmanaged and uncorrected errors at this phase can lead to system failures.

### 4.3 Industrial survey

To comprehend the point of view of practitioners in the field of safety-critical software architecture design, we decided to develop an industrial survey. The goal of the survey was to understand which are the challenges and issues most frequently encountered when software development organizations have to comply with the safety standard guidelines regarding the software architecture design defined by the ISO 26262§6.7. We used focus groups to conduct the survey and followed the guidelines defined by Krueger and Casey [105]. We decided to use focus groups since their purpose is to better understand how people feel or think about an issue, idea, product, or service. Focus groups are led by a moderator who asks questions to participants and often encourages discussion between them

---

[105]. *Focus group interviews typically have five characteristics that relate to the ingredients of a focus group: (1) a small group of people, who (2) possess certain characteristics, (3) provide qualitative data (4) in a focused discussion (5) to help understand the topic of interest.* For our focus group, we recruited a sample of software engineers from 4 international software companies of the automobile domain. The companies included OEMs and Tier-2 suppliers of the automotive supply chain. We selected 3 engineers for each company and the focus group comprehended 12 people. To guarantee that participants were similar to each other in a way that was important to the research, we made sure that all the interviewed had more than 3 years of experience and were skilled in architecture design and implementation for automotive safety-related software in compliance with the requirements defined by the ISO 26262. We planned and executed three focus group interviews and each interview was moderated by one of the authors. In each interview, that lasted 3 hours, the moderator let the interview free to explain and discuss one of the following three topics:

1. *the challenges they deal with to meet the ISO 26262§6.7 requirements when they are asked to develop the SAD;*
2. *the challenges deal with meeting the principles defined by the ISO 26262§6.7 when they are asked to develop the SAD;*
3. *the challenges they deal with to meet the properties defined by the ISO 26262§6.7 when they are asked to develop the SAD.*

Two authors collected and analyzed the answers to abstract the main challenges addressed by the practitioners in developing SAD. The answers for which the two authors did not reach a consensus about the abstraction of the challenges, were analyzed by the third author. Finally, a two-hour final focus group interview was executed with the 12 participants to validate the inferred challenges.

The list of challenges that were abstracted by the survey, together with the ones that emerged from the study of the literature, were finally collated. Table 4.1 reports the final list of challenges and, for each of them, an ID, the reference to the specific ISO 26262§6.7 clause, and a brief description. We identified three categories of challenges reflecting the ISO Standard structure. The first three table items (evidenced in azure in the

table) regard the difficulties encountered in satisfying the SAD *Properties*. The items from  $C_4$  to  $C_{11}$  (yellow rows) are related to the difficulties in verifying the design *Principles*. The latter four ones (highlighted in green) are related to assuring specific *Requirements* at the implementation level.

## 4.4 Proposed Template

Thanks to the focus group interviews and supported from what emerged from the literature study we understood that most of the challenges were mainly related to a scant documentation of the SAD and an improper use of design tools in supporting both traceability and consistency management among the different architectural views and models. To overcome these problems we proposed a *documentation template* for describing the SAD in compliance with ISO 26262§6.7. More in detail, the template was introduced for providing a set of solutions to:

1. manage and guarantee the consistency among the different views and models' elements of the SAD;
2. support the SAD refinement process from the highest levels of abstraction towards the lowest levels of detail closer to code implementation;
3. aid the verification process of the design principles defined by the ISO Standard;
4. introduces new models for describing both static and dynamic safety-related characteristics of the software architecture.

To design the template, we followed the guidelines proposed by the SEI [51] for documenting software architectures and by the IEEE 42010 "*Systems and software engineering - Architecture description*" [6]. Lastly, the template was implemented in Enterprise Architect (EA), a tool that supports UML modeling and design, facilitating many phases of the software development process. EA support starts from the analysis of requirements until the model design, building, testing, and maintenance. Although the template is generic and tool independent, we adopted EA as the modeling environment since it was used in all four companies and because it has

---

ID	ISO REFERENCE	CHALLENGE	DESCRIPTION
$C_1$	<b>Properties, 7.4.1.b:</b> consistency	<b>Consistency management</b>	Difficulties in ensuring the consistency between the different artifact produced during the documentation process. One of the most common problem reported by interviewees is for example the difficulties in ensuring consistency between static views and behavior views: occurrence of the same component in different views, but with different names
$C_2$	<b>Properties, 7.4.1.d:</b> verifiability	<b>Verification of the design principles recommended by the ISO-26262</b>	Difficulty in finding methods for checking that the SAD adheres to the characteristics defined by the standard (in 7.4.3 Table 3) such as: modularity, maintainability and consistency
$C_3$	<b>Properties 7.4.1.f:</b> abstraction	<b>Document the SAD with a hierarchical structure</b>	Abstraction can be supported by using hierarchical structures, grouping schemes or views to cover static, dynamic or deployment aspects of an architectural design
$C_4$	<b>Principle, 7.4.3-1:</b> Appropriate hierarchical structure of the software component	<b>Verifiability of the principle</b>	Difficulties in ensuring consistency and traceability between the artifacts, usually lead to a non-well documented hierarchical structure
$C_5$	<b>Principle, 7.4.3-2:</b> restricted size and complexity of the software component	<b>Verifiability of the principle</b>	Difficulties in documenting the design of the software components at a proper level of detail to support a qualitative evaluation of the complexity for the verification of the principle
$C_6$	<b>Principle, 7.4.3-3:</b> restricted size of interfaces	<b>Verifiability of the principle</b>	Difficulties in documenting software component interfaces to help the verification of the principle
$C_7$	<b>Principle, 7.4.3-4,5:</b> strong cohesion and loose coupling	<b>Verifiability of the principle</b>	Difficulties in documenting both static and dynamic relationships among software components for aiding the evaluation of cohesion and coupling metrics to support the adherence to the principle
$C_8$	<b>Principle, 7.4.3-6:</b> appropriate scheduling properties	<b>Verifiability of the principle</b>	Difficulties in documenting the dynamic behavior interactions among the software at a low level of detail to help the verification of the principle
$C_9$	<b>Principle, 7.4.3-7:</b> restricted use of interrupts	<b>Verifiability of the principle</b>	Difficulties in documenting the dynamic behavior interactions among the software at the lower level of detail to support the verification of the principle
$C_{10}$	<b>Principle, 7.4.3-8:</b> appropriate spatial isolation of the software component	<b>Verifiability of the principle</b>	Difficulties in documenting the static aspects of the software components and their relationships at the lower level of detail to help the verification of the principle
$C_{11}$	<b>Principle, 7.4.3-9:</b> appropriate management of shared resources	<b>Verifiability of the principle</b>	Difficulties in documenting the static aspects of the software components and their relationships at lower level of detail to aid the verification of the principles
$C_{12}$	<b>Requirement, 7.4.2.b:</b> suitability for configurable software	<b>Documentation for Configuration variant and Calibration parameter</b>	Difficulties in explicitly document of configuration variant and calibration parameter inside the SAD
$C_{13}$	<b>Requirement, 7.4.4:</b> The software architectural design shall be developed down to the level where the software units are identified	<b>Document the SAD down to the unit design view</b>	Difficulty in documenting SAD down to the unit design level, especially in the early stages of a project where not all the information is yet available to ensure proper documentation.
$C_{14}$	<b>Requirement, 7.4.5:</b> Static Design shall address: external interfaces, global variables	<b>Document the SAD down to the unit design view</b>	Difficulty in documenting SAD down to the unit design level, especially in the early stages of a project where not all the information is yet available to ensure proper documentation.
$C_{15}$	<b>Requirement, 7.4.5.b.5:</b> dynamic design aspects shall address the temporal constraints	<b>Documentation for timing behavior and constraint</b>	Difficulties in documenting the temporal constraint inside the design aspects
$C_{16}$	<b>Requirement, 7.4.10.2:</b> identify or confirm the safety related part of the software.	<b>Find mechanism to easily identify safety related part</b>	Difficulty in identifying or confirm the safety related part of the software.

**Table 4.1.** Challenges in developing software architecture design emerged from the Industrial Survey and the Literature Study

been used in other industrial work, which enhances its extension capabilities through the development of add-on [113]. In the following, we describe the documentation template, how it is implemented in EA, and how the proposed solution can overcome the challenges reported in Table 4.1. The template was designed and implemented starting from the challenges we collected from the survey. The interviewees did not participate in this activity.

#### 4.4.1 The documentation template

When documenting software architectures we have to choose a set of related *architecture views*. An architectural description is by its nature multi-view because it has to fulfill different stakeholders that have different interests in the system [6]. The model of the SAD documentation template is shown in Figure 4.1<sup>2</sup>, and it's divided in two main packages: i) *Architecture View* and ii) *Requirements Views* representing two different viewpoints. Complex software systems can contain thousands of elements, all with different levels of nesting and detail. Presenting all these elements inside a single package can be confusing, making the SAD unreadable. Dividing the documentation of the entire system into different *View Packages* is a solution to split the huge amount of information enclosed in the single *Architectural View* into smaller "chunks" of information [51]. Each package represents a portion of the system with a greater level of detail. Thus, the overall *Architectural View* of the system has a hierarchical structure made of packages and sub-packages. We adopted the *UML Semiformal Notations* language as the reference modeling language to design both the static and dynamic aspects of the architecture design, rather than using Architectural description languages (ADLs) or informal notations. Usually, more formal notations, like ADLs, take longer to create, but they provide less ambiguity and better support for analysis. In contrast, less formal notations are simpler to use but offer fewer assurances. Although it was defined as a modeling language for object-oriented systems, UML can be considered as a general-purpose modeling language for any platforms or implementation technologies [51].

---

<sup>2</sup>All figures in high definition are available at: <https://drive.google.com/uc?export=download&id=1esiBRFJs7G-ao-qFP4bYsps80x0o0C29>

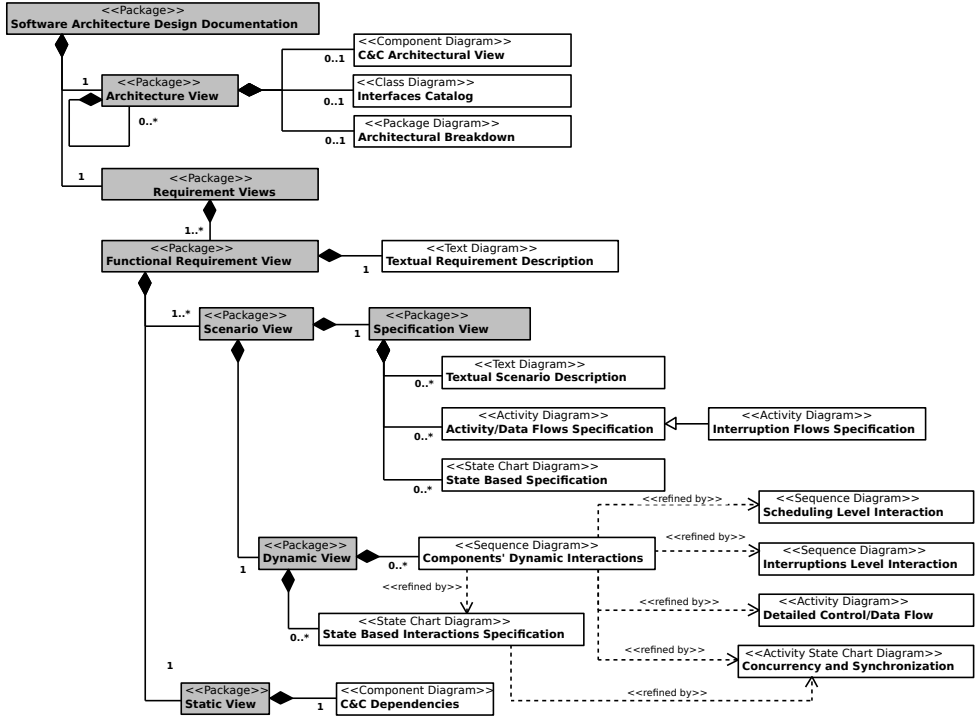


## Architecture View Package

The *Architecture View* contains the views and the static models of the overall architecture at different levels of detail. The *Architecture View* package can contain additional architecture views. This hierarchical organization supports a refinement process, where a high-level architectural view can be gradually refined and decomposed into views with a greater level of detail on the individual architectural units. Each *Architecture View* can contain up to three UML diagrams, i.e., *C&C Architectural Structure*, *Interfaces Catalog*, and *Architectural Breakdown*. The former is a *Component Diagram*, describing the software components, their connections, and dependencies at a given architectural level. The second model is actually a *Class Diagram* reporting the catalog and the description of all the *Interfaces* of the software components belonging to a specific level of the architecture. The latter is a *Package Diagram* that describes the breakdown of the architecture in terms of packages and their relationship. A more detailed description of the details, we intend to consider in the template, for each software component (*SW Component*) is shown in Figure 4.2. As rendered in the figure, a *SW Component* belongs to a package of the *Architectural Breakdown*, assuring the traceability between each component and the package where it is contained.

Moreover, each *SW Component* can be distinguished, via stereotype tags, as a *ASIL* component if it is involved in safety-related parts of the software, or as a *Quality Management* component if not. The Automotive Safety Integrity Level (*ASIL*) is a risk classification scheme defined as the result of the hazard analysis and risk assessment (*HARA*), which ranges from ASIL D to QM [93]. The most severe level of safety measures required by ISO 26262 to avoid an unreasonably high residual risk is designated as ASIL-D. If a software component is designated as *ASIL*, it is assigned with at least one safety requirement. Otherwise, *Quality Management* (QM) levels indicate that the risk of a hazardous event is not high enough to warrant safety measures, and no safety requirements are allocated to the component. In our documentation template each *SW Component* also *exposes* one or more interfaces, where each *Interface* can be either *Provided*, *Required*, or *External*, meaning that it is a third party interface required by the *SW Component*. Since, at the lowest level of detail, the *SW Component* is implemented in C, C++, Matlab, or Simulink,

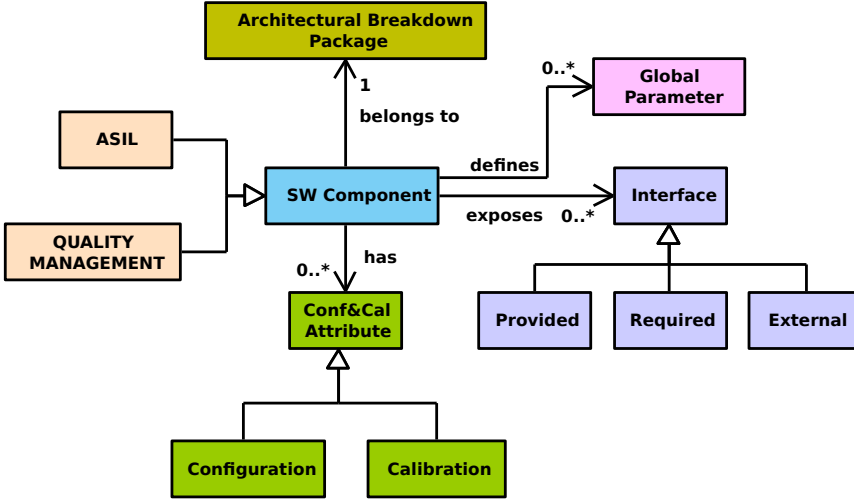
it must be taken into account whether it *defines* some *Global Parameters* or it *has* specific *Configuration* and *Calibration* attributes.



**Figure 4.1.** Model of the Software Architecture Design Documentation Template

## Requirement Views Package

The goal of this package is to collect all the functional requirement views of the architecture, where each *Functional Requirement View* package contains the views and the models describing both the static and dynamic design choices for implementing a specific functional requirement provided by the architecture. This package contains: one *Text Diagram* reporting a textual description in natural language of the requirement, one or more *Scenario View* packages and one *Static View* package. The latter package holds a *Component Diagram* named *C&C Dependencies* describing



**Figure 4.2.** Details on the Software Component Architecture Design Documentation

only the dependencies among the components involved for implementing the requirement. The *Scenario View* package contains in turn the *Specification View* package and the *Dynamic View* package. The former contains UML diagrams for the specification of the requirement. During the focus group, some practitioners indicated that it is also a good practice to have textual and visual descriptions of the scenario, hence, we decided to adopt, when needed, combinations of *Text Diagram*, *Activity Diagram* and *State Chart Diagram* to specify them. Also during the focus groups, in some companies emerged the need to specify at high level the presence of interrupt flows; this can be specified by more detailed *Activity Diagrams* with *Interruptible Regions*. According to [51] the role of the *Dynamic View* is to enrich the *Static View* by describing the interactions among the software components, at runtime, to implement the scenario. Usually, these interactions are described by means of *Sequence Diagrams* reporting the exchange of messages between the software components. When needed, in the cases where a SW Component implements a protocol, the best way to describe this dynamic behavior is through a *State Chart Diagram*. Moreover, sequence diagrams can be refined by other diagrams to highlight additional

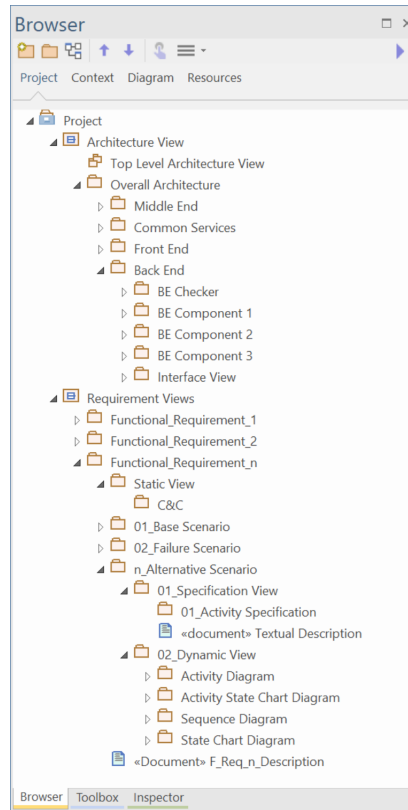
aspects of the dynamic interactions between the components such as: i) *sequence diagrams* for modeling scheduling and interruption properties, ii) *activity diagrams* for describing detailed control and data flow, and iii) *state chart diagrams* for modeling state based interactions.

#### 4.4.2 Implementation of the proposed SAD template

The template for documenting the SAD has been implemented in the Sparx EA modeling environment to support its usage by practitioners. In this section, we show examples of instances of the template in EA and why the instances of views and models in combination with features provided by the tool could be possible a solution for the challenges reported in Table 4.1. Figure 4.3 shows how the structure defined by the documentation template has been reflected in EA, where packages and sub-packages of the template correspond to folders or sub-folders in the *Project Browser* of the modeling tool respectively; each folder contains one or more models as requested by the template. In accordance with the proposed template, a SAD project is divided into two main folders, i.e., *Architecture View* and *Requirement View*. The package diagram in Figure 4.4, is an example of top-level architectural module view of a multi-tier architecture. This model, belonging to the *Architecture View*, reports an eagle view on the main packages of the architecture, their dependencies, and the sub-packages they include. EA allows one to navigate the packages and the sub-packages of the architecture breakdown, up to the most detailed view of each single component contained in a package. The component diagram, shown in Figure 4.5, reports an example of view at the lowest level of detail of a SW component, in accordance with the documentation template of Figure 4.2. This diagram gives a one-shot view of all the needed information of a SW component, such as the provided and exposed interfaces, the calibration and configuration attribute, the global parameters it defines, and whether it is an ASIL component or not. In addition, the use of the *stereotypes* mechanism in this diagram highlights the configuration and calibration parameters as well as the safety-related or unrelated nature of a component, making this information readily recognizable, as indicated by the need of the practitioners.

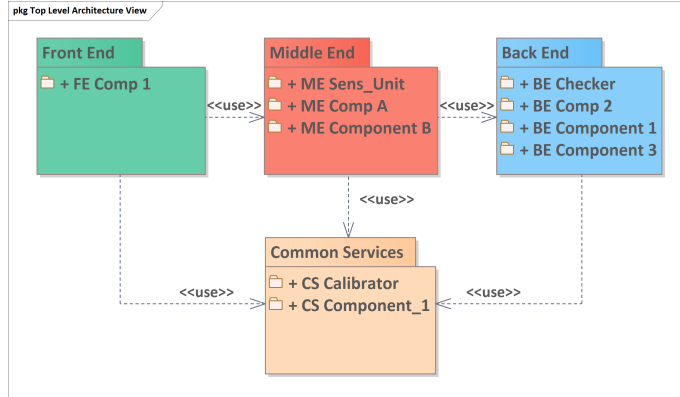
The documentation project, shown in Figure 4.3, reports an example of how the *Requirement Views* is broken down into multiple folders, each

---



**Figure 4.3.** Documentation Structure of the proposed template inside EA

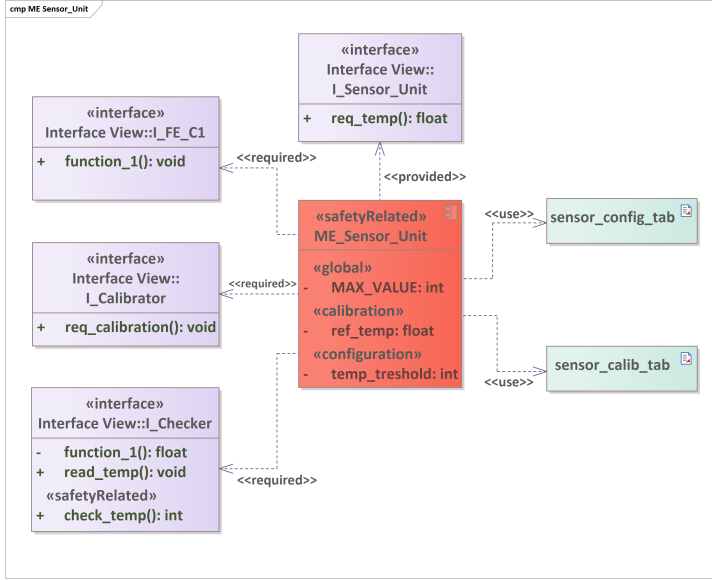
one for documenting a specific functional requirement. Each functional requirement contains a high-level textual description and a static view, that is actually a component diagram like the one rendered in Figure 4.6. This diagram shows the static relationships among all the SW components used to implement the requirement. Our example shows a possible decomposition of the functional requirements documentation into three folders, each one for a specific scenario, i.e., Base Scenario, Failure Scenario, and Alternative Scenario. All the details of a scenario are stored in a *Specification View* folder containing a *Textual Scenario Description* that can be further specified by an activity diagram and/or state-based diagram. As an example, due to their extensive interaction with the environment,



**Figure 4.4.** Example of High-Level Hierarchical View

automotive real-time and embedded systems are mostly *interrupt-driven*. An interrupt can be raised at any time adding non-determinism and concurrency to the system. A large number of potential system behaviors are produced as a result of the unpredictable interrupt arrival and preemptive interrupt handling, making it both challenging and expensive to ensure the correctness of such systems. As described in [65], [182] to specify at high level of detail the *interrupt-driven* nature of the automotive software, our template recommends the use of activity diagram models exploiting the *Interruptible Activity Regions* to surround a group of activities that can be interrupted by the triggering of an asynchronous event, as shown in Figure 4.7.

Scenarios can be further documented at the lower levels of detail by the models stored in the *Dynamic View* folder. As an example, Figure 4.8 renders a very detailed sequence diagram showing the flow of messages and the function calls that are executed at run-time by the software components involved in the implementation of a given scenario. This sequence diagram also shows how the interrupt can be modeled at code level. According to [141], in our template, we suggest the use of *Interrupt Sequence Diagram*, an extension of the sequence diagram that exploits the *int CombinedFragment* to model the interruptions' flow. The ISO 26262 requires that dynamic aspects shall document also *time constraints*. Our interviewees indicated this requirement as a challenge, indeed, it emerged

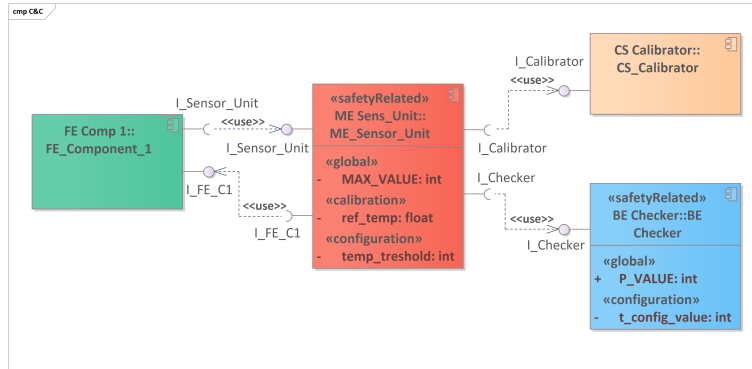


**Figure 4.5.** Detailed Component Diagram

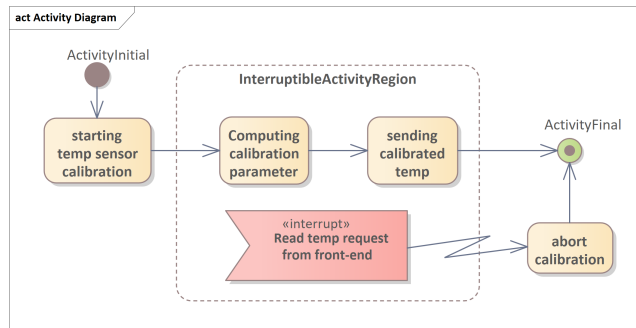
that the modeling and the verification of time constraints is often written down manually on a piece of paper. Our documentation template proposes to group within the same *CombinedFragment* a sequence of actions that must be performed in a given time interval. The *Time Constraint* notation described in [113] is placed within the *CombinedFragment* description, as reported in the first fragment of Figure 4.8.

A sequence diagram can be further refined by another sequence diagram, like the one shown in Figure 4.9, to model scheduled interactions. In our template, we suggest using the *Loop Fragment* for documenting the scheduled nature of sequences of actions.

The EA modeling environment also provides helpful features for the practitioners. One of these is the consistency management among the models since it is possible to reuse the elements already defined in other models. As an example, if the user wants to reuse components and interfaces already designed, he can fetch these elements from the folder in the project browser and move them toward the new diagram with a simple drag and drop as shown in Figure 4.10. This action generates a *hyper-link*



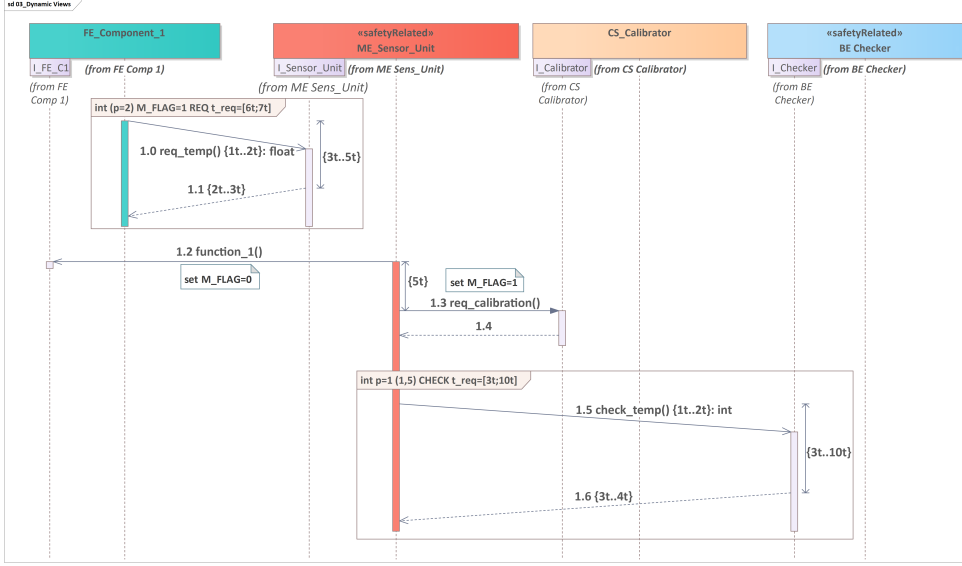
**Figure 4.6.** Component-and-Connector (C&C) View



**Figure 4.7.** High-level modeling of the interruptions flow

*instance* of the selected element in the new model and when the element is modified (e.g., changing its name, adding a function to an interface) the changes are propagated to all the hyper-link instances of that element, allowing that the information of the element are kept consistent in all the other views. Another useful feature provided by the tool is the ability to export diagrams in .xml files, like the one shown in Figure 4.11. The analysis of the .xml files can also support the verification of the design principles defined in the ISO 26262. Indeed, the files can be automatically analyzed to evaluate complexity metrics such as the average number of parameters of a function or the average number of functions of interfaces.

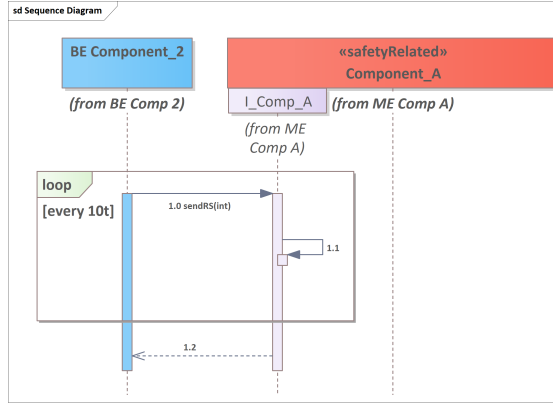




**Figure 4.8.** Interruptions Level Sequence Diagram

#### 4.4.3 Mapping between challenges and solutions

This section briefly discusses how the identified challenges summarized in Table 4.1 have been addressed by the solutions provided by the documentation template implemented in EA. The mapping between challenges and the proposed solutions are reported in Table 4.2. As table shows, the challenges from  $C_1$  to  $C_4$  have been solved thanks to the features provided by EA. Indeed, through its built-in control functionality, EA allows consistency management between different views. In addition, the package and sub-package structures of the documentation template found a direct implementation in EA through the use of folders and sub-folders, making possible to organize the SAD within a hierarchical structure in support of the abstraction principle. Challenges  $C_4 - C_{11}$  are related to the support that the documentation should provide for verifying that the principles required by the Standard are met. The use of .xml files can support a quantitative analysis for the verification of most of these principles like the restricted size of the interfaces or the cohesion and the coupling between software components. Moreover, the use of the proposed detailed



**Figure 4.9.** Sequence Diagram for Scheduling Properties

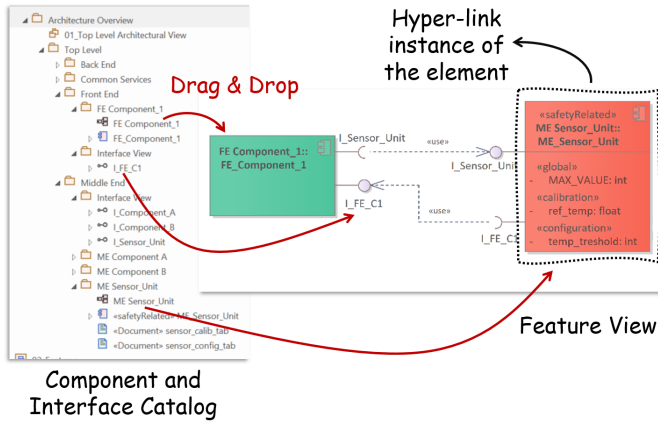
views for modeling component dynamics interactions, such as the ones for modeling interrupts or scheduling properties, can support qualitative verification of the principles related to them. The last group of challenges,  $C_{12} - C_{16}$ , is related to the Standard requirements for which code-level design solutions have been proposed. As an example, the use of stereotypes allows to easily identify safety-related parts, or the introduction of views to describe the time constraints helps in documenting time behaviors. In conclusion, from Table 4.2 it is possible to observe that all the challenges that emerged from the survey have been covered by at least one solution of the documentation template.

## 4.5 Industrial Case Study

We conducted a case study to answer the following research questions:

1. *RQ1*: to what extent is the documentation template accepted by the practitioners to produce SAD compliant with the ISO 26262§6.7?
2. *RQ2*: to what extent does the proposed documentation template overcome the safety-related challenges reported in Table 4.1?

The case study relied on the execution of the steps described in the following.



**Figure 4.10.** Usage example in Enterprise Architect

	Proposed Solution									
ID	Architectural hierarchical structure	Detailed SW Component view	Stereotype Mechanism	C&C Dependencies	Interrupt Modelling	Time constraint Modeling	Scheduling Modeling	Use of Enterprise Architect	.xml analysis	Count
C <sub>1</sub>								✓		1
C <sub>2</sub>								✓		1
C <sub>3</sub>								✓		1
C <sub>4</sub>	✓							✓		2
C <sub>5</sub>									✓	1
C <sub>6</sub>									✓	1
C <sub>7</sub>				✓					✓	2
C <sub>8</sub>							✓			1
C <sub>9</sub>					✓					1
C <sub>10</sub>				✓					✓	2
C <sub>11</sub>									✓	1
C <sub>12</sub>		✓	✓							2
C <sub>13</sub>	✓									1
C <sub>14</sub>		✓	✓							2
C <sub>15</sub>						✓				1
C <sub>16</sub>			✓							1
Count	2	2	3	2	1	1	1	4	5	

**Table 4.2.** Traceability Matrix between Challenges and Solutions

## Subjects selection

We recruited from each company the same three software engineers that were interviewed during the focus groups. We had one 30-minute meeting with the subjects, where one of the authors presented them the documentation template implemented in Enterprise Architect.



Figure 4.11. xml file exported from EA

## Objects selection

In each company we chose a closed project, i.e., a project that had been already shipped to the customers and was under maintenance, on which the subjects have been stakeholders, i.e., designer, developer, or tester. We asked the project managers of each project to select from these projects 6 requirements having the following three characteristics: (i): the requirement had to be safety-related; (ii): the requirement had to be of medium or high complexity, (iii) the requirement was not to have been designed, nor implemented, or tested by none of the subjects. For each project, two of the authors and the project manager implemented in EA the folders and the views for modeling the high-level view of the architecture like the one shown in Figure 4.4.

## Experiment execution

We allocated for each subject two requirements, in this way a requirement was assigned to one subject. Then we asked the subjects to model the requirements assigned to them in the EA template. We gave them two hours to complete this task, and the subjects were free to analyze and to

take cues from the SAD documentation already implemented without our template. All the subjects were able to accomplish the task in two hours.

### Validation survey

In each company, we had one hour meeting with the three subjects to present them, for each requirement, both the old and the new produced documentation. Finally, we submitted them a 30-minute survey made by 4 open questions to answer *RQ1* and 16 closed questions for *RQ2* answering. The four open questions are the following.

- Q1 *Could you briefly describe the improvements, if any, provided by the documentation template in SAD?*
- Q2 *Could you briefly describe which are the strengths of the documentation template?*
- Q3 *Did you have difficulty applying the template? If yes, could you please summarize them briefly?*
- Q4 *Could you briefly describe the limitations of the template?*

For each challenge  $C_x$  of Table 4.1 interviewed had to answer, by selecting one of the values of a 5 levels Likert scale, the following closed question:

*"Based on your experience, how do you rate the support of the template to overcome the challenge  $C_x$ ?"*

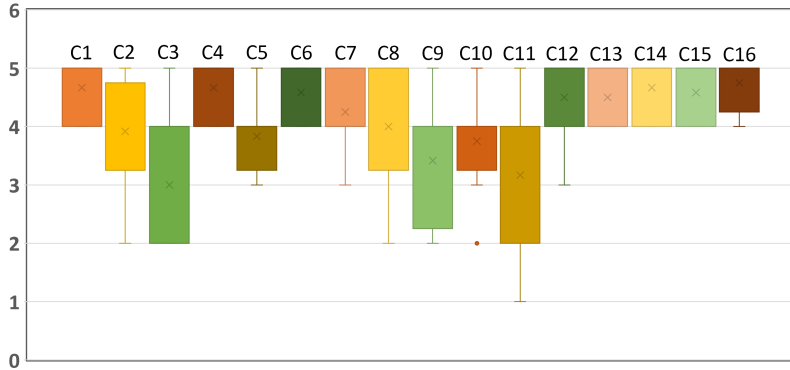
The possible values of the Likert scale were: 1 : No support, 2 : Low, 3 : Neutral, 4 : Good, 5 : Excellent.

### Data analysis and RQs answering

the data were analyzed to obtain the evidence needed for answering the research questions. Regarding *RQ1*, the answers collected through the open questions were independently analyzed by two researchers. They marked quotes that were seen as interesting for the study and coded them. The coded data were analyzed to gain findings that were used to answer the research question. The results of the researchers' analysis were compared using a data triangulation approach. Moreover, the results were also

---

reviewed by the interviewed subjects to identify and solve possible misinterpretations. As for *RQ2*, the answers were grouped in the box plots chart shown in Figure 4.12. Every single box plot displays the distribution of the twelve answers we collected for each challenge.



**Figure 4.12.** Box plot showing the results of the survey for answering *RQ2*

**Answer to *RQ1*** from the collected data we summarized the evidence described in the following.

- A1 All the interviewees found improvements in the proposed documentation template w.r.t. the previous one. In particular, they appreciated the enhancements due to (1) the aid provided for the automated consistency management among the different views and the models' elements, (2) the new kinds of models introduced for describing temporal constraints and scheduled activities, (3) the interrelated module views of the software architecture design, at different levels of detail, and (4) the detailed characterization of the software component in terms of *global parameters*, *exposed interfaces*, and *configuration and calibration attributes*.
- A2 All the interviewees described at least one strength in the proposed template. Almost all the participants have pointed out that the proposed documentation facilitates the verifiability of the principles requested by the Standard. Two of them found very useful the simple

mechanism to distinguish between safety components and non-safety components. In particular, they found very promising and helpful the generation of the `.xml` file for the evaluation of the complexity metrics of the project.

- A3 Participants did not raise particular difficulties in applying the template. Three of them had some trouble in managing the reuse of modules and interfaces in the construction of detailed sequence diagrams. Another one evidenced a possible ripple effect after a reused element has been renamed.
- A4 The main limitations pointed out by the participants are (1) the lack of traceability links to the requirements that are handled by each company in a different tool, like IBM Doors or Confluence, and (2) the lack of mechanisms for supporting multi-users collaborative work.

These evidences allowed us to answer the *RQ1* as follows:

*The proposed documentation template has been accepted by the surveyed practitioners as a valid means to produce SAD compliant with ISO 26262§6.7.*

**Answer to *RQ2*** from the box plots shown in Figure 4.12 it is possible to observe that the template has been rated, by all the practitioners, between *Good* and *Excellent* to overcome 9 of the 16 challenges. Only for challenges  $C_3$ ,  $C_9$ , and  $C_{11}$  the template actually did not provide any useful improvement for some of the surveyed practitioners. As for  $C_3$  we understood that in some companies the template adopted in many companies already had a good abstraction of the architecture at different levels of details. Regarding  $C_9$  although the practitioners appreciated the models to represent the interruptions, some of them did not judge the template useful to verify if the designed architecture does a restricted use of them. From a more detailed analysis of the answers, we realized, for challenge  $C_{11}$ , that the template in some case is very useful to verify how shared software resources have been managed, but it does not give any information on very low level shared resources such as memories, CPUs, hardware

devices, etc.. From this analysis, we were able to answer the *RQ2* as follows:

*The proposed documentation template was considered, by the surveyed practitioners, a good solution for overcoming the challenges evidenced in this study.*

### Threats to validity

the study presents some threats to its general validity. The main threat affecting the generalizability of the results is due to the limited number of companies where the study was conducted. A possible solution to mitigate this threat is to extend the proposed approach in other companies trying to diversify the characteristics of the sample. Another threat is due to the choice of the subjects and objects of the study. Indeed, the same people interviewed in the focus groups to collect the challenges were the subjects of the case study that evaluated the documentation template on requirements already designed and documented. A possible mitigation is to execute focus groups on a greater sample of interviews, to collect additional challenges, and to validate the template by involving different subjects for designing and documenting projects starting from scratch. Another mitigation is to perform the same study by considering other parts of the ISO 26262, or even in other domains like railway and aircraft.

## 4.6 Conclusion and Future Work

In this paper, we presented the results of a practical experience we conducted in collaboration with four automotive partner companies. We first performed focus group interviews to understand the challenges that the practitioners, employed in these companies, have to deal with when they design the software architectures in compliance with the Standard Chapter 6, Clause 7 (ISO 26262§6.7). From the interviews, we understood that these challenges were related to a scant documentation template. To overcome these challenges we proposed a new documentation template in accordance to the guidelines suggested by the SEI. Moreover, the template

---



was implemented in the Enterprise Architect modeling tool. Finally, we performed a case study demonstrating that the proposed template was accepted by the practitioners employed in the four companies and to what extent it can be considered as a valid support to overcome the raised challenges.

The case study also pointed out some limitations that we intend to address in the future. We plan to find suitable solutions to integrate the Enterprise Architect documentation template with other tools used in the software development life cycles, such as requirements management systems, software repositories, and tools supporting the testing phase. Moreover, we want to overcome the limitation due to the lack of support for collaborative and multi-user work, a feature that is increasingly required in agile processes. To mitigate the threats to the validity of our study we intend to extend our process to other companies also involved in different domains. Moreover, to allow the reuse of models and views from projects already implemented, we want to introduce a reverse engineering process for supporting the re-documentation of software architectures according to our template. Lastly, we intend to apply machine learning or natural language processes approaches to the `.xml` files for the automated verification of safety design principles.

---



# Automated Architecture Recovery for Embedded Software Systems: An Industrial Case Study

**Abstract** The software architecture documentation of embedded systems is often overlooked in industry, due to time pressure, project budget constraints, and lack of culture. However, adequately documenting the architecture from different points of view is mandatory to reach the expected maintainability, testability, and safety requirements. This paper presents a software architecture recovery (SAR) process for automating the documentation process of embedded system software architectures. The approach uses static code analysis to extract detailed information about the systems and reconstruct architectural models. It has been implemented in a tool that automatically generates different UML models, including package diagrams, component diagrams, component and connector diagrams, and state machine diagrams. To evaluate the effectiveness of our approach, we conducted a survey with industrial experts within Micron, that allowed us to assess the accuracy and usefulness of the generated documentation.

## 5.1 Introduction

Software documentation, as a critical aspect of software development processes, faces numerous challenges, particularly in industrial settings where code rapidly evolves through frequent development iterations. The changes in the code throughout multiple development cycles create a complex set of demands for software documentation. In such a situation, keeping the documentation accurate, corresponding to the contemporary state of the code, and not getting lost during development processes is challenging. Indeed, as software projects evolve or change ownership, their documentation can become outdated or fragmented. Industries engage in re-documentation efforts to ensure that the documentation remains accurate, comprehensive, consistent, and reflective of the current state of the project. Moreover, Continuous Integration/Continuous Deployment (CI/CD) practices have revolutionized software development lifecycles by automating build, testing, and deployment processes. Integrating documentation tools into the CI/CD pipelines is essential for establishing a robust mechanism to ensure continuous harmony between the codebase and its corresponding documentation. This proactive approach minimizes the risk of inconsistencies and fosters agility and efficiency in the development lifecycle. While various reverse engineering techniques exist for recovering software documentation from traditional software [114], few are tailored to embedded software systems [187]. These systems, designed to work on dedicated hardware platforms, have to satisfy strict real-time and other quality requirements, are often implemented in low-level coding languages, and pose unique comprehension challenges. Therefore, there's a pressing need for techniques to extract structural and behavioral models from embedded systems, enhancing comprehension and simplifying maintenance.

This paper introduces a tool-supported Software Architecture Recovery (SAR) process aimed at automatically generating Software Architecture Documentation (SAD) from the code of embedded software. The SAR process utilizes a reverse engineering approach, leveraging static analysis of the C code of embedded software systems to recover models that can be integrated into the architectural views of the documentation template proposed in [16]. The resulting SAD includes various UML models, such as

---

package diagrams, component diagrams, component and connector (C&C) diagrams, and state chart diagrams. Finally, we conducted a questionnaire-based industrial survey involving 14 participants from Micron developing embedded software systems. The survey had a dual purpose: to evaluate the accuracy and utility of the UML diagrams automatically produced by the tool and to gather feedback on the limitations of the SAR process for further refinement. The contribution of this paper is threefold:

1. proposing a tool-supported SAR process that automatically produces UML models from static analysis of C code for embedded software systems;
2. conduction of an industrial survey to validate the effectiveness of the SAR process within real-world contexts;
3. evaluation of the accuracy of the UML diagrams generated by the tool and its usefulness through a survey involving developers of the industry in which the study was conducted.

The paper is structured as follows, Section 5.2 describes the studies related to this work, Section 5.3 presents the proposed SAR process, Section 5.4 discusses implementation details, Section 5.5 presents the industrial survey we conducted, and finally, conclusions and future research directions are summarized in Section 5.6.

## 5.2 Related Studies

Reverse engineering techniques apply to software and hardware systems, to extract and identify system structure and design for better comprehension. This process enhances program understanding in software systems by extracting design information about components and their relationships, presenting them at a higher abstraction level. Reverse engineering techniques are particularly valuable for maintaining and comprehending poorly documented or legacy systems, which are often affected by Architectural Technical Debt due to resource limitations and inadequate documentation [110, 196]. As outlined by Nelson *et al.* [137] reverse engineering can have different goals, i.e., *Design Rediscovery*, *Reengineering*, and *Redocumentation*. *Design Rediscovery* aims to develop a system model

---

at a higher level of abstraction. *Reengineering* combines reverse and forward engineering techniques to understand which functionalities need to be refactored, deleted, or added. *Restructuring* aims to refactor the whole system to improve maintainability, readability, or other quality attributes by keeping the system’s functionalities intact. Lastly, *Redocumentation* aims at generating new documentation of updating existing ones depicting the system behavior. Moreover, reverse engineering processes rely on the execution of three steps [178]: *Extraction*, to gather data from the source code and existing documentation to retrieve design and construction artifacts; *Abstraction*: to synthesize and abstract the extracted information in a format less dependent on its implementation; *Presentation*: to convert the abstracted information into a format that is user-friendly and easy to comprehend.

### 5.2.1 Software Architecture Recovery (SAR)

SAR processes are an extension of reverse engineering and focus on abstraction and presentation, highlighting the architectural structure [185]. SAR addresses issues like architectural discrepancies during software evolution [193, 169, 74]. SAR techniques are critical for software maintenance, especially in understanding and modifying legacy systems. They assist in conformance checking, reconstructing descriptions, and analysis for meeting new requirements [47]. Different SAR techniques have been developed, ranging from static and dynamic analysis to machine learning approaches. Static analysis does not require the executing of the code and extracts architectural structures and dependencies [181]. Dynamic analysis, involving code execution, aims to create high-level system models. Revealer [148], using lexical and syntactic code analysis, abstracts high-level architectural views using regular expressions and XML-based patterns. Sora *et al.* [173] introduced a static analysis-based approach using PageRank algorithm to identify important classes. NEGAR, a machine learning algorithm by Chem *et al.* [45], uses graph representations of software dependencies for clustering related files, aiding maintenance. These approaches vary in accuracy and effectiveness but collectively contribute to a deeper understanding and maintenance of software systems.

---

### 5.2.2 Reverse Engineering of State Chart Diagrams

Recovering state machine behavior involves two main strategies: dynamic analysis, which logs features and execution flow through code instrumentation and static analysis, based on source code, allowing for customization. Static analysis as the one proposed by Walkinshaw *et al.* [132] utilizes symbolic execution and program conditioning on source code. It identifies state transitions and annotates them with corresponding code segments, observing that transition points often correlate with specific syntax elements. This model is especially effective for small, structured systems like object-oriented ones. Tonella *et al.* [188] consider each method call on an object as a transition, an assumption not applicable to embedded firmware with implicit state machines. Similarly, Kung *et al.* [106] and Sen and Mall [162] employ symbolic execution for automatic state machine extraction from C++ and Java sources, focusing on small methods and state variables influencing branching in object-oriented systems. Bae *et al.* [25] suggest generating state charts from contract-based specifications, using method pre and postconditions instead of path conditions, and strengthening transition conditions when necessary. Researchers like van den Brand *et al.* [194], Knor *et al.* [102], and Somé *et al.* [172] have worked on extracting state machines from procedural code with specific patterns, like nested-choice structures. However, these techniques are limited and not widely applicable to more diverse code categories.

## 5.3 The proposed reverse engineering process

In this section, we present the proposed Software Architecture Recovery (SAR) process, which is based on static analysis and used to recover both the structural and behavioral views of embedded software. According to the five-axis taxonomy for SAR characterization presented by Ducasse *et al.* [61], our SAR focuses on documentation and conformance. Our SAR was used to analyze an embedded firmware developed for managing NAND memory systems. This firmware provides different functionalities for memory access to external hosts and it's designed to be configurable for running on different hardware platforms. Moreover, the firmware follows a layered architectural style, where the lower layers are strictly related to the hardware architecture. To ensure this portability requirement, the

---

firmware is developed according to company-specific coding practices that make extensive usage of macros and pre-processor directives. As a consequence, the resulting code is difficult to analyze and comprehend. The SAR outputs include various UML diagrams, including Package, Component, Component and Connector (C&C), and State Chart models. The recovered models can fill the views of the documentation template proposed in [16], designed for documenting ISO26262-compliant automotive software systems. This template is suitable for documenting embedded software systems as well. The SAR process, as illustrated in Figure 5.1, consists of four phases. It has been implemented in a tool that exploits the features of both Enterprise Architect<sup>1</sup> (EA) for UML modeling and reverse engineering, and Pycparser<sup>2</sup>, a Python library for analyzing C code and generating Abstract Syntax Trees (AST). In the following, we provide an overview of these phases and their inner activities.

**1. Pre-Processing:** This phase includes several activities: first, during the *Source Code Compilation at “Preprocessing Stage”*, the compilation is stopped early to expand macros, resolve `#include` directives and headers, and remove comments, producing the *preprocessed source code*. This step is essential for processing with Pycparser and Enterprise Architect. Next, in *Data Cleaning*, the source code is cleaned up of all the code generated by the expansion of header files and directives, resulting in a clean version free of macros or keywords. Finally, in *Source Code Folder Structure Generation*, the source code is reorganized into a new configuration of folders and subfolders that reflects the architectural design of the software.

**2. Extraction:** This extraction process involves two main activities: first, the *Enterprise Architect Reverse Engineering* feature is used to generate structural models of the software system, that include package and component diagrams. Second, the *Parsing of preprocessed code* involves adapting and parsing the *preprocessed code* to produce an abstract-syntax tree (AST). This AST is subsequently analyzed to create an intermediate representation (IR) of a state chart, which includes detailed information about states and transitions, including guards, signals, and triggers.

---

<sup>1</sup>Enterprise Architect: <https://sparxsystems.com/>

<sup>2</sup>Pycparser: <https://github.com/eliben/pycparser>

---



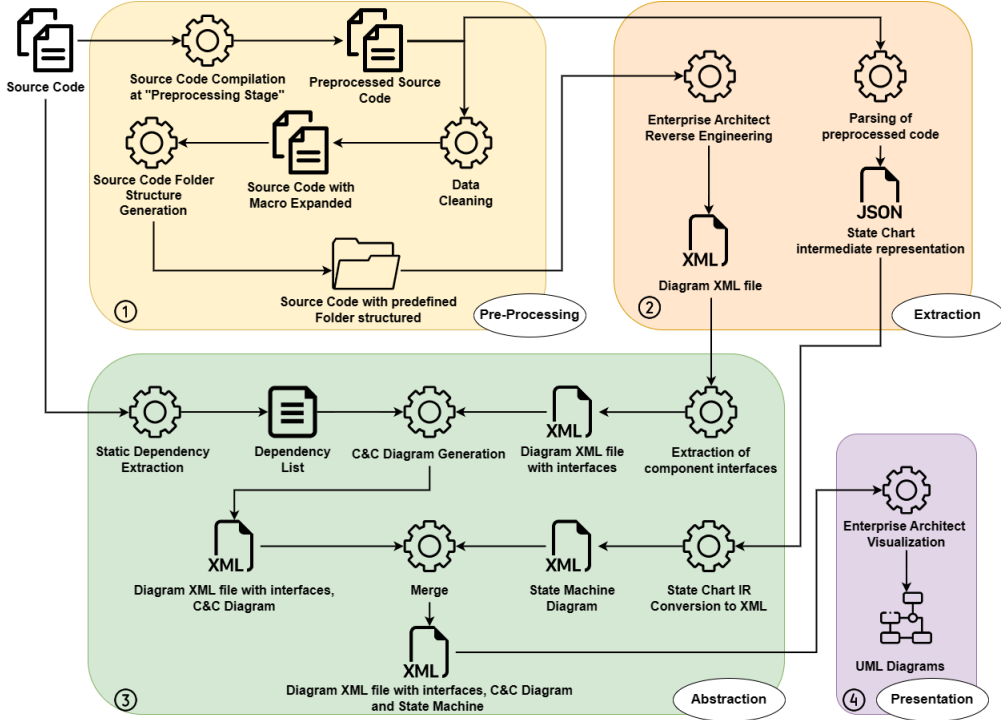


Figure 5.1. The proposed SAR process

**3. Abstraction:** The models produced by EA are enhanced by extracting component interfaces and generating C&C diagrams. This process includes several key activities: First, *Static Dependency Extraction* involves analyzing the source code to identify static dependencies between components. Next, in *Extraction of Component Interfaces*, the XML file representing the extracted UML diagram is automatically edited to include UML interfaces. *C&C Diagram Generation* follows, utilizing the list of static dependencies and the XML file to generate the diagrams automatically. The *State Chart IR Conversion to XML*, converts the intermediate representation (IR) of the state chart in an XML format compatible with the syntax of EA. Finally, a *Merge* between the XML file containing the State Chart Diagram and the XML file containing the structural model is performed to obtain an XML file that integrates both structural and

behavioral models.

**4. Presentation:** Using Enterprise Architect, XML files are displayed as UML diagrams. The *Enterprise Architect Visualization* activity requires inputting the merged XML into the tool to create the UML representation.

## 5.4 Implementation Details

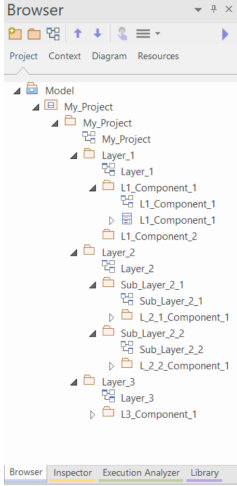
In the following, we describe the adopted technological solutions to recover the architectural models.

**Pre-processing.** Is a crucial activity in our recovery process, designed to address company-specific coding and programming practices, especially the extensive use of macros that complicate analysis and understanding. This phase standardizes the code's syntax to align with the common syntax standards of the programming language. Our starting point is a collection of source code files organized in a single folder, including both .c files and associated header files (.h). To align the source code with the requirements of EA and Pycparser, it undergoes a pre-processing phase via the C compiler. This is ensured by executing the compiler command `gcc -E` which interrupts the compilation of the code in the preprocessing phase, resulting in what we define as *preprocessed source code*. This phase expands macros, resolves `#include` directives and headers, and removes comments from the source code. The pre-processed code has a dual function: it is routed to Pycparser for the identification of the finite-state machine and is simultaneously submitted to the *Data Cleaning* step. After this step, a variant of the source code identical to the original is obtained, but with expanded macros and keywords and resolved directives.

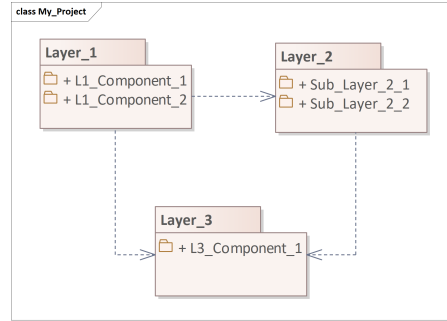
**Package Diagram generation.** In the provided project, the names of the .c files and their corresponding .h files reflect the layered structure of the embedded software system. We implemented a Python script that analyzes the names of all pairs of .c and .h files and creates a package structure, in terms of folders and sub-folders, mirroring the architecture of the software system. The newly-organized source code is then input into EA to leverage its static code analysis capabilities and generate UML

---

diagrams. Consequently, the tool produces a package diagram of the entire project, reflecting the folder and sub-folder structure provided as input, as shown in Figure 5.2 and Figure 5.3.



**Figure 5.2.** Package Diagram browser view



**Figure 5.3.** Reconstructed Package Diagram

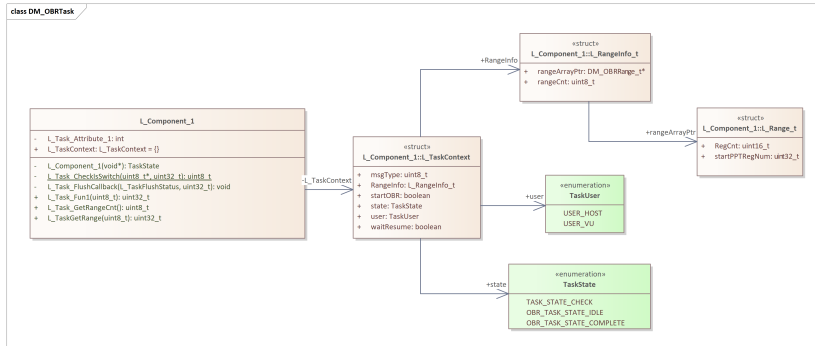
**Component Diagram generation.** For each .c and .h file pair, EA automatically generates a UML class diagram, like the one shown in Figure 5.4. These diagrams include attributes, operations, structs, and enumerations. Attributes and operations are classified as public if declared in the .h file and implemented in the .c file; otherwise, they are private if only defined in the .c file. EA allows these UML diagrams to be converted into an XML format for easy modification. These XML files are structured into tables such as `t_package`, `t_object`, and `t_diagram`, each one holding different UML elements like packages, classes, and diagrams respectively, as shown in listing 5.1). Moreover, each table row represents a UML element with attributes including `Name`, `Object_Type`, and `ea_guid`, the latter serving as a unique identifier for the element. To enhance the diagrams generated by EA and align them with the reference metamodel, we developed a Python script to manipulate the XML files representing UML diagrams. Firstly, the script updates the `Object_Type` attribute of

each *Class* object to convert it into a *Component* object. Subsequently, an *Interface* object is created and added to the `t_object` table. To achieve this, we follow the XML file structure, setting the `Object_Type` attribute to *Interface* and adding the `ea_guid` of the package where we intend to place the interface within the `<Extension>` tag. This ensures that the Interface and its corresponding Component reside in the same package. Next, the tool generates a Realization `Object_Type` to establish the relationship between the Interface and Component. This is achieved by specifying the respective `ea_guid` of the interface and component within the `<Extension>` tag of the realization object being created. Finally, the script analyzes the `t_operation` table, appropriately modifying the `ea_guid` within the `<Extension>` tag to bind the interface with the operation it should contain. In Figure 5.5, the diagram reconstructed by our tool is depicted, where we can observe the interface extracted from the component.

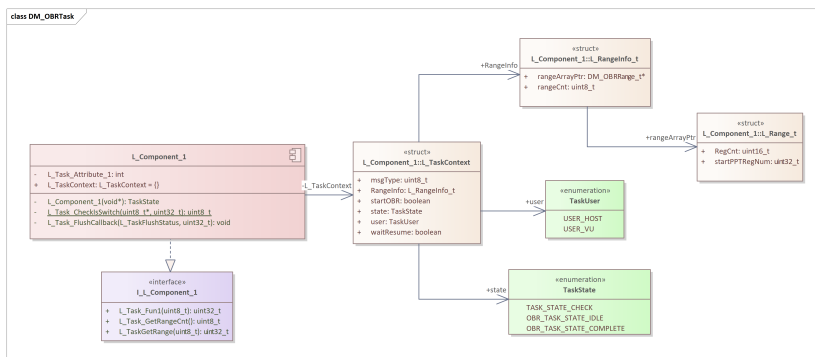
**Listing 5.1.** Sample of the XML file generate by EA

```
<Package name="My_Project" guid="{350E9CE0-4958-42c0-8F83-1E449CB016AC}">
  <Table name="t_package">
    <Row> [...] </Row>
  </Table>
  <Table name="t_object">
    <Row>
      <Column name="Object_id" value="4"/>
      <Column name="Object_Type" value="Component"/>
      <Column name="Diagram_id" value="2"/>
      <Column name="Name" value="My_Component"/>
      <Column name="Author" value="marco"/>
      <Column name="Package_id" value="2"/>
      <Column name="GenType" value="C"/>
      [...]
      <Column name="Scope" value="Public"/>
      <Column name="ea_Guid" value="{A7FCE46C-3C14-48c6-8BFB-07FAB08F22D7}"/>
      [...]
      <Extension Package_ID="{350E9CE0-4958-42c0-8F83-1E449CB016AC}"/>
    </Row>
    <Row> [...] </Row>
  </Table>
  <Table name="t_diagram"> [...] </Table>
  <Table name="t_operation"> [...] </Table>
  <Table name="t_connector"> [...] </Table>
</Package>
```

**Component and Connector (C&C) Diagram generation.** Using `gcc` to compile code it is possible to generate a `MapFile`, which details the memory layout of code including functions and variables. This file also includes *cross-references* among symbols to identify function calls, enabling the extraction of static dependencies list between different functions within the system. We have developed an additional Python script

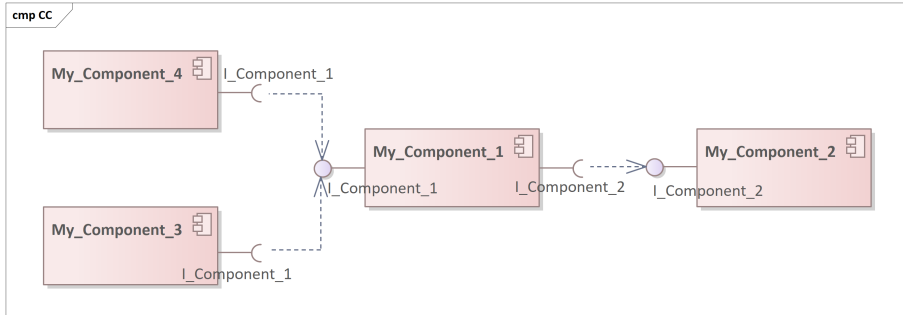


**Figure 5.4.** Class Diagram reconstructed by Enterprise Architect



**Figure 5.5.** Component Diagram reconstructed by the tool

to automatically reconstruct C&C diagrams by taking as input the list of static dependencies extracted from the *MapFile* and the XML files of the UML diagrams. For each component present in the XML file, the script adds *Provided* and *Required* interface objects, specifying their *Object\_Type* and inserting the *ea\_guid* of the corresponding component in the `<Extension>` tag. Subsequently, the script inserts a new diagram object into the *t\_diagram* table of the XML file. Leveraging the list of static dependencies, the script can: i) incorporate into the diagram the components with which the component in question interacts, ii) reconstruct the dependency relationships between the various components, which are appropriately created and inserted into the *t\_connector* table. At the end of



**Figure 5.6.** C&C Diagram reconstructed by the tool

the process, the C&C diagram for each component is reconstructed, and the new XML file, representing the UML representation, is provided as output, depicted in Figure 5.6.

**State Chart Diagram generation.** Our SAR process also reconstructs State Chart diagrams from the source code, to represent the system's states and transitions for a better understanding of its behavior. To achieve this, we rely on the task table, a table containing the function pointer of the APIs scheduled by the embedded operative system as a task. This table is used to identify tasks in the pre-compiled source code, ensuring proper parsing. The tasks are units of execution and are implemented as state machines. After processing the code into a parsing tree, we generate an intermediate representation (IR) in JSON format, which is then converted into XML format, to be given as input to EA. The recovery of state chart diagrams relies on several assumptions. It presupposes that each task consistently produces a return value indicating its status and employs a **switch-case** construct to define a state machine. The tool also requires well-formed **switch-case** statements composed solely of **case-statements** to define behavior. The state variables are identified as the condition expression in the **switch-case** statement. Signals and triggers are discerned through the use of APIs provided by the embedded operating system. Furthermore, tasks may include multiple exit points, and the presence of spaghetti code allows for the use of jump instructions such as **goto**. All these cases are handled appropriately by our tool. The proposed reverse

engineering approach relies on the execution of the following 4 steps:

**Prepare state-machine before starting:** the *preprocessed source code* is elaborated to simplify the abstraction of state-machines. This adaptation is necessary because the C language allows various patterns for describing state machines, primarily using the **switch-case** structure. In some cases, **switch-case** statements are embedded within selective or iterative flows, which are not well-formed for straightforward conversion into state chart diagrams. Therefore, these **switch-case** statements require refactoring to align the C constructs with the state chart diagram representation.

**State variable definition:** according to the criteria outlined by Said [153], our approach is based on a specific pattern of **switch-case** implementation. In this context, the state variable is defined as the condition expression of the **switch** statement.

**Guard & transition definition:** drawing from Walkinshaw’s strategy [132], our method extracts guards from source code analyzing two key variables: the state variable, discerned within **switch-case** statements, and the return variable, typically located at the begging of variable declarations. These transitions between states are further classified into three categories: event-driven, **goto**, and rescheduling transitions. Event-driven transitions correspond to specific events, while **goto** transitions occur independent of triggers, and rescheduling transitions follow guard condition verification, subsequently rescheduling tasks in subsequent cycles. Furthermore, to enhance readability, we prioritize transitions, inspired by Said et al. [153], over a sequential read approach, and conduct boolean reduction on conditions. Additionally, recurring sub-expressions or API calls in the code are identified and simplified with the use of symbolic names.

**Generate XML diagram EA compatible:** the step for generating the XML output file involves several steps. Initially, Entry and Exit point UML state chart objects are created to mark the beginning and end of the state machine. An additional Exit point may also be generated to handle error conditions if necessary. Subsequently, node states are defined for each state in the state machine, transitions

---

between states are established, and guards, triggers, and signals are applied as needed.

In Figure 5.7, we present an example of a task-based switch-case statement. Figure 5.8 shows the corresponding reconstructed state chart diagram.

```
TaskReturn_t task_example(void* args) {
    TaskReturnState_em ret;
    switch(taskContext.state){
    STATE_CMD_GET: case STATE_CMD_GET:{
        CmdPtr_t* CmdHandler = HAL_GetCommand();
        if (CmdHandler != NULL){
            taskContext.cmd = CmdHandler;
            goto STATE_CHECK_PRIORITY;
        }
        return TASK_READY;
    }
    STATE_CHECK_PRIORITY: case
    STATE_CHECK_PRIORITY: {
        CmdHeader_t* cmd =
            taskContext.cmd->header;
        if (cmd->flag & ATTR_MASK) ==
            PRIORITY_MASK){
            if (cmd->hit){
                OS_WaitEvent(EVENTID_HIGH_PRIORITY_DONE);
                return TASK_SUSPEND;
            }
            else
                cmd->flag =
                    Analyze_Command(CmdHandler);
        }
    }
    STATE_EXECUTE_CMD: case STATE_EXECUTE_CMD: {
        uint8_t cmdIdx = taskContext.cmd->cmdIdx;
        CmdPayload_t* cmdEntry =
            taskContext.cmd->payload;
        if (CmdAbortStatus(cmdIdx) == true){
            if(OS_ResourceLocked() == true)
                OS_ReleaseResource();
            goto STATE_CMD_GET;
        }
    }
    }
}
```

Figure 5.7. Task Code Example

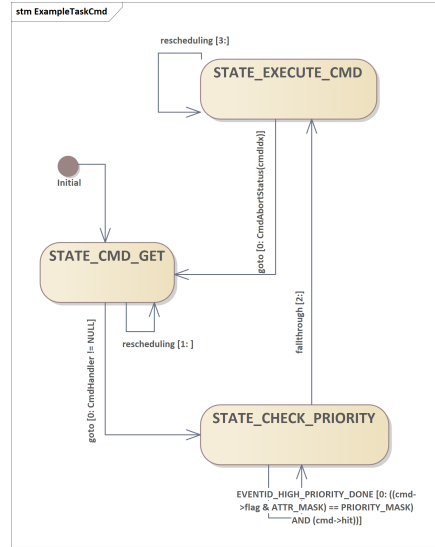


Figure 5.8. Reconstructed Task State Chart Diagram

## 5.5 Experimental evaluation

We aimed to assess the effectiveness of the proposed documentation process in terms of extracting accurate information from the source code and representing it in UML diagrams, as well as its usability in aiding system comprehension. To achieve this, we conducted a survey involving



software engineers from the embedded systems domain. The survey aimed to answer the following Research Questions (RQs):

1. *RQ1 - To what extent the recovered software architecture documentation is considered accurate by the practitioners?*
2. *RQ2 - If important information is missing, what kind of information is it?*
3. *RQ3 - To what extent have the practitioners found the proposed software architecture recovery (SAR) tool useful?*

**Interviewees selection.** We recruited 14 embedded software engineers currently employed in Micron where the case study was conducted. Each participant has been with the company for at least three years and is experienced in working on software documentation and development tasks. Additionally, all the engineers understand the principles of software architecture documentation and are familiar with using UML notation.

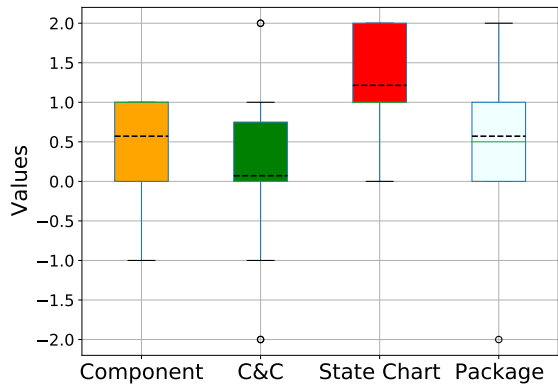
**Questionnaire Design.** The 14 participants were surveyed through a questionnaire we appositely designed to answer the three RQs. The questionnaire was organized into three sections, to evaluate the accuracy and comprehensibility of diagrams generated by the tool, along with evaluating the tool's usability. An additional fourth section was introduced to collect data about the strengths and the limitations of the tool. It consisted of 30 questions, mainly closed-ended, with respondents given the opportunity to provide additional feedback through open-ended questions. The closed-ended questions were measured using a 5-point Likert scale. The questionnaire was implemented in Microsoft Forms. The data analysis relied on both quantitative and qualitative methods to answer the research questions.

**Survey execution.** We conducted a preliminary training phase with all the participants, by executing two focus group sessions, each one lasting two hours. The first focus group session introduced the participants to the tool and included small reverse engineering tasks to familiarize them with its use. Three days later, the second focus group session was devoted to exploring more in detail the tool's features and addressed any doubts that

---

arose during the reverse engineering tasks performed in the first session. Following this training, participants were tasked with a software documentation recovery exercise to be executed with the tool. Each task involved two components of the system they had previously manually designed and implemented. All components were of medium complexity according to the company standards, and each respondent analyzed two different components. We assigned them one hour to complete the task. After that, the respondents filled out the Microsoft Forms questionnaire.

**Answer to RQ1 - To what extent the recovered software architecture documentation is considered accurate by the practitioners?** The box plot in Figure 5.9 shows the accuracy distribution for the diagrams reconstructed by the tool, based on the responses from practitioners. Higher values indicate a good perception of accuracy. From the chart, we can observe that the distribution of the State Chart Diagram is skewed on the higher part of the graph, indicating that it is considered to be reconstructed with a high level of accuracy.



**Figure 5.9.** Box plot showing the accuracy distribution of diagrams reconstructed automatically by the tool

Both the Component Diagram and Package Diagram have a positive median, suggesting that the accuracy of these two diagrams is generally regarded as good. The C&C diagrams have a distribution concentrated

---

on lower values, and the presence of outliers with low values indicates a negative perception of their accuracy. From the analysis of open-ended answers, we observed that some respondents mentioned that the tool sometimes struggled to accurately identify interfaces, enumeration variables, or functions, which they had to manually add to the diagram. They believe this issue may be attributed to the wide variety of programming styles present in the code.

Our analysis revealed a generally positive perception of accuracy, particularly with State Chart diagrams being deemed the most accurate. Component and Package diagrams were also considered sufficiently accurate, while improvements were identified as necessary for the C&C diagrams.

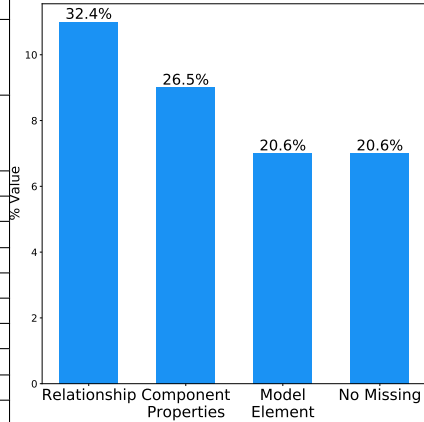
**Answer to RQ2 - If important information is missing, what kind of information is it?** To identify the types of missing information, two authors manually analyzed the responses to the open-ended questions, extracted keywords from the responses, and finally classified the missing information into the four categories shown in Figure 5.10. Another author validated the proposed classification. The histogram rendered in Figure 5.11 shows the distribution of the categories of missing information. In Figure 5.11, the "Relationship" category was identified as the most commonly missing element, reported by 32.4% of respondents. This category highlights the tool's difficulty in capturing interactions within the diagrams, such as the realization relationships between components and interfaces, dependency relationships between provided and required interfaces, and transitions between states. The "Module Structure" category was the next most frequently mentioned, with 26.5% of responses indicating that the tool does not adequately capture structural details of the recovered Components such as methods or attributes. Both the "Model Element" category and "No Missing" were noted by 20.6% of respondents each. These results indicate that about one-fifth of users find shortcomings in how the tool reconstructs system components, packages, and interfaces, while another-fifth see no missing information.

**Answer to RQ3 - To what extent have the practitioners found the proposed software architecture recovery (SAR) tool useful?** The

---

Category	Keyword
Relationship	Realization relationship between component and Interface
	Dependency relationship between component and enumeration
	Dependency relationship between provided and required interfaces
	Transitions between State
	Dependency between package
Component Properties	Method
	Enumeration
	Required Interfaces
	Provided Interfaces
	Attribute Type
Model Element	Interface
	Component
	Package
No Missing	No missing information

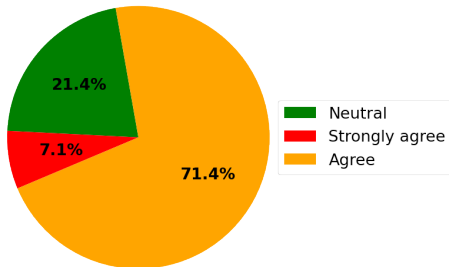
**Figure 5.10.** Identified categories of missing information and their relative keyword



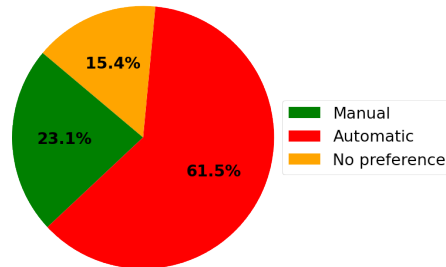
**Figure 5.11.** Distribution of the missing information categories

pie charts in Figures 5.12 and 5.13 display the results of responses to specific questions regarding the tool’s usefulness in aiding the comprehension of the embedded system architecture. As shown in Figure 5.12, the majority of the interviewed participants (71.4%) agreed that the tool is useful in supporting the comprehension of the system. Similarly, as shown in Figure 5.13, a significant portion of the respondents (61.5%) expressed a preference for the models automatically generated by the tool to understand the system architecture. To answer *RQ3*, we found strong evidence indicating a positive response regarding the effectiveness of the tool in facilitating the understanding of architectural elements and relationships. These aspects are often complex and challenging to discern from code alone. However, some respondents remained neutral, suggesting that certain aspects of the reconstructed documentation may not fully meet the needs or expectations of all practitioners. These respondents prefer consulting the source code over UML diagrams for understanding the system’s structure. Others noted that while auto-generated documentation aids in comprehension, it lacks in capturing the system’s “semantics”. These limitations suggest

potential areas for future development, such as implementing customized rules to better suit various working contexts.



**Figure 5.12.** Pie chart on the closed-ended question: “Based on my experience in using the tool, I find it useful in supporting the comprehension of the system.”



**Figure 5.13.** Pie chart on the closed-ended question: “Which reconstructed models, whether generated automatically or manually, do you find most useful for understanding the system?”

**Threats to validity.** Our study faces several potential threats to validity. The primary external threat arises from the fact that the SAR process proposed is tailored specifically to the company programming practices in which the case study was conducted, resulting in a static analysis of the process customized for the C code developed by Micron. We plan to mitigate this threat by extending the case study to other industries developing embedded software systems. Another threat to external validity is the heavy reliance on predefined reverse engineering rules within the EA tool. We intend to mitigate this threat by implementing tool-independent reverse engineering rules. Finally, the narrow scope of our company survey, involving only a small number of participants from a single organization, may also impact the generalizability of the study. To mitigate this threat, we plan to interview more personnel from Micron and expand the survey to include other companies in the embedded systems software development sector.

## 5.6 Conclusion and Future Works

This paper presented a tool-supported Software Architecture Recovery (SAR) process for the automated generation of Software Architecture Documentation (SAD) from C code in embedded systems. The evaluation was conducted via a questionnaire with 14 industry professionals, demonstrating the tool's utility in enhancing documentation practices and providing significant insights into its performance and areas for improvement. In conclusion, our tool demonstrated to be accurate in reconstructing documentation and supporting system comprehension, addressing critical challenges within the embedded systems industry.

We've identified several areas for future work based on the insights from this initial study. To enhance the applicability of our tool, we'll extend the SAR process to other companies in the embedded software industry, including those using different programming languages, ensuring adaptability to diverse development environments. We also plan to develop independent reverse engineering rules to reduce reliance on EA-specific ones, making the SAR process more versatile across various software development contexts. Additionally, we'll expand our industrial survey to include more participants from the surveyed company and to encompass additional companies, gathering broader feedback to refine the tool and better cater to different organizational needs.

---

# Characterizing Software Architectural Metrics for Continuous Compliance in the Automotive Domain

**Abstract** The software of critical systems, such as automotive, is increasingly required to change and evolve after production. In the automotive domain, this is a consequence of self-driving and connected cars, which continuously collect data from the field that is then exploited to produce safer and more advanced and reliable versions of the used algorithms or AI modules. Consequently, there exists a need for techniques and tools to facilitate incremental and Continuous Compliance with safety and security standards.

This paper focuses on software architectural metrics that can be used for Continuous Compliance in the automotive domain. Our initial stride involved a literature review to find metrics capable of assessing software architectures. Subsequently, in collaboration with architecture, safety, and security experts in the automotive domain, we proposed a framework defining the characteristics these metrics must possess for continuous evaluation of software architectural compliance. The framework was used to characterize 48 metrics gathered from the literature review and to associate them with a score expressing their suitability to be used in software architecture

Continuous Compliance processes.

## 6.1 Introduction

The automotive domain lives a revolution guided by business goals and technology drivers [52, 26], such as self-driving, connected vehicles, and over-the-air software updates [38]. The most advanced original equipment manufacturers (OEMs) often define themselves as software companies [11]. They are selling to their customers the possibility to have new functionalities and features on their cars over time via over-the-air (OTA) software updates [143, 4, 7]. The possibility of performing OTA updates of software is also an important safety and security instrument, e.g., to push, as soon as possible, new versions of algorithms that fix security or safety issues or that reduce the risks of accidents. However, OTA updates are also risky and can easily compromise the functioning of vehicles. This is the case of the problems recently caused by an OTA update to Rivian’s vehicles due to a fat finger where the wrong build with the wrong security certificates was sent out: the 2023.42.0 update fails and unintentionally soft-bricks the infotainment systems of R1T and R1S all-electric trucks.<sup>1</sup> Even though the update did not brick the vehicles completely (only the entertainment system is affected), this example clearly explains the need for mechanisms and instruments to avoid undesired, costly, and potentially dangerous side effects of software updates.

The possibility of performing OTA updates after production and when the systems are already used in the operations environment poses questions on the software certification process. The typical process of certifying software before production becomes obsolete and new approaches of incremental and Continuous Compliance to safety and security standards are necessary [163, 57, 126, 129]. This is a challenging and complex problem that touches various facets from technical aspects to processes and organizational ones [163].

In this paper, we aim to contribute to this problem by identifying and characterizing architectural metrics that can be used for Continuous Compliance. To give more concreteness to the results, we focus on the automotive domain. As a first step, we study the literature to extract and identify

---

<sup>1</sup><https://insideevs.com/news/696177/rivian-ota-update-softbricks-evs/>

---



metrics that can be potentially used in Continuous Compliance processes in the automotive domain. As a parallel activity, we build a framework to evaluate metrics and characterize those that can be used in Continuous Compliance processes. This framework is built in collaboration with some experts in safety and security certification and software architectures. We then applied the framework to the list of collected metrics to identify and characterize 48 metrics that, according to our study, can be used in Continuous Compliance processes.

Summarizing, the main contributions of this paper are:

1. at the best of our knowledge this is the first work proposing a set of metrics that could be applied to evaluate the Continuous Compliance of software architectures in the automotive domain;
2. a general framework, defined in collaboration with practitioners for characterizing a metric from the point of view of Continuous Compliance. The framework points out the main attributes and the possible values these attributes can assume;
3. an evaluation system, based on the framework's attributes, to evaluate a score indicating the suitability of a metric for being applied in Continuous Compliance processes.

The paper is structured as follows. Section 6.2 provides background information and compares the work with related works. Section 6.3 describes the methodology used in this study. Section 6.4 discusses and shows the results of the analyses on the metrics that have passed the selection process. Section 6.5 reports threats to the validity of the study and, finally, the paper concludes in Section 6.6 with final remarks and directions for future works.

## 6.2 Background and Related Studies

The concept of Continuous Compliance for a safety-critical domain, such as the Automotive one, was defined to manage safety in an environment increasingly subject to safety standards and software updates [164, 156]. The importance of software architectures for the development of complex and safety-critical software systems such as Automotive is universally

---

recognized [99]. Over the years, numerous metrics have been proposed to calculate specific aspects of the quality of architectures [170]. Since, to the best of our knowledge there are no works on architectural metrics for Continuous Compliance, in this section, we explore the state of the art regarding Continuous Compliance, in Section 6.2.1, and metrics for architectures, in Section 6.2.2.

### 6.2.1 Continuous Compliance

Scientific literature is scarce on Continuous Compliance in the automotive domain. To the best of our knowledge in the automotive domain, there are only two articles that discuss Continuous Compliance. In the first [164] the authors discuss the challenges and problems that led to the development of the Continuous Compliance concept. In the second article [156] the authors focus on the main applications of Continuous Compliance in the automotive domain and then give a formal definition of the term Continuous Compliance. Finally, they provide an overview of the Continuous Compliance process applied to the automotive domain.

The recent literature on Continuous Compliance in general domains aims to ensure adherence to security, safety, privacy, and internal guidelines. In the article [70], Filepp *et al.* developed a framework called “Continuous Compliance”. This tool aligns teams with the company’s policies and best practices while identifying anomalies. D’Alessandro *et al.* [57, 126] discuss the economic problem companies face if they violate privacy requirements. The work in [57] introduces Continuous Compliance Assurance (CCA), a module that detects privacy anomalies in XML messages. In the second article, CCA is tested and confirmed effective in various use cases.

Cheng *et al.* [49], propose a solution to the inefficiency and high cost of the manual compliance acquisition process for security in software systems. Their approach is based on ontologies, natural language processing, heuristics, and secure systems development lifecycle to achieve Continuous Compliance with standards and regulations. The framework they suggest harmonizes existing ontologies and defines test cases to be run continuously. However, the natural language process engine for populating the ontology struggles with expressing itself effectively.

In their article, Moyon *et al.* [129], argue that as agile development

---

methods continue to advance, ensuring security and compliance becomes increasingly important, given that security and compliance techniques have traditionally evolved linearly. To address this issue, the authors propose a methodology that enables continuous and safe development by mapping standard security requirements into agile models. This methodology involves three main steps: (i) creating different visual models that comply with the standards, (ii) evaluating these models using natural language descriptions, and (iii) using the models to verify completeness, correctness, and consistency.

At the *Continuous software engineering: Challenge areas and frameworks* workshop [103], several speakers discussed the challenges faced by software systems like adaptive systems, microservice-based systems, mobile applications, IoT applications, and cyber-physical systems. These systems have multiple components that need to be adaptable, reconfigurable, and reusable while maintaining specific performance levels, reliability, stability, security, and compliance. The authors also addressed key challenges for monitoring and Continuous Compliance, including determining when and what to monitor, interpreting monitored data, and adjusting monitoring intensity.

The work of Dännart *et al.* [63] stresses the need to comply with security standards to gain customers' trust. They want to comply with the IEC 62443 standard but face the challenges of ensuring security in an agile environment. They propose an assessment methodology for agile security-compliant processes, based on the SAFe and IEC 62443-4-1 models, and define quality artifacts from levels 1 to 4. They create a compliance matrix by combining quality artifacts with process maturity levels. They evaluated their work by conducting expert interviews.

In their publication [145], Phipps and Zacchiroli present a novel challenge regarding the upkeep of open-source code in software systems by guaranteeing its Continuous Compliance. Although regular checks are performed, automating and ensuring Continuous Compliance requires the use of several tools. The authors propose the implementation of linked tools to achieve this goal.

In [50] Cheng *et al.*, proposes an automatic method for complying with increasing numbers of requirements, focusing on security improvements to existing ontologies. The paper presents a framework for Continuous

---

Compliance monitoring of multiple requirements documents, with flexible and customizable audit and validation using test cases and unit testing frameworks.

Bicaku *et al.* [31], address the increasing challenges of compliance with standards due to globalization and digitization of industrial systems. They provide a set of guidelines and standards for best practices and offer a metric template for compliance verification. Additionally, they present a prototype of the standard compliance monitoring and verification framework for an Internet of Things industrial use case.

### 6.2.2 Metrics for Architecture

Concerning metrics for software architectures, the literature offers numerous publications. The state of the art in this field is summarized in a literature review [54], where Coulin *et al.* highlight the importance of metrics for the qualification of architectures as *“In Software Engineering, early detection of architectural issues is key. It helps mitigate the risk of poor performance and lowers the cost of repairing these issues. Metrics give a quick overview of the project which helps designers with the detection of flaws or degradation in their architecture.”*. A more recent work by Silva *et al.* surveys quality metrics in software architecture [170]. The work brings together all the metrics that are available for software architectures over the last 12 years. The article lists all the metrics with their description, they are then classified based on their application domain (general, specific) and, based on whether they are internal (regarding the software architecture) or external (regarding other stakeholders). In total, this work surveys 52 metrics relating exclusively to software architectures and 38 quality attributes.

## 6.3 Research methodology

In this section, we present the methodology we followed *to characterize the metrics that could be used for the Continuous Compliance of software architectures in the automotive domain*. The goal has been broken down into the following two research questions:

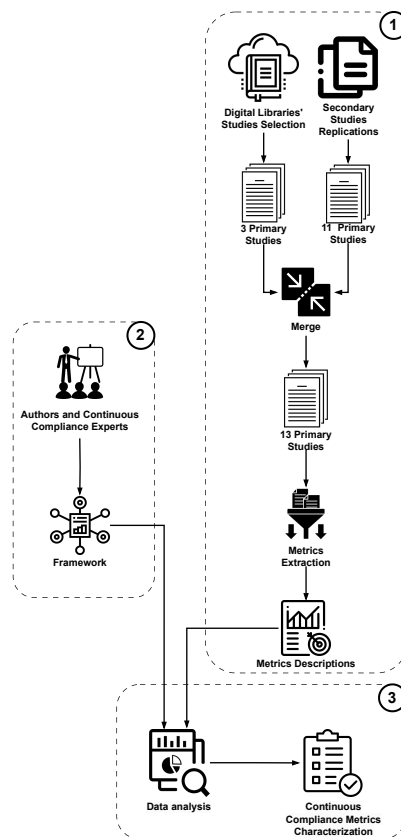
RQ1: *Which are the architectural metrics proposed in the literature that*

---

*can be used in the Continuous Compliance of automotive software architectures?*

RQ2: *How can these metrics be characterized?*

The adopted methodology is represented in Figure 6.1. As the figure shows the methodology relies on the execution of the three steps described in the following.



**Figure 6.1.** Overview of the research methodology

### 6.3.1 Step 1: Metrics gathering and description

The first step was executed to build a set of architectural metrics that have been proposed in the literature. The metrics were extracted from a set of primary studies that were retrieved by combining two sources of information. As shown in the box ① of Figure 6.1, 3 primary studies were selected from digital libraries, and 11 works were obtained by replicating secondary studies. These 14 primary studies were merged, the duplicated ones were removed, and the final set of 13 primary studies listed in Table 6.1 was obtained. Figure 6.2 shows the publication years distribution of the selected papers. From the figure emerges that more than 60% (8/13) of the selected studies have been published in the last 5 years, and almost 92% (12/13) of them was published in the last ten years. This gives evidence of the growing interest of the research community in measuring the conformance of software architectures in the automotive domain.

Each primary study of the set was analyzed by one researcher who extracted the metrics it presents and for each metric also extracted sentences describing them. Moreover, the researcher extracted sentences providing shreds of evidence on how the metrics can be evaluated and how these metrics were applied in case studies or experiments. This activity produces the Metrics Descriptions document collecting the evidence extracted from the primary studies.

Authors applied the following six inclusion criteria (IC) when selecting the documents and deciding on their pertinence to the research scope:

- IC<sub>1</sub>: the primary study discusses KPI, SPI, or metrics for evaluating the compliance of software architecture;
- IC<sub>2</sub>: the study is in the context of the automotive domain;
- IC<sub>3</sub>: the work is written in English;
- IC<sub>4</sub>: the work is not a secondary or tertiary study;
- IC<sub>5</sub>: the study can be downloaded;
- IC<sub>6</sub>: the primary study is not a duplicate of other selected works.

Two researchers reviewed each candidate source and voted on applying the inclusion criteria first to the title and abstract and then to the full

---

paper reading. Each researcher classified each information source into Accept, Doubt, or Exclude. To be included in the sample, documents must have at least one Accept and one Doubt vote. A third researcher was involved in discussing the inclusion or exclusion of studies with two Doubt votes.

The processes we followed to select the primary studies from the information sources relied on the execution of the two activities briefly described in the following.

**Digital Libraries' Studies Selection** in this activity we executed at the June 30<sup>th</sup> 2023, the two search strings reported below in the Scopus and the Web of Science (WoS) digital libraries respectively. We got a total of 147 primary studies, of which 81 were returned by Scopus and 66 by WoS. By applying the inclusion criteria 3 studies overcame the selection.

#### Scopus Search String

```
TITLE-ABS (( compliance OR adherence* OR conformance*) AND (kpi*
OR spi OR metric OR metrics OR indicator OR indicators) AND
(automotive OR car OR cars OR vehicle OR vehicles OR *26262 OR
ASPICE OR *24089 OR *21434)) AND (LIMIT-TO (SUBJAREA,"COMP"))
```

#### WoS Search String

```
(TI=(compliance OR adherence* OR conformance*) OR AB=(compliance
OR adherence* OR conformance*)) AND (TI=(kpi* OR "spi" OR "metric"
OR "metrics" OR "indicator" OR "indicators" ) OR AB =(kpi* OR "spi"
OR "metric" OR "metrics" OR "indicator" OR "indicators" )) AND
(TI=("automotive" OR "car" OR "cars" OR "vehicle" OR "vehicles" OR
26262 OR "ASPICE" OR 24089 OR 21434) OR AB = ("automotive" OR "car"
OR "cars" OR "vehicle" OR "vehicles" OR 26262 OR "ASPICE" OR 24089
OR 21434))
```

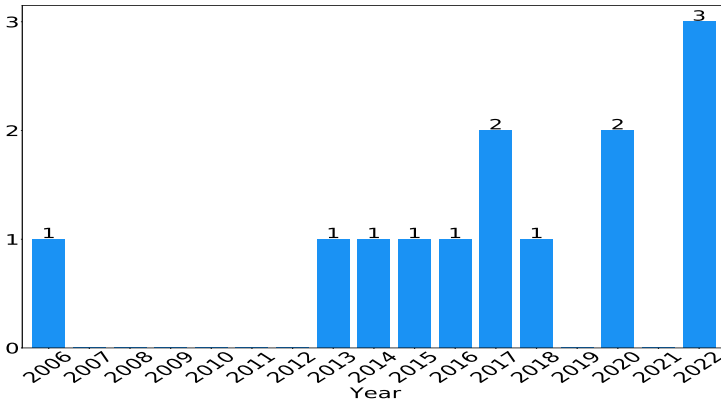
**Secondary Studies Replications** in this activity the literature surveys presented by Silva *et al.*[170] and Vogel *et al.* [198] were replicated. To replicate each study two steps were executed. Firstly, the defined IC were applied to the primary studies already surveyed by the two works. In this way 2 over the 33 primary studies already analyzed by Silva *et al.*[170] were selected, and 6 of the 38 primary studies analyzed by Vogel *et al.*[198] were included. Secondly, the same selection processes followed by the two

secondary studies were replicated with two main differences: *i)* the inclusion and exclusion criteria used in the studies were substituted by the ones defined in this work and *ii)* the search strings, used in these works, to query the digital libraries were limited to dates after those in which the studies were performed. No new primary studies were obtained by replicating the selection process presented in [170], meanwhile by rejuvenating the survey [198], stuck at 2018, 6 new primary studies were included as stated by the 219 works returned by executing the search strings.

**Table 6.1.** Selected Primary Studies

ID	Title	Year	Ref
PS1	A viewpoint-based evaluation method for future Automotive Architectures	2022	[86]
PS2	Performance Assessment of Traditional Software Development Methodologies and DevOps Automation Culture	2022	[134]
PS3	Measurements of Support Processes: Proposed Improvements on Automotive SPICE PAM V3.1 in Light of OEM Standard Supplier Quality Requirements	2022	[128]
PS4	Playground for Early Automotive Service Architecture Design and Evaluation	2020	[42]
PS5	Towards Metrics for analyzing System Architectures Modeled with EAST-ADL	2020	[66]
PS6	Improved Technique for Measuring the Number of Defects in Automotive Agile SW Development Defect debt trend	2018	[154]
PS7	Traceability Metrics as Early Predictors of Software Defects?	2017	[135]
PS8	Cyber-Physical Codesign at the Functional Level for Multidomain Automotive Systems	2017	[200]
PS9	Metrics design for safety assessment	2016	[111]
PS10	Metrics for Verification and Validation of Architecture in Powertrain Software Development	2015	[195]
PS11	On Bringing Object-Oriented Software Metrics into the Model-Based World – Verifying ISO 26262 Compliance in Simulink	2014	[131]
PS12	Measuring the impact of changes to the complexity and coupling properties of automotive software systems	2013	[62]
PS13	Process Family Points	2006	[100]





**Figure 6.2.** Distribution of the selected primary studies per publication year

### 6.3.2 Step 2: Framework definition

This step aimed to identify the key characteristics a metric needs to have to consistently meet compliance with the standards continuously.

As shown in the box ② of Figure 6.1, this step was performed by the authors in collaboration with experts in Continuous Compliance to define a Framework that could be used *i)* to characterize software architecture metrics from the point of view of Continuous Compliance and *ii)* to evaluate the suitability of a metric to be applied in Continuous Compliance processes.

To build such a framework we surveyed three experts selected from our lists of industrial collaborations. All the interviewees had more than 10 years of experience in the safety and security compliance field for software architectures in the automotive domain. One expert works for an automotive original equipment manufacturer (OEM), and the other two work for two automotive supplier companies. Each expert has been interviewed individually through semi-structured interviews, following the guidelines outlined by Wilson *et al.* [205].

Each interview lasted 2 hours and allowed the participants to freely discuss and elaborate on these topics:

1. Identifying the key characteristics of a metric for the Continuous

Compliance to define the framework's Dimension.

2. Defining the set of values that can be assigned to each dimension of the framework.
3. Define an evaluation criteria, based on the above-defined dimension, for assessing the suitability of a metric to be used in the context of Continuous Compliance.

Two of the authors were responsible for collecting and interpreting the responses. Their goal was to abstract the main characteristics of a metric, as identified by the practitioners, and to identify the dimension of the framework. Furthermore, they aimed to delineate how the different values that can be assumed by the different dimensions can be used to assess the effectiveness of the metric in continuously supporting the compliance process.

From the surveys, we build the Framework shown in [Figure 6.3](#). It was validated in a final 2 hours meeting that involved all the authors and the three interviewed experts.

As [Figure 6.3](#) shows, the framework features eight dimensions. Five dimensions were indicated by the experts as representative for evaluating the suitability of a metric to be introduced in Continuous Compliance processes. For each one of such dimensions, it has also been possible to abstract the possible values these dimensions must have. Moreover, to evaluate the suitability of a metric to be used in the Continuous Compliance context, we developed an evaluation system based on the framework's dimension. Based on the guidance provided by the experts during the survey, each value of the different framework dimensions has been assigned a score reflecting their relevance. A '++' score is assigned to values that have been indicated as *very significant* for Continuous Compliance, whereas, the '+' score refers to values that have been considered as *significant* by the experts. Conversely, to values that were considered as *irrelevant* for Continuous Compliance, it was assigned a "neutral" value, indicated with '○'.

Moreover, the last three dimensions of the framework do not directly influence the evaluation of metrics' suitability for Continuous Compliance, experts have highlighted their importance in offering a more detailed description of the metrics. This encompasses both the property measured by

---

Metric	Assessment approach	Input Artifact Type	Metric Description Type	Experimental Settings	Experimental Objects Type	Measured Property	Output Category	Application Field	Score
Average Output Interface Size	Methodology (Ø)	Architecture (Ø)	Formula (+)	Instrudial (+)	Proprietary (Ø)	Complexity	Component	Hybrid control Software	++
System Risk Identification	Tool (++)	Architecture (Ø)	Formula (+)	Academic (Ø)	Open Source (+)	Coverage	Process	DevOps	++++

**Table 6.2.** Example of metrics description and evaluation according to the proposed Framework

the metric and its application context, providing valuable insights to characterize metrics from the perspective of their application contexts as well. Each dimension of the framework and its values and score are described in the following.

### *D1 - Assessment Approach*

outlines the technique used in the analyzed paper to evaluate the metric. When a metric is assessed within a *Methodology* it means that is provided with a high-level conceptual structure and description lacking in explicit guidance or steps for the adoption of the metric. In contrast, a *Framework* approach means that the study in which the metric is presented provides well-defined and detailed assessment procedures like an algorithm or step-by-step process. Finally, a metric can also be assessed using a *Tool*. The availability of a *Tool* was positively assessed by the experts, who assigned a score of ‘++’ to this value, as the presence of a tool suggests that the metric has already been applied in a practical solution. Otherwise, if a metric assessment approach was based on a *Framework*, it received a ‘+’ score, reflecting its lower impact on the metric suitability for Continuous Compliance. Methodological assessment approaches did not receive favorable evaluations from experts and were consequently assigned a ‘neutral’ value, indicated by ‘Ø’.

### *D2 - Input Artifact Type*

identifies the artifacts required as input for calculating the metric. *Architecture* artifacts encompass information that is available in the software architectural design (SAD) documentation, such as static or dynamics

view. For the metric that only requires information that resides in the source code, we have outlined *Code* as input artifact. In some instances, the information required as input by the metrics may be found in other types of artifacts used in different phases of the software development life like analysis or deployment for which the value *Requirement* and *Deployment* are designed. Lastly, metrics can also take into account information that is obtained at the end of the testing phase and is identified as *Test* artifacts. For this dimension, the *Code*, *Test*, and *Deployment* values received a ‘+’ valuable score by the experts. This decision was based on the expert opinion that code, test, and deploy artifacts usually are always available and update in an iterative development context. In contrast, metrics utilizing architectural artifacts as input were deemed unsuitable by the experts for the context of Continuous Compliance. One motivation was that artifacts related to software architectural views might not be consistently updated across different development iterations, due to architectural degradation [184] and lack of automated software architectural documentation recovery tools.

### *D3 - Metric Description Type*

specifies whether the metric is described by a *Formula* or by a detailed *Textual* description, as metrics without a description were excluded in the early stages of our work. The availability of a *Formula* has been scored with a ‘+’, whereas the *Textual* description has been scored with a ‘⊙’.

### *D4 - Experimental Setting*

The experimental setting in which the metric was applied is divided into two categories. Metrics with *Industrial* value have been introduced in studies conducted in collaboration with industrial partners, specifically in the automotive field, and were utilized in a real-world context with actual data. For this reason, the *Industrial* values are assigned with a ‘+’ score. Instead, *Academic* metrics, assigned with the ‘⊙’ have been introduced in studies not conducted in collaboration with the industrial partner.

---

### *D5 Experimental Objects Type*

specifies the nature of the data employed in the experimental setting. The dimension can assume only two different values: *Open Source Data*, assigned with a ‘+’ score, denotes data that are publicly available in a dataset. The *Proprietary* values, on the other hand, refer to data that are not accessible to the public and are assigned with a ‘ $\ominus$ ’ score.

### *D6 - Measured Property*

identifies the properties of the software system that are measured using the metric. For this dimension, it is not feasible to define a predetermined set of values. Additionally, the property measured by the metric has not been involved in the introduced scoring procedure. In the Result section, we will discuss the measured properties observed from the analyzed metrics.

### *D7 - Output Category*

represents a grouping of the measured properties identified in the previous dimension. Due to the diverse and variable nature of the measured properties, it is not possible to define a set of possible values by a metric that can be observed. Even in this case, this dimension is not involved in the metric’s characterization phase within the context of Continuous Compliance.

### *D8 - Application Field*

indicates the specific automotive sub-domain where the metric has been implemented or used. This detail offers an indication of the specific context in which the metric was used, allowing a more detailed and targeted analysis of the metric. Determining the specific scope within the automotive environment where the metric has been tested could be useful to understand its impact, relevance, and validity in the domain of interest. Also, in this case, the dimension is not involved in the characterizing phase and is not provided with a pre-defined set of possible values.

---

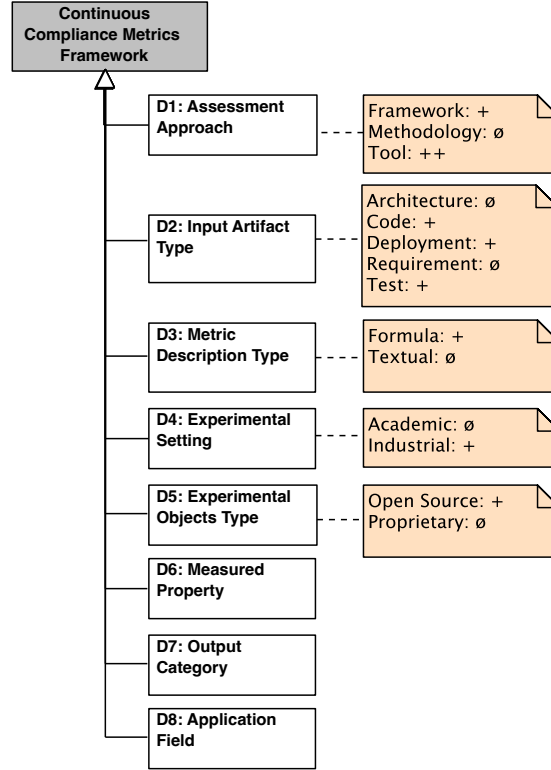


Figure 6.3. The proposed Framework

### 6.3.3 Step 3: Metrics characterization and evaluation

In this step, a Data Analysis activity was executed to obtain a set of Continuous Compliance Metrics Characterizations as shown in box ③ of Figure 6.1. The Metrics Description produced in ① have been analyzed by taking into account the Framework defined in ②. The Data Analysis was performed blindly by two groups, the groups were made by two authors. Each group worked independently to recast all the metric descriptions in accordance with the dimensions and the values of the Framework. The two groups had one-hour weekly meetings to discuss about the analysis results, and, every two weeks a one-hour meeting was conducted with the three experts, interviewed in the ②, to validate the results. The Data

Analysis step lasted two months, and the final Continuous Compliance Metrics Characterizations are available in the supplemental material.

The evaluation system we proposed rates a metric by concatenating the ‘+’ symbols of its attributes’ values.

Table 6.2 shows an example of how two metrics, extracted from our sample, have been described according to our framework. As the table shows, the “Average Output Interface Size” metric reaches a score of ‘++’ since it assumes the values *Formula* and *Industrial* for the *Metric Description Type* and *Experimental Settings* domains, respectively. Similarly the “System Risk Identification” has been evaluated with ‘++++’ obtained by concatenating the ‘+’ symbols of the *Tool*, *Formula*, and *Open Source* values.

## 6.4 Results and Discussion

In this section, we discuss and show the results of the analyses on the metrics that have passed the selection process. Thanks to the opinion of the experts we were able to carry out 8 analyses, that are mapped to the dimensions of the framework, which we will explain in detail in the following sections.

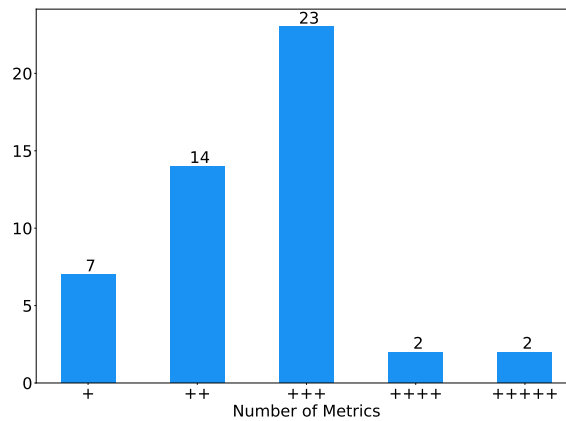
### 6.4.1 RQ1: *Which are the architectural metrics proposed in the literature that can be used in the Continuous Compliance of automotive software architectures?*

The evaluation system we proposed allowed us to assess the suitability of each selected metric in Continuous Compliance processes. Table 6.3 summarizes the evaluation results we obtained. More in detail, the table shows for each analyzed primary study, the metrics we extracted from it along with the Continuous Compliance score we evaluated. Moreover, the plot shown in Figure 6.4 represents the distribution of the metrics based on the obtained scores.

From these results, we have evidence that no metrics reached the maximum value allowed by the proposed evaluation system, i.e., six ‘+’ (‘+++++’). The data we obtained also show that only 4 metrics over 48 ( $\sim 8\%$ ) are almost ready for being applied in Continuous Compliance

since they were rated with four or five '+'. These metrics were extracted from the primary study PS2 [134]. These metrics have not reached the maximum value since they were not applied in an industrial setting and two of them need architectures or requirements as input artifacts. The primary reason these two metrics achieved such scores is their automatic evaluation due to a tool support [133]. Additionally, they earn extra positive evaluations due to their presentation as formulas, the use of open source data as experimental objects type, and their utilization of input artifacts from the Deployment and Test phases.

On the other hand, 21 metrics over 48 ( $\sim 4\%$ ), are far, or very far, from their application in a Continuous Compliance process since they were scored with one '+' (7/48) or two '+' (14/48). Lastly, 23 metrics ( $\sim 48\%$ ), have been evaluated with three '+'. We believe that these metrics could become suitable for Continuous Compliance with a little extra effort. Indeed, we observed that most of these metrics lack a tool supporting their automatic evaluation, even if they propose a clear framework explaining how they should be applied in practice. The implementation of tools able to execute these frameworks may lead to additional metrics ready for being used in Continuous Compliance processes.



**Figure 6.4.** Distribution of the metrics over the Continuous Compliance Evaluation Score



**Table 6.3.** Continuous compliance Metrics Evaluation

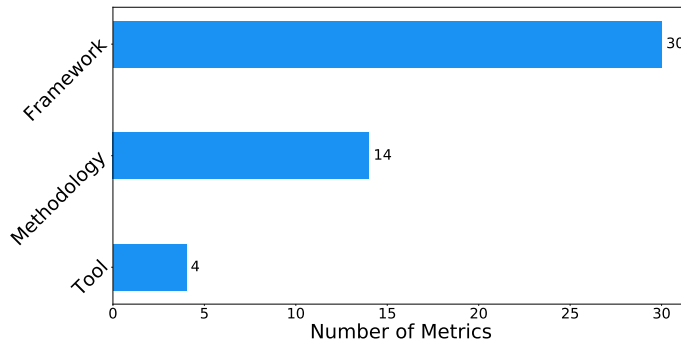
ID	Metric's Name	Evaluation Score
PS1	Average Input Interface Size of Components	++
	Testability Metric	++
	Cost Function	++
	Coupling of Components	++
	Score View	++
PS2	Project Defect Density	+++++
	Release Deployment Frequency	+++++
	Process Productivity	++++
	System Risk Identification	++++
PS3	Software Requirement Coverage	++
	Software Static / Dynamic Code Analysis Coverage Metrics	+++
	Implemented Software Requirements	+
PS4	Service Group Interface Count	+++
	Service Group Local Interface Count	+++
	Service Group Interface Exposure Degree	+++
	Service Group Exposure Count	+++
	Service Group Required Interfaces Count	+++
	Service Group Required Groups Count	+++
	Service Groups Dependency Intensity	+++
PS5	Service Group Reallocation Capacity	+++
	Coupling between Object Classes	+
	Response for a Class	++
	Coupling Factor	+
PS6	Clustering Factor	+
	Defect Debt Trend	+++
PS7	Traced Components per Requirement	+++
	Traced Requirements per Component	+++
PS8	Cost of the architecture (perspective)	++
	Cost of the architecture (robustness)	++
PS9	Functional Safety Requirements and Safety Goals	++
	Number of functional safety requirements	++
PS10	Average Function Interaction within a Component	+++
	Number of Requirements associated with a Software Component	+
	Number of Functions per Software Components	+
	Average Input Interface Size	++
	Average Input Interface Size	++
PS11	Number of Elements	++
	Element Hiding Factor	+++
	Fan In / Fan Out	+++
	Range of Demeter	+++
	Halstead Volume	+++
	Tight Block Cohesion	+++
	Loose Block Cohesion	+++
PS12	Single Component	+++
	Component Complexity	+++
	Package Coupling Coup	+++
	Package Coupling Metrics	+++
PS13	Process Family Points	+

#### 6.4.2 RQ2: *How can these metrics be characterized?*

Thanks to the proposed framework, it was possible to characterize the selected metrics from the point of view of Continuous Compliance. In the following, we describe the results obtained for each dimension of the framework, showing both aggregate views and brief discussions of the results.

## D1: Assessment Approach Analysis

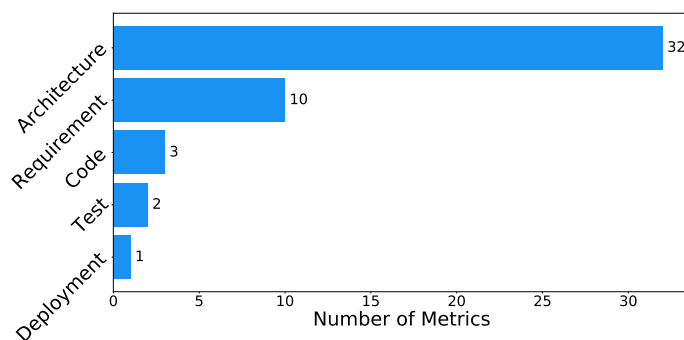
Figure 6.5 shows the distribution of the metrics over the values they assume for the  $D_1$  *Assessment Approach* dimension. Most of the metrics, 30 out of 48, were evaluated within frameworks. In this category, we find primary studies in which there are algorithms and well-structured processes from which step-by-step procedures can be abstracted to obtain the metrics. This allows us to say that these identified metrics are described within well-defined processes and therefore can be used with little effort. Instead, 14 out of 48 metrics are presented within a *Methodology* without providing specific information on the steps to be carried out to evaluate the metrics. In this context, we identified several case studies that offered interesting examples of metric applications. However, these studies lacked comprehensive and clearly defined procedures for the implementation of the metrics. Finally, 4 out of 48 metrics are already ready to be used as they are supported by a tool featured in the work of Narang *et al.* [133]. This result from the perspective of Continuous Compliance shows us that there is currently a lack of tools for calculating metrics in the literature but at the same time, the community is working towards the definition of well-defined and concrete procedures.



**Figure 6.5.** Distribution of the metrics over the *Assessment Approach* domain of the Framework

## D2: Input Artifact Type Analysis

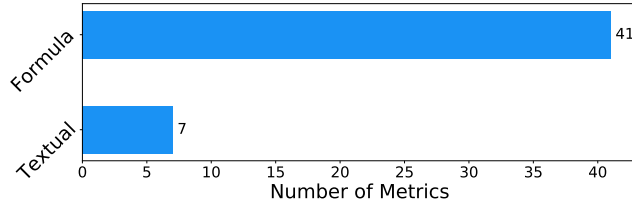
The diagram in Figure 6.6, shows how the metrics are distributed on the different types of artifacts required as input. This figure indicates that most of the metrics, 32 out of 48, take architectural artifacts as input. In this category, we observed both the use of high-level architectural views, showing the components and the relationships among them, as well as more detailed architectural views focusing on individual components. This result reflected the observation that most of the analyzed metrics, as can be seen in the following, evaluate properties related to cohesion and coupling between components. To evaluate these properties, the metrics require two inputs: information about the components and the relationship between them, which resides within the documentation of the software architecture. In the context of this study, most of the metrics under analysis take architectural artifacts as input. Although standards require such architectural views, and some papers in the literature call for solutions to produce them in compliance with standards [17], experts considered these artifacts as inadequate to meet the requirements of Continuous Compliance. This inadequacy restricts the practical adaptation of these metrics in the context of Continuous Compliance. The second occurrence of input artifacts concerns the requirements, with 10 metrics out of 48. In terms of code, testing, and deployment, our analysis revealed that only a limited number of metrics provide architectural insights based on these elements. Specifically, just 6 out of 48 metrics are derived from these artifacts.



**Figure 6.6.** Distribution of the metrics over the *Input Artifact Type* domain of the Framework

### D3: Metric Description Type Analysis

As Figure 6.7 shows, most of the metrics, 41 out of 48, have been presented with a mathematical formula. The remaining 7 metrics were described through detailed textual descriptions.

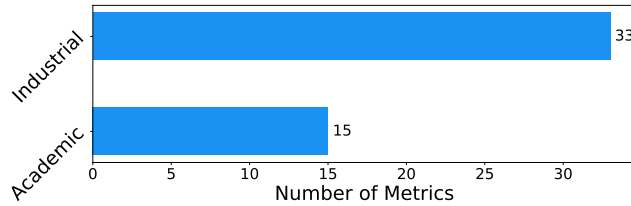


**Figure 6.7.** Distribution of the metrics over the *Metric Description Type* domain of the Framework

### D4: Experimental Setting Analysis

The plot rendered in Figure 6.8 gives evidence that most of the metrics (33/48) have been experimented in industrial settings. The industrial partners we identified are from the automotive sector, both suppliers and OEMs. The remaining metrics were used exclusively in the academic field.

The finding that most of the metrics were used in real contexts with real data in collaboration with industrial partners suggests that interest in the practical application and evaluation of metrics is high in the industry.

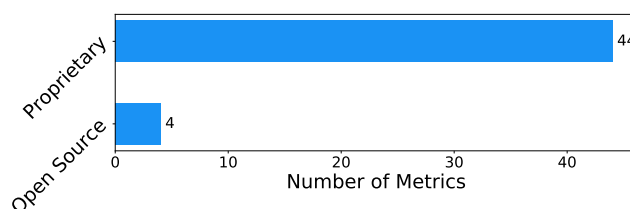


**Figure 6.8.** Distribution of the metrics over the *Experimental Setting* domain of the Framework

---

### D5: Experimental Objects Type Analysis

The distribution of the metrics over the *Experimental Object Type* dimension is illustrated in Figure 6.9. As rendered in the diagram, most of the metrics (44 out of 48) have been evaluated using proprietary data. Only the remaining 4 metrics have been evaluated with open source data. This finding is in line with the observations of the previous analysis, which highlighted that most of the metrics have been experimented in industrial settings where data are often not publicly accessible.

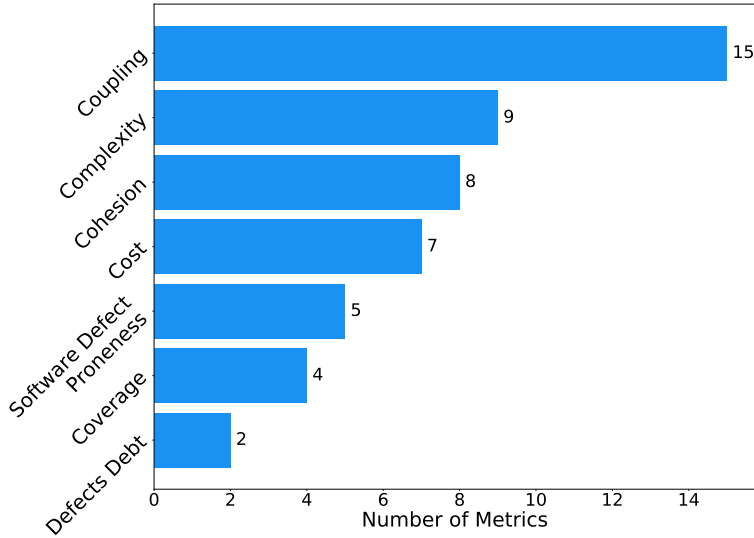


**Figure 6.9.** Distribution of the metrics over the *Experimental Objects Type* domain of the Framework

### D6: Measured Property Analysis

The graph reported in Figure 6.10 shows the distribution of the metrics based on their measured properties. As depicted in Figure most of the metrics focus on measuring system properties like complexity, as well as aspects related to the interaction and relationships between system components, such as cohesion and coupling. This encompasses metrics that assess the average number of functions per interface, Fan In-Fan Out, as well metrics to assess coupling at both the component and package levels. From a strictly standards compliance perspective, it is interesting to observe that there are metrics, categorized with properties related to system defectiveness, such as *software defect proneness* and *defect debt*, two aspects that should be kept under control within iterative development contexts that can be adapted within Continuous Compliance methodologies. In this analysis, the total sum of the occurrences is greater than the 48 metrics because some of them explicitly have two Measured Properties. In particular, the metric "Element Hiding Factor" has complexity and co-

hesion as measured properties, while the metric “Service Group Interface Exposure Degree” has both cohesion and coupling as measured properties.

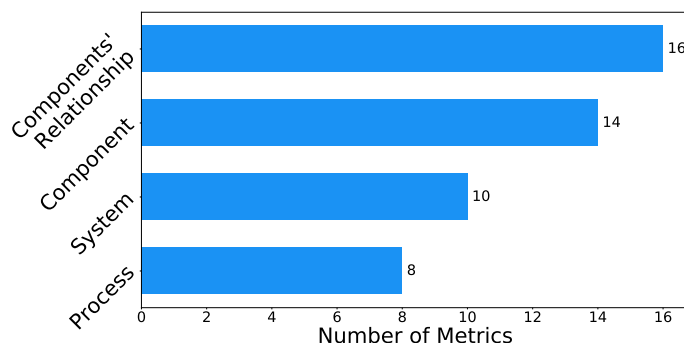


**Figure 6.10.** Distribution of the metrics over the *Measured Property* domain of the Framework

### D7: Output Category Analysis

The bar chart in Figure 6.11 shows how the metrics are distributed on the output types. This analysis can be seen as a high-level abstraction of the values observed for the Measured Property. The value with the highest number of occurrences is *Components' Relationship*. In this group, metrics are focusing on cohesion and coupling like “Loos Block Cohesion” and “Coupling Factor”. Additionally, this category encompasses metrics such as ‘Service Group Dependency Intensity’, which quantifies the intensity of the relationship among components. The identified value *Component* counts 14 out of 48 metric. This subset includes metrics like “Component Complexity” and “Number of Requirements associated with a Software Component”, both metrics that evaluate properties relative to a single component. In the *System* group, we found metrics related to cost

or metrics like the “Range of Demeter” used to evaluate the hierarchical structure of software components. Lastly, within the *Process* group, we identified the “Defect Debt Trend” and the “System Risk identification” both useful for process-oriented evaluation.



**Figure 6.11.** Distribution of the metrics over the *Output Category* domain of the Framework

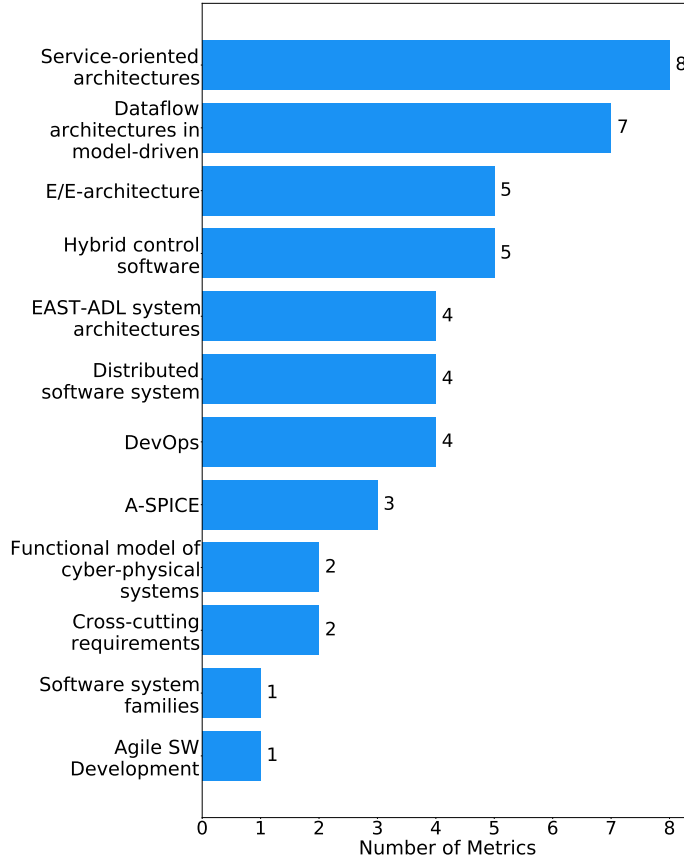
### D8: Application Field Analysis

In Figure 6.12 we render how the metrics are distributed on the application field on which they have been applied. We have observed that each analyzed primary study presents a different application field always concerning the automotive domain. From the point of view of Continuous Compliance, it is interesting to note how some metrics are taken from primary studies that deal with DevOps and Agile techniques within the automotive context. These processes encourage continuous and iterative development and show us how there is interest in the literature in using metrics that can be used to support iterative processes.

## 6.5 Threats to validity

This study encounters threats that pertain to its external and internal validity, each requiring attention and potential mitigation strategies.

---



**Figure 6.12.** Distribution of the metrics over the *Application Field* domain of the Framework

### 6.5.1 External validity threats

The primary challenge impacting the generalizability of the outcomes stems from the limited pool of experts engaged in defining the framework for characterizing continuous compliance metrics. This constrained involvement may have led to a potential lack of additional dimensions that could hold relevance for other practitioners. To address this limitation, an expansion of the approach to include a more diverse set of domain experts could enrich the framework and capture broader practitioner perspectives.



Another concern involves the absence of a validation step for the Continuous Compliant Metrics Characterization. While the results were validated with the same experts involved in defining the Framework, this may not ensure the general applicability of the proposed metrics and their descriptions. To mitigate this, a case study assessing the actual adoption of these metrics within real industrial software development processes would provide valuable insights.

### 6.5.2 Internal validity threats

An internal threat identified in this study revolves around the methodology employed to gather metrics and construct the framework. Metrics were obtained by scrutinizing primary studies from digital libraries and replicating selection processes from secondary studies. To mitigate this limitation and expand the pool of potential metrics, a more formal systematic literature review could be conducted. Additionally, conducting industrial surveys or multivocal literature reviews would support the diversity of collected metrics.

Regarding the internal threat linked to the limited number of experts involved in framing the framework, a proactive step to mitigate this challenge is to plan an empirical study involving a broader sample of experts from various automotive companies, thereby capturing a more comprehensive range of perspectives during the interview process.

## 6.6 Conclusion and Future Works

Continuous Compliance is an approach that can be considered as a useful mechanism to ensure ongoing compliance with standards, best practices, and internal policies, particularly within safety-critical domains like automotive systems prone to Over-The-Air (OTA) updates. Identifying and defining metrics that consistently quantify work quality becomes significant. Collaborating extensively with experts in safety, security, and architecture, we crafted the technical facets of this paper. In this study, we derived a collection of 48 architectural metrics already proposed in the literature for the automotive domain. These metrics have been selected by analyzing 13 primary studies we selected by analyzing 147 works, returned

---

by the Scopus and the Web of Science digital libraries, and 219 studies obtained by replicating two known secondary studies.

These metrics have been characterized following a general framework we defined in collaboration with practitioners. This framework was proposed for characterizing a metric from the point of view of Continuous Compliance in terms of eight specific attributes and the values these attributes can assume. Moreover, each metric was evaluated according to a scoring system we proposed. The system scores each metric based on the values it assumes for some of the framework's attributes. The obtained score allows the assessment of the suitability of a metric for being applied in Continuous Compliance processes.

On the one hand, the results allowed us to have a list of four metrics having high suitability for being adopted in Continuous Compliance processes since they have been rated with five '+' or four '+'. Moreover, an additional set of 23 metrics, rated with three '+', that could become suitable with an extra tool development effort has been also proposed.

On the other hand, as an additional result, we obtained a deep characterization of the analyzed metrics from the point of view of Continuous Compliance.

In future work, we intend to increase the number of experts to be interviewed, and also recruit from a broader and varied sample of automotive companies, to improve the generalizability of the proposed framework.

Moreover, we are planning to perform an empirical study to evaluate the feasibility of the selected metrics in Continuous Compliance processes performed in real industrial automotive settings. Another possible future work is to expand the number of metrics and explore metrics related to other software life cycle phases instead of just focusing on architecture. This is possible both through conducting industrial surveys or multivocal literature reviews, but also by relaxing the constraint of the automotive domain, trying to adapt metrics developed in other domains to the automotive one.

---

# Conclusions

The integration of embedded software into automotive systems has deeply transformed vehicle design and development. Software now plays a pivotal role in controlling nearly every aspect of modern vehicles, from basic engine functionality to advanced safety systems like ADAS. The complexity of automotive software has grown exponentially, driven by increasing consumer expectations and the industry's shift toward autonomous driving technologies. This transformation has brought unprecedented opportunities for innovation, but also significant challenges in terms of software development, quality assurance, and compliance with safety standards.

The growing reliance on software in vehicles has made it critical to manage both the technical and organizational challenges that come with the increased complexity. This thesis has addressed these challenges by exploring several key areas in the automotive software development process. First, it highlighted the importance of collaboration between different engineering disciplines (e.g., mechanical, electrical, and software) underscoring the need for multidisciplinary approaches to manage the intricate interactions between hardware and software. Community detection and expert finding techniques were identified as crucial tasks in ensuring that the right skills are brought to bear on increasingly complex development projects. Our proposed solution of applying community detection within developer social networks presents an innovative way to identify experts and optimize team formation in large and heterogeneous development en-

vironments.

Another key contribution of this thesis is the development of a new software documentation template designed to address the challenges of traceability and maintainability in architectural documentation, specifically within safety-critical domains that must comply with ISO 26262. The proposed template, validated through a case study in collaboration with several automotive companies, provides a structured approach to improving documentation quality, reducing the risk of defects and ensuring long-term maintainability of software systems.

This thesis also introduces a Software Architecture Recovery (SAR) tool aimed at addressing poor documentation practices in the industry, often caused by time and budget constraints. The tool automates the generation of software architecture documentation from code, addressing the gap between design and implementation, which frequently leads to misalignment and higher maintenance costs. The SAR tool was validated through an industrial case study, demonstrating its effectiveness in improving system understanding and documentation accuracy.

Lastly, this research presented a framework for characterizing software architecture metrics to support Continuous Compliance processes. As automotive systems evolve, ensuring ongoing compliance with safety standards, best practices, and internal policies becomes essential. The framework developed in this study identifies and assesses metrics that are highly suited for continuous compliance, providing a foundation for integrating these metrics into real-world industrial processes.

While this research has made significant contributions to addressing the challenges of software development in the automotive domain, there are several areas for future work. First, the community detection approach for expert finding could be further refined by incorporating additional data sources, such as other developer social networks or code repositories beyond GitHub. Extending the validation of this approach to different domains could also provide insights into its broader applicability.

The documentation template and SAR tool could benefit from further integration with other tools commonly used in the software development life cycle, such as requirements management and testing systems.

The framework for software architecture metrics could be expanded to include metrics from other stages of the software development life cycle,

---

such as testing, deployment, and maintenance. This would provide a more comprehensive view of software quality and compliance, enabling a more holistic approach to Continuous Compliance. Furthermore, future research could explore the applicability of metrics developed in other domains to the automotive industry, potentially uncovering new ways to enhance software quality and safety.

Finally, expanding the scope of industrial surveys and case studies to include a wider range of automotive companies and development environments would help validate the findings of this research and improve its generalizability. By continuing to refine and expand the methodologies proposed in this thesis, the automotive industry can better manage the complexity of software development while maintaining the high levels of safety and reliability required for modern vehicles.

---



# Bibliography

- [1] AUTOSAR architecture. [https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR\\_EXP\\_LayeredSoftwareArchitecture.pdf](https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf). [Online; accessed 1-September-2024].
- [2] AUTOSAR Partner. <https://www.autosar.org/about/partners>. [Online; accessed 1-September-2024].
- [3] Honda Admits Software Problem, Recalls 175,000 Hybrids. [https://www.eetimes.com/honda-admits-software-problem-recalls-175000-hybrids/?\\_ga](https://www.eetimes.com/honda-admits-software-problem-recalls-175000-hybrids/?_ga). [Online; accessed 20-August-2024].
- [4] Making Fully Vehicle OTA Updates Reality WP. <https://www.nxp.com/docs/en/white-paper/Making-Full-Vehicle-OTA-Updates-Reality-WP.pdf>. [Online; accessed 01-December-2020].
- [5] Software-Related Recalls and the Auto Industry's Ongoing Evolution. <https://www.wardsauto.com/software-defined-vehicles/software-related-recalls-and-the-auto-industry-s-ongoing-evolution>. [Online; accessed 20-August-2024].
- [6] Iso/iec/ieee systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, 2011.
- [7] Amit Agarwal. Understanding Automotive OTA (Over-the-Air Update), howpublished = "<https://www.pathpartnertech.com/understanding-automotive-ota-over-the-air-update/>".
- [8] Emad Aghajani, Csaba Nagy, Mario Linares-Vasquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. Software docu-

- mentation: The practitioners perspective. page 590 – 601, 2020. Cited by: 16.
- [9] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1199–1210, 2019.
  - [10] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1199–1210, 2019.
  - [11] S. Magnus Ågren, Rogardt Heldal, Eric Knauss, and Patrizio Pelliccione. Agile beyond teams and feedback beyond software in automotive systems. *IEEE Trans. Engineering Management*, 69(6):3459–3475, 2022.
  - [12] Deniz Akdur, Vahid Garousi, and Onur Demirörs. A survey on modeling and model-driven engineering practices in the embedded software industry. *Journal of Systems Architecture*, 91:62–82, 2018.
  - [13] Mikael Åkerholm and Rikard Land. Towards systematic software reuse in certifiable safety-critical systems. In *RESAFE-International Workshop on Software Reuse and Safety, Falls Church, VA*, 2009.
  - [14] Mohammed Al-Taie, Seifedine Nimer Kadry, and Obasa Isiaka Adekunle. Understanding expert finding systems: domains and techniques. *Social Network Analysis and Mining*, 8:1–9, 2018.
  - [15] D. Amalfitano, M. De Luca, A. Fasolino, P. Pelliccione, and T. Santilli. Characterizing software architectural metrics for continuous compliance in the automotive domain. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)*, pages 182–193, Los Alamitos, CA, USA, jun 2024. IEEE Computer Society.
  - [16] Domenico Amalfitano, Marco De Luca, and Anna Rita Fasolino. Documenting software architecture design in compliance with the iso 26262: a practical experience in industry. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, 2023.
  - [17] Domenico Amalfitano, Marco De Luca, and Anna Rita Fasolino. Documenting software architecture design in compliance with the iso 26262: a practical experience in industry. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages i–xi, 2023.
-



- 
- [18] Domenico Amalfitano, Marco Luca, Domenico Angelis, and Anna Fasolino. *Automated Architecture Recovery for Embedded Software Systems: An Industrial Case Study*, pages 53–68. 09 2024.
  - [19] Samuil Angelov, Paul Grefen, and Danny Greefhorst. A framework for analysis and design of software reference architectures. *Information and Software Technology*, 54(4):417–431, 2012.
  - [20] Noppadol Assavakamhaenghan, Morakot Choetkiertikul, Suppawong Tuarob, Raula Gaikovina Kula, Hideaki Hata, Chaoyong Ragkhitwetsagul, Thanwadee Sunetnanta, and Kenichi Matsumoto. Software team member configurations: A study of team effectiveness in moodle. In *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 19–195, 2019.
  - [21] Noppadol Assavakamhaenghan, Waralee Tanaphantaruk, Ponlakit Suwanworaboon, Morakot Choetkiertikul, and Suppawong Tuarob. Quantifying effectiveness of team recommendation for collaborative software development. *Automated Software Engineering*, 29, 08 2022.
  - [22] Marco Autili, Luca Berardinelli, Vittorio Cortellessa, Antinisca Marco, Davide Di Ruscio, Paola Inverardi, and Massimo Tivoli. A development process for self-adapting service oriented applications. volume 4749, pages 442–448, 09 2007.
  - [23] G. Avelino, L. Passos, A. Hora, and M. T. Valente. A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10, 2016.
  - [24] Nicholas Ayres, Lipika Deka, and Daniel Paluszczyszyn. Continuous automotive software updates through container image layers. *Electronics*, 10:739, 03 2021.
  - [25] Jung Ho Bae and Heung Seok Chae. Systematic approach for constructing an understandable state machine from a contract-based specification: controlled experiments. *Softw. Syst. Model.*, 15(3):847–879, jul 2016.
  - [26] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.
  - [27] Natércia A. Batista, Michele A. Brandão, Gabriela B. Alves, Ana Paula Couto da Silva, and Mirella M. Moro. Collaboration strength metrics and analyses on github. In *Proceedings of the International Conference on Web Intelligence, WI '17*, page 170–178, New York, NY, USA, 2017. Association for Computing Machinery.
-

- 
- [28] Stephan Baumgart, Joakim Fröberg, and Sasikumar Punnekkat. Industrial challenges to achieve functional safety compliance in product lines. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 356–360, 2014.
  - [29] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.
  - [30] Avijit Bhattacharjee, Sristy Sumana Nath, Shurui Zhou, Debasish Chakroborti, Banani Roy, Chanchal K. Roy, and Kevin Schneider. An exploratory study to find motives behind cross-platform forks from software heritage dataset. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR ’20, page 11–15, New York, NY, USA, 2020. Association for Computing Machinery.
  - [31] Ani Bicaku, Markus Tauber, and Jerker Delsing. Security standard compliance and continuous verification for industrial internet of things. *International Journal of Distributed Sensor Networks*, 16:155014772092273, 06 2020.
  - [32] David Fernández Blanco, Frédéric Le Mouël, Trista Lin, and Marie-Pierre Escudié. A comprehensive survey on software as a service (saas) transformation for the automotive systems. *IEEE Access*, 11:73688–73753, 2023.
  - [33] Barry W. Boehm and Victor R. Basili. Software defect reduction top 10 list. *Computer*, 34:135–137, 2001.
  - [34] P. Boldi, A. Pietri, S. Vigna, and S. Zacchiroli. Ultra-large-scale repository analysis via graph compression. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 184–194, 2020.
  - [35] Manfred Broy. Challenges in automotive software engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE ’06, page 33–42, New York, NY, USA, 2006. Association for Computing Machinery.
  - [36] Manfred Broy, Mario Gleirscher, Stefano Merenda, Doris Wild, Peter Kluge, and Wolfgang Krenzer. Toward a holistic and standardized automotive architecture description. *Computer*, 42(12):98–101, 2009.
  - [37] Alessio Bucaioni and Patrizio Pelliccione. Technical architectures for automotive systems. In *2020 IEEE International Conference on Software Architecture (ICSA)*, pages 46–57, 2020.
-

- 
- [38] Alessio Bucaioni, Patrizio Pelliccione, and Rebekka Wohlrab. Aligning architecture with business goals in the automotive domain. In *2021 IEEE 18th International Conference on Software Architecture (ICSA)*, pages 126–137, 2021.
  - [39] Georg Buchgeher, Claus Klammer, Bernhard Dorninger, and Albin Kern. Providing technical software documentation as a service - an industrial experience report. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 581–590, 2018.
  - [40] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.
  - [41] X. Cai, J. Zhu, B. Shen, and Y. Chen. Greta: Graph-based tag assignment for github repositories. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 63–72, 2016.
  - [42] Vadim Cebotari and Stefan Kugele. Playground for early automotive service architecture design and evaluation. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1349–1356, 2020.
  - [43] Tanmoy Chakraborty, Ayushi Dalmia, Animesh Mukherjee, and Niloy Ganguly. Metrics for community analysis: A survey. *ACM Comput. Surv.*, 50(4), aug 2017.
  - [44] Chih-Chieh Chang, Ming-Yi Chang, Jhao-Yin Jhang, Lo-Yao Yeh, and Chih-Ya Shen. Learning to extract expert teams in social networks. *IEEE Transactions on Computational Social Systems*, 9(5):1552–1562, 2022.
  - [45] Jiayi Chen, Zhixing Wang, Yuchen Jiang, Jun Pang, Tian Zhang, Minxue Pan, and Jianwen Sun. Negar: Network embedding guided architecture recovery for software systems. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, pages 367–376, 2022.
  - [46] Jie-Cherng Chen and Sun-Jen Huang. An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software*, 82(6):981–992, 2009.
  - [47] Jie-Cherng Chen and Sun-Jen Huang. An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software*, 82(6):981–992, 2009.
  - [48] Yufeng Chen, Jinwang Wu, and Zhongrui Wu. China’s commercial bank stock price prediction using a novel k-means-lstm hybrid approach. *Expert Systems with Applications*, 202:117370, 2022.
-

- 
- [49] Danny C. Cheng, Jod B. Villamarin, Gregory Cu, and Nathalie Rose Lim-Cheng. Towards compliance management automation thru ontology mapping of requirements to activities and controls. In *2018 Cyber Resilience Conference (CRC)*, pages 1–3, 2018.
  - [50] Danny C. Cheng, Jod B. Villamarin, Gregory Cu, and Nathalie Rose Lim-Cheng. Towards end-to-end continuous monitoring of compliance status across multiple requirements. *International Journal of Advanced Computer Science and Applications*, 9, 2018.
  - [51] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford. Documenting software architectures: views and beyond. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 740–741, 2003.
  - [52] Paul Clements and Len Bass. Relating business goals to architecturally significant requirements for software systems. Technical report, Carnegie-Mellon Univ Pittsburgh Software Engineering Institute, 2010.
  - [53] Riccardo Coppola and Maurizio Morisio. Connected car: Technologies, issues, future trends. *ACM Comput. Surv.*, 49(3), oct 2016.
  - [54] Théo Coulin, Maxence Detante, William Mouchère, and Fabio Petrillo. Software architecture metrics: a literature review. 01 2019.
  - [55] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, CSCW '12*, page 1277–1286, New York, NY, USA, 2012. Association for Computing Machinery.
  - [56] Yanja Dajsuren, Mark van den Brand, Alexander Serebrenik, and Rudolf Huisman. Automotive adls: a study on enforcing consistency through multiple architectural levels. *QoSA '12*, page 71–80, New York, NY, USA, 2012. Association for Computing Machinery.
  - [57] Joseph M. D'Alessandro, Cynthia D. Tanner, Bonnie W. Morris, and Tim Menzies. Is continuous compliance assurance possible? In *2009 Sixth International Conference on Information Technology: New Generations*, pages 1599–1599, 2009.
  - [58] Raghad Dardar, Barbara Gallina, Andreas Johnsen, Kristina Lundqvist, and Mattias Nyberg. Industrial experiences of building a safety case in compliance with iso 26262. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, pages 349–354, 2012.
-

- 
- [59] Marco De Luca, Anna Rita Fasolino, Antonino Ferraro, Vincenzo Moscato, Giancarlo Sperlì, and Porfirio Tramontana. A community detection approach based on network representation learning for repository mining. *Expert Systems with Applications*, 231:120597, 2023.
  - [60] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong Nguyen, and Riccardo Rubei. Topfilter: An approach to recommend relevant github topics. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, New York, NY, USA, 2020. Association for Computing Machinery.
  - [61] Stephane Ducasse and Damien Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591, 2009.
  - [62] Darko Durisic, Martin Nilsson, Mirosław Staron, and Jörgen Hansson. Measuring the impact of changes to the complexity and coupling properties of automotive software systems. *Journal of Systems and Software*, 86(5):1275–1293, 2013.
  - [63] Sebastian Dännart, Fabiola Moyón, and Kristian Beckers. *An Assessment Model for Continuous Security Compliance in Large Scale Agile Environments: Exploratory Paper*, pages 529–544. 05 2019.
  - [64] Ulf Eliasson, Rogardt Heldal, Patrizio Pelliccione, and Jonn Lantz. Architecting in the automotive domain: Descriptive vs prescriptive architecture. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*, pages 115–118, 2015.
  - [65] Gregor Engels, Alexander Förster, Reiko Heckel, and Sebastian Thöne. Process modeling using uml. pages 83 – 117, 10 2005.
  - [66] Christoph Etzel, Florian Hofhammer, and Bernhard Bauer. Towards metrics for analyzing system architectures modeled with east-adl. pages 441–448, 01 2020.
  - [67] Fabian Fagerholm, Marko Ikonen, Petri Kettunen, Jürgen Münch, Virpi Roto, and Pekka Abrahamsson. How do software developers experience team performance in lean and agile environments? In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery.
  - [68] Asma Fariha, Sanaa Alwidian, and Akramul Azim. A systematic literature review on requirements engineering and maintenance for embedded software. *IEEE Access*, 12:114263–114279, 2024.
-

- 
- [69] H. Fennel, Stefan Bunzel, Harald Heinecke, Jürgen Bielefeld, Simon Fürst, K. P. Schnelle, Walter Grote, Nico Maldener, Thomas Weber, Florian Andreas Wohlgemuth, Jens Ruh, Lennart Lundh, Tomas Sandén, Peter Heitkämper, Robert Rimkus, Jean Leflour, Alain Gilberg, Ulrich Virnich, Stefan Voget, Kenji Nishikawa, Kazuhiro Kajio, Klaus-Jörn Lange, Thomas Scharnhorst, and Bernd Kunkel. Achievements and exploitation of the autosar development partnership. 2006.
  - [70] Robert Filepp, Constantin Adam, Milton Hernandez, Maja Vukovic, Nikos Anerousis, and Guan Qun Zhang. Continuous compliance: Experiences, challenges, and opportunities. In *2018 IEEE World Congress on Services (SERVICES)*, pages 31–32, 2018.
  - [71] Erzheng Fu, Yingqiu Zhuang, Jianxi Zhang, Jiayun Zhang, and Yang Chen. Understanding the user interactions on github: A social network perspective. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 1148–1153. IEEE, 2021.
  - [72] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. pages 1797–1806, 11 2017.
  - [73] Barbara Gallina, Kathyayani Padira, and Mattias Nyberg. Towards an iso 26262-compliant oslc-based tool chain enabling continuous self-assessment. *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 199–204, 2016.
  - [74] David Garlan, Robert Allen, and John Ockerbloom. Architectural mismatch: Why reuse is still so hard. *IEEE Software*, 26(4):66–69, July 2009.
  - [75] Vahid Garuslu, Michael Felderer, Cagri Karapicak, and Ugur Yilmaz. Testing embedded software: A survey of the literature. *Information and Software Technology*, 104, 07 2018.
  - [76] Franz-Xaver Geiger, Ivano Malavolta, Luca Pascarella, Fabio Palomba, Dario Di Nucci, and Alberto Bacchelli. A graph-based dataset of commit history of real-world android apps. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR ’18, page 30–33, New York, NY, USA, 2018. Association for Computing Machinery.
  - [77] G. Gousios and D. Spinellis. Ghtorrent: Github’s data from a firehose. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 12–21, 2012.
  - [78] Georgios Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR ’13, page 233–236. IEEE Press, 2013.
-

- 
- [79] K. Grimm. Software technology in an automotive company - major challenges. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 498–503, May 2003.
  - [80] Viral Gupta, Parmod Kumar Kapur, and Deepak Kumar. Measuring architecture and design efficiency for enterprise applications. *International Journal of Industrial and Systems Engineering*, 28(4):494 – 529, 2018. Cited by: 4.
  - [81] Ayşe Günsel, Atif Açıkışz, Ayça Tükel, and Emine Ögüt. The role of flexibility on software development performance: An empirical study on software development teams. *Procedia - Social and Behavioral Sciences*, 58:853–860, 2012. 8th International Strategic Management Conference.
  - [82] Alireza Haghighatkhah, Ahmad Banijamali, Olli-Pekka Pakanen, Markku Oivo, and Pasi Kuvaja. Automotive software engineering: A systematic mapping study. *Journal of Systems and Software*, 128:25–55, 2017.
  - [83] N. Hajiakhoond Bidoki and G. Sukthankar. Network semantic segmentation with application to github. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1281–1284, 2018.
  - [84] Wen-Ming Han and Sun-Jen Huang. An empirical analysis of risk components and performance on software projects. *Journal of Systems and Software*, 80(1):42–50, 2007.
  - [85] Bernd Hardung, Thorsten Koelzow, and Andreas Krüger. Reuse of software in distributed embedded automotive systems. pages 203–210, 09 2004.
  - [86] Jacqueline Henle, Laurenz Adolph, Carl Philipp Hohl, and Eric Sax. A viewpoint-based evaluation method for future automotive architectures. In *2022 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–8, 2022.
  - [87] Steffen Herbold, Aynur Amirfallah, Fabian Trautsch, and Jens Grabowski. A systematic mapping study of developer social network research. *Journal of Systems and Software*, 171:110802, 2021.
  - [88] A. Hocking, John Knight, M. Aiello, and Shinichi Shiraishi. Arguing software compliance with iso 26262. In *Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014*, 11 2014.
  - [89] Tingting Hou, Xiangjuan Yao, and Dunwei Gong. Community detection in software ecosystem by comprehensively evaluating developer cooperation intensity. *Information and Software Technology*, 130:106451, 10 2020.
-

- 
- [90] Tingting Hou, Xiangjuan Yao, and Dunwei Gong. Community detection in software ecosystem by comprehensively evaluating developer cooperation intensity. *Information and Software Technology*, 130:106451, 2021.
  - [91] Yan Hu, Jun Zhang, Xiaomei Bai, Shuo Yu, and Zhuo Yang. Influence analysis of github repositories. *SpringerPlus*, 5(1):1268, 2016.
  - [92] ISO. ISO 26262 — Road vehicles — Functional safety, 2018.
  - [93] ISO. ISO 26262 — Road vehicles — Functional safety — Part 3: Concept phasel, 2018.
  - [94] ISO. ISO 26262 — Road vehicles — Functional safety — Part 6: Product development at the software level, 2018.
  - [95] Oskar Jarczyk, Blazej Gruszka, Leszek Bukowski, and Adam Wierzbicki. On the effectiveness of emergent task allocation of virtual programmer teams. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 01*, WI-IAT '14, page 369–376, USA, 2014. IEEE Computer Society.
  - [96] Houye Ji, Xiao Wang, Chuan Shi, Bai Wang, and Philip Yu. Heterogeneous graph propagation network. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.
  - [97] J. Jiang, L. Zhang, and L. Li. Understanding project dissemination on a social coding site. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 132–141, 2013.
  - [98] Katharina Juhnke, Matthias Tichy, and Frank Houdek. Challenges concerning test case specifications in automotive software testing. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 33–40, Aug 2018.
  - [99] Tim Kelly. Using software architecture techniques to support the modular certification of safety-critical systems. 69, 05 2007.
  - [100] Sebastian Kiebusch, Bogdan Franczyk, and Andreas Speck. Process-family-points. pages 314–321, 05 2006.
  - [101] Johannes Kloos, Tanvir Hussain, and Robert Eschbach. Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 26–33, 2011.
-



- 
- [102] Roland Knor, Georg Trausmuth, and Johannes Weidl. Reengineering c/c++ source code by transforming state machines. page 97–105. Springer-Verlag, 1998.
  - [103] Kostas Kontogiannis, Michael Athanasopoulos, and Chris Brealey. Continuous software engineering: Challenge areas and frameworks. In *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*, CASCON '16, page 335–338, USA, 2016. IBM Corp.
  - [104] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21, 2012.
  - [105] Richard A. Krueger and Mary Anne Casey. *Focus Groups A Practical Guide for Applied Research*. Sage, 2015.
  - [106] D. Kung, N. Suchak, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen. On object state testing. In *Proceedings Eighteenth Annual International Computer Software and Applications Conference (COMPSAC 94)*, pages 222–227, 1994.
  - [107] Rikard Land, Mikael Åkerholm, and Jan Carlson. Efficient software component reuse in safety-critical systems – an empirical study. In Frank Ortmeier and Peter Daniel, editors, *Computer Safety, Reliability, and Security*, pages 388–399, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
  - [108] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 467–476, New York, NY, USA, 2009. Association for Computing Machinery.
  - [109] William Leibzon. Social network of software development at github. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1374–1376, 2016.
  - [110] Yazmin Luna-Herrera, Juan Carlos Pérez-Arriaga, Jorge Ocharán-Hernández, and Angel Sanchez Garcia. *Comprehension of Computer Programs Through Reverse Engineering Approaches and Techniques: A Systematic Mapping Study*, pages 126–140. 10 2022.
  - [111] Yaping Luo and Mark van den Brand. Metrics design for safety assessment. *Information and Software Technology*, 73:151–163, 2016.
  - [112] Z. Luo, X. Mao, and A. Li. An exploratory research of github based on graph model. In *2015 Ninth International Conference on Frontier of Computer Science and Technology*, pages 96–103, 2015.
-

- 
- [113] Christian Manteuffel, Dan Tofan, Paris Avgeriou, Heiko Koziolk, and Thomas Goldschmidt. Decision architect – a decision documentation tool for industry. *Journal of Systems and Software*, 112:181–198, 2016.
  - [114] Onaiza Maqbool and Haroon Babri. Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*, 33(11):759–780, 2007.
  - [115] J.F. Maranzano, Sandra Rozsypal, G.H. Zimmerman, G.W. Warnken, P.E. Wirth, and David Weiss. Architecture reviews: Practice and experience. *Software, IEEE*, 22:34 – 43, 04 2005.
  - [116] Raluca Marinescu, Mehrdad Saadatmand, Alessio Bucaioni, Cristina Seceleanu, and Paul Pettersson. A model-based testing framework for automotive embedded systems. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 38–47, 2014.
  - [117] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. Impression formation in online peer production: Activity traces and personal profiles in github. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW '13*, page 117–128, New York, NY, USA, 2013. Association for Computing Machinery.
  - [118] Silverio Martínez-Fernández, Claudia P. Ayala, Xavier Franch, and Elisa Y. Nakagawa. A survey on the benefits and drawbacks of autosar. WASA '15, page 19–26, New York, NY, USA, 2015. Association for Computing Machinery.
  - [119] Nicholas Matragkas, James R. Williams, Dimitris S. Kolovos, and Richard F. Paige. Analysing the 'biodiversity' of open source ecosystems: The github case. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, page 356–359, New York, NY, USA, 2014. Association for Computing Machinery.
  - [120] Niklas Mellegard, Miroslaw Staron, and Fredrik Torner. A light-weight defect classification scheme for embedded automotive software and its initial evaluation. pages 261–270, 11 2012.
  - [121] Nilton Mendes Souza, Diógenes Dias, Lucas Bueno Ruas de Oliveira, Cristiane Aparecida Lana, Elisa Yumi Nakagawa, and José Carlos Maldonado. Exploring together software architecture and software testing: A systematic mapping. In *2016 35th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–12, Oct 2016.
-

- 
- [122] Hoang-Le Minh, Thanh Sang-To, Magd Abdel Wahab, and Thanh Cuong-Le. A new metaheuristic optimization based on k-means clustering algorithm and its application to structural damage identification. *Knowledge-Based Systems*, 251:109189, 2022.
  - [123] MIRA Ltd. MISRA-C:2004 Guidelines for the use of the C language in critical systems, October 2004.
  - [124] Audris Mockus, Diomidis Spinellis, Zoe Kotti, and Gabriel John Dusing. A complete set of related git repositories identified via community detection approaches based on shared commits. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 513–517, New York, NY, USA, 2020. Association for Computing Machinery.
  - [125] Behnaz Moradi-Jamei, Brandon L. Kramer, J. Bayoán Santiago Calderón, and Gizem Korkmaz. Community formation and detection on github collaboration networks. In *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '21, page 244–251, New York, NY, USA, 2022. Association for Computing Machinery.
  - [126] Bonnie Morris, Cynthia Tanner, and Joseph D'Alessandro. Enabling trust through continuous compliance assurance. In *2010 Seventh International Conference on Information Technology: New Generations*, pages 708–713, 2010.
  - [127] Vincenzo Moscato and Giancarlo Sperli. A survey about community detection over on-line social and heterogeneous information networks. *Knowledge-Based Systems*, 224:107112, 2021.
  - [128] Noha Moselhy, Yasser Ali, and Reem Mamdouh. Measurements of support processes: Proposed improvements on automotive spice pam v3.1 in light of oem standard supplier quality requirements. In Murat Yilmaz, Paul Clarke, Richard Messnarz, and Bruno Wöran, editors, *Systems, Software and Services Process Improvement*, pages 568–592, Cham, 2022. Springer International Publishing.
  - [129] Fabiola Moyon, Kristian Beckers, Sebastian Klepper, Philipp Lachberger, and Bernd Bruegge. Towards continuous security compliance in agile software development at scale. In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 31–34, 2018.
  - [130] Bhavesh Raju Mudhivarthi, Vaibhav Saini, Ayush Dodia, Pritesh Shah, and Ravi Sekhar. Model based design in automotive open system architecture.
-

- In *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 1211–1216, 2023.
- [131] Lukas Mäurer, Tanja Hebecker, Torben Stolte, Michael Lipaczewski, Uwe Möhrstädt, and Frank Ortmeier. On bringing object-oriented software metrics into the model-based world – verifying iso 26262 compliance in simulink. 09 2014.
  - [132] N. Walkinshaw, K. Bogdanov, S. Ali, and M. Holcombe. Automated discovery of state transitions and their functions in source code. page 99 – 121, 2008.
  - [133] P. Narang and P. Mittal. Implementation of devops based hybrid model for project management and deployment using jenkins automation tool with plugins. *International Journal of Computer Science and Network Security*, 22(8):249–259, Aug 2022.
  - [134] P. Narang and P. Mittal. Performance assessment of traditional software development methodologies and devops automation culture. *Engineering, Technology & Applied Science Research*, 12(6):9726–9731, Dec. 2022.
  - [135] Bashar Nassar and Riccardo Scandariato. Traceability metrics as early predictors of software defects? In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 235–238, 2017.
  - [136] Nicolas Navet and Françoise Simonot-Lion. *Automotive embedded systems handbook*. CRC press, 2017.
  - [137] Michael L. Nelson. A survey of reverse engineering and program comprehension. *ArXiv*, abs/cs/0503068, 2005.
  - [138] M. Newman. Equivalence between modularity optimization and maximum likelihood methods for community detection. *Physical Review E*, 94, 11 2016.
  - [139] P.T. Nguyen, J. Di Rocco, R. Rubei, and D. Di Ruscio. An automated approach to assess the similarity of github repositories. *Software Quality Journal*, 28(2):595–631, 2020. cited By 2.
  - [140] Liam O’Brien, Paulo Merson, and Len Bass. Quality attributes for service-oriented architectures. In *International Workshop on Systems Development in SOA Environments (SDSOA’07: ICSE Workshops 2007)*, pages 3–3, 2007.
  - [141] Minxue Pan, Shouyu Chen, Yu Pei, Tian Zhang, and Xuandong Li. Easy modelling and verification of unpredictable and preemptive interrupt-driven systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 212–222, 2019.
-

- 
- [142] Juraj Pancik, Aleš Vémola, Robert Kledus, Marek Semela, and Albert Bradáč. Auto recalls and software quality in the automotive sector. *ICST Transactions on Scalable Information Systems*, 5:154808, 05 2018.
- [143] Stefan Penthin. Software-Over-The-Air (SOTA): An Automotive Accelerator. <https://www.bearingpoint.com/en-se/our-success/thought-leadership/software-over-the-air-sota-an-automotive-accelerator/>.
- [144] Duc Truong Pham, Stefan S Dimov, and Chi D Nguyen. Selection of k in k-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 219(1):103–119, 2005.
- [145] Simon Phipps and Stefano Zacchiroli. Continuous open source license compliance. *Computer*, 53(12):115–119, 2020.
- [146] Antoine Pietri, Guillaume Rousseau, and Stefano Zacchiroli. Determining the intrinsic structure of public software development history. In *Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20*, page 602–605, New York, NY, USA, 2020. Association for Computing Machinery.
- [147] G. Pinto, I. Steinmacher, and M. A. Gerosa. More common than you think: An in-depth study of casual contributors. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 112–123, 2016.
- [148] M. Pinzger, M. Fischer, H. Gall, and M. Jazayeri. Revealer: a lexical pattern matcher for architecture recovery. In *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, pages 170–178, 2002.
- [149] Shobha S Prabhu, Hem Kapil, and Shashirekha H Lakshmaiah. Safety critical embedded software: Significance and approach to reliability. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 449–455, 2018.
- [150] Md. Mokhlesur Rahman and Jean-Claude Thill. Impacts of connected and autonomous vehicles on urban transportation and environment: A comprehensive review. *Sustainable Cities and Society*, 96:104649, 2023.
- [151] Mohammad Masudur Rahman and Chanchal K. Roy. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, page 364–367, New York, NY, USA, 2014. Association for Computing Machinery.
-

- 
- [152] Fabíola Gonçalves C. Ribeiro, Achim Reuberg, Carlos E. Pereira, and Michel S. Soares. An approach for architectural design of automotive systems using marte and sysml. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1574–1580, 2018.
  - [153] Wasim Said, Jochen Quante, and Rainer Koschke. Mining understandable state machine models from embedded code. *Empirical Software Engineering*, 25(6):4759 – 4804, 2020.
  - [154] Ionut-Andrei Sandu and Alexandru Salceanu. Improved technique for measuring the number of defects in automotive agile sw development : Defect debt trend. In *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*, pages 0765–0768, 2018.
  - [155] Teodora Sanislav and Liviu Miclea. Cyber-physical systems - concept, challenges and research areas. *Control Engineering and Applied Informatics*, 14:28–33, 01 2012.
  - [156] T. Santilli, P. Pelliccione, R. Wohlrab, and A. Shahrokni. What is continuous compliance? *IEEE Software*, (01):1–10, dec 5555.
  - [157] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, 2017.
  - [158] V. Schettino, V. Horta, M. A. P. Araújo, and V. Ströele. Towards community and expert detection in open source global development. In *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 350–355, 2019.
  - [159] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. pages 593–607, 2018.
  - [160] Philipp Seifer, Johannes Härtel, Martin Leinberger, Ralf Lämmel, and Steffen Staab. Empirical study on the usage of graph query languages in open source java projects. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2019, page 152–166, New York, NY, USA, 2019. Association for Computing Machinery.
  - [161] Kalyani Selvarajah, Ziad Kobti, and Mehdi Kargar. Cultural algorithms for cluster hires in social networks. *Procedia Computer Science*, 170:514–521, 2020. The 11th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 3rd International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.
-

- 
- [162] Mall R. Sen T. Extracting finite state representation of java programs. In *Softw Syst Model*, page 497–511, 2016.
  - [163] Ali Shahrokni and Patrizio Pelliccione. Significance of continuous compliance in automotive. In *The International Conference on Evaluation and Assessment in Software Engineering 2022*, EASE 2022, page 272–273, New York, NY, USA, 2022. Association for Computing Machinery.
  - [164] Ali Shahrokni and Patrizio Pelliccione. Significance of continuous compliance in automotive. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, EASE '22, page 272–273, New York, NY, USA, 2022. Association for Computing Machinery.
  - [165] Rebecca Shannon-Spicer, Amin Vahabaghaie, George Bahouth, Ludwig Drees, Robert Bülow, and Peter Baur. Field effectiveness evaluation of advanced driver assistance systems. *Traffic Injury Prevention*, 19:1–5, 12 2018.
  - [166] Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. Understanding "watchers" on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 336–339, New York, NY, USA, 2014. Association for Computing Machinery.
  - [167] Adam Sherer, John Rose, and Riccardo Oddone. Ensuring functional safety compliance for iso 26262. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–3, 2015.
  - [168] Ikhlaz Sidhu, Sudarshan Gopalakrishnan, and Rajarathnam Balakrishnan. Effectiveness factors for algorithm based team formation with data project case application. In *2021 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–6, 2021.
  - [169] Lakshitha Silva and Dharini Balasubramaniam. Controlling software architecture erosion: A survey. *Journal of Systems and Software*, 85:132–151, 01 2012.
  - [170] Samira Silva, Adiel Tuyishime, Tiziano Santilli, Patrizio Pelliccione, and Ludovico Iovino. Quality metrics in software architecture. In *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, pages 58–69, 2023.
  - [171] Jacopo Sini, Massimo Violante, and Fabrizio Tronci. A novel iso 26262-compliant test bench to assess the diagnostic coverage of software hardening techniques against digital components random hardware failures. *Electronics*, 11(6), 2022.
-

- 
- [172] Stéphane S. Somé and Timothy C. Lethbridge. Enhancing program comprehension with recovered state models. In *Proceedings of the 10th International Workshop on Program Comprehension, IWPC '02*. IEEE Computer Society, 2002.
  - [173] Ioana Şora. Helping program comprehension of large software systems by identifying their most important classes. In Leszek A. Maciaszek and Joaquim Filipe, editors, *Evaluation of Novel Approaches to Software Engineering*, pages 122–140, Cham, 2016. Springer International Publishing.
  - [174] Mirosław Staron. *Automotive software architectures: An introduction*. 2017. Cited by: 24; All Open Access, Green Open Access.
  - [175] Mirosław Staron. *Introduction*, pages 1–18. Springer International Publishing, Cham, 2021.
  - [176] Mirosław Staron and Darko Durisic. *AUTOSAR Standard*, pages 81–116. Springer International Publishing, Cham, 2017.
  - [177] Kim Strandberg, Ulf Arnljung, Tomas Olovsson, and Dennis Kengo Oka. Secure vehicle software updates: Requirements for a reference architecture. In *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, pages 1–7, June 2023.
  - [178] C. Stringfellow, C.D. Amory, D. Potnuri, A. Andrews, and M. Georg. Comparison of software architecture reverse engineering methods. *Information and Software Technology*, 48(7):484–497, 2006.
  - [179] Emre Sülün, Eray Tüzün, and Uğur Doğrusöz. Reviewer recommendation using software artifact traceability graphs. In *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE'19, page 66–75, New York, NY, USA, 2019. Association for Computing Machinery.
  - [180] Mikael Svahnberg, Claes Wohlin, Lars Lundberg, and Michael Mattsson. A method for understanding quality attributes in software architecture structures. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE '02*, page 819–826, New York, NY, USA, 2002. Association for Computing Machinery.
  - [181] Tarja Systä. Static and dynamic reverse engineering techniques for java software systems. 2000.
  - [182] Sparx Systems. Using UML Part Two – Behavioral Modeling Diagrams, 2018.
-



- 
- [183] Emre Sülün, Eray Tüzün, and Uğur Doğrusöz. Rstrace+: Reviewer suggestion using software artifact traceability graphs. *Information and Software Technology*, 130:106455, 2021.
  - [184] Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2010.
  - [185] A. Telea, A. Maccari, and C. Riva. An open visualization toolkit for reverse architecting. In *Proceedings 10th International Workshop on Program Comprehension*, 2002.
  - [186] Thiviyan Thanapalasingam, Lucas Berkel, Peter Bloem, and Paul Groth. Relational graph convolutional networks: A closer look. 07 2021.
  - [187] Sam L. Thomas, Jan Van den Herrewegen, Georgios Vasilakis, Zitai Chen, Mihai Ordean, and Flavio D. Garcia. Cutting through the complexity of reverse engineering embedded devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, (3), 2021.
  - [188] Paolo Tonella. Reverse engineering of object oriented code. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, page 724–725, New York, NY, USA, 2005. Association for Computing Machinery.
  - [189] Théo Trouillon, Christopher R. Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *J. Mach. Learn. Res.*, 18:130:1–130:38, 2017.
  - [190] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. 2016.
  - [191] Sara Tucci-Piergiovanni, Chokri Mraidha, Ernest Wozniak, Agnes Lanusse, and Sebastien Gerard. A uml model-based approach for replication assessment of autosar safety-critical applications. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1176–1187, 2011.
  - [192] Gias Uddin and Martin P. Robillard. How api documentation fails. *IEEE Software*, 32(4):68–75, 2015.
  - [193] Burak Uzun and Bedir Tekinerdogan. Architecture conformance analysis using model-based testing: A case study approach. *Software - Practice and Experience*, 49(3), 2019.
  - [194] van Zeeland D. van den Brand M, Serebrenik A. Extraction of state machines of legacy c code with cpp2xmi. In *Proceedings of 7th belgian-netherlands software evolution workshop*, pages 28–30, 2008.
-

- 
- [195] Hariharan Venkitachalam, Johannes Richenhagen, Axel Schlosser, and Thomas Tasky. Metrics for verification and validation of architecture in powertrain software development. In *2015 First International Workshop on Automotive Software Architecture (WASA)*, pages 27–33, 2015.
  - [196] Roberto Verdecchia, Philippe Kruchten, and Patricia Lago. Architectural technical debt: A grounded theory. In *Software Architecture: 14th European Conference, ECSA 2020, L'Aquila, Italy, September 14–18, 2020, Proceedings*, Berlin, Heidelberg, 2020. Springer-Verlag.
  - [197] M. Visconti and C.R. Cook. An overview of industrial software documentation practice. volume 2002-January, page 179 – 186, 2002. Cited by: 12; All Open Access, Green Open Access.
  - [198] Martin Vogel, Peter Knapik, Moritz Cohrs, Bernd Szyperek, Winfried Pueschel, Haiko Etzel, Daniel Fiebig, Andreas Rausch, and Marco Kuhrmann. Metrics in automotive software development: A systematic literature review. *Journal of Software: Evolution and Process*, 33(2):e2296, 2021. e2296 smr.2296.
  - [199] Marco Wagner, Dieter Zobel, and Ansgar Meroth. Towards runtime adaptation in autosar: Adding service-orientation to automotive software architecture. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–7, 2014.
  - [200] Jiang Wan, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Cyber-physical codesign at the functional level for multidomain automotive systems. *IEEE Systems Journal*, 11(4):2949–2959, 2017.
  - [201] D. Wang, J. Cao, S. Qian, and Q. Qi. Investigating cross-repository socially connected teams on github. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pages 490–497, 2019.
  - [202] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S Yu Philip. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Transactions on Big Data*, 2022.
  - [203] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and Philip S. Yu. A survey on heterogeneous graph embedding: Methods, techniques, applications and sources. *IEEE Transactions on Big Data*, pages 1–1, 2022.
  - [204] Z. Wang and D. E. Perry. Role distribution and transformation in open source software project teams. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 119–126, 2015.
-

- 
- [205] Chauncey Wilson. Chapter 2 - semi-structured interviews. In Chauncey Wilson, editor, *Interview Techniques for UX Practitioners*, pages 23–41. Morgan Kaufmann, Boston, 2014.
  - [206] Rebekka Wohlrab, Ulf Eliasson, Patrizio Pelliccione, and Rogardt Høldal. Improving the consistency and usefulness of architecture descriptions: Guidelines for architects. In *2019 IEEE International Conference on Software Architecture (ICSA)*, pages 151–160, 2019.
  - [207] Lu Xiao, Yuanfang Cai, Rick Kazman, Ran Mo, and Qiong Feng. Identifying and quantifying architectural debt. ICSE '16, page 488–498, New York, NY, USA, 2016. Association for Computing Machinery.
  - [208] Yu Xie, Bin Yu, Shengze Lv, Chen Zhang, Guodong Wang, and Maoguo Gong. A survey on heterogeneous network representation learning. *Pattern Recognition*, 116:107936, 2021.
  - [209] Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E. Hassan, and Naoyasu Ubayashi. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of the 14th International Workshop on Principles of Software Evolution, IWPSE 2015*, page 46–55, New York, NY, USA, 2015. Association for Computing Machinery.
  - [210] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.
  - [211] Luting Ye, Hailong Sun, Xu Wang, and Jiaruijie Wang. Personalized teammate recommendation for crowdsourced software developers. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE '18*, page 808–813, New York, NY, USA, 2018. Association for Computing Machinery.
  - [212] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun. Detecting similar repositories on github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 13–23, 2017.
  - [213] Morteza Zihayat, Mehdi Kargar, and Aijun An. Two-phase pareto set discovery for team formation in social networks. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 02, WI-IAT '14*, page 304–311, USA, 2014. IEEE Computer Society.
-



# Author's publications

1. P. Tramontana, M. De Luca, A. R. Fasolino. *An Approach for Model Based Testing of Augmented Reality Applications*, RCIS Workshops, 2022
2. M. De Luca, A. R. Fasolino, A. Ferraro, V. Moscato, G. Sperlí, P. Tramontana. A community detection approach based on network representation learning for repository mining. *Expert Systems with Applications*, Volume 231, 2023, DOI: 10.1016/j.eswa.2023.120597.
3. D. Amalfitano, M. De Luca, A. Rita Fasolino. *Documenting Software Architecture Design in Compliance with the ISO 26262: a Practical Experience in Industry*, IEEE 20th International Conference on Software Architecture Companion (ICSA-C), 2023, DOI: 10.1109/ICSA-C57050.2023.00022.
4. M. De Luca, A. R. Fasolino, P. Tramontana. *Investigating the robustness of locators in template-based Web application testing using a GUI change classification model*, *Journal of Systems and Software*, Volume 210, 2024, DOI: 10.1016/j.jss.2023.111932.
5. D. Amalfitano, M. De Luca, D. F. De Angelis, A. Rita Fasolino. *Automated Architecture Recovery for Embedded Software Systems: An Industrial Case Study*, 18th European Conference on Software Architecture (ECSA), 2024, DOI: 10.1007/978-3-031-70797-1\_4
6. M. De Luca, S. Di Meglio, A. R. Fasolino, L. L. L. Starace, P. Tramontana. *Automatic Assessment of Architectural Anti-patterns and Code Smells in Student Software Projects*, 28th International Conference on Evaluation and Assessment in Software Engineering (EASE), 2024, DOI: 10.1145/3661167.3661290
7. D. Amalfitano, M. De Luca, A. Fasolino, P. Pelliccione and T. Santilli. *Characterizing Software Architectural Metrics for Continuous Compliance*

*in the Automotive Domain*, IEEE 21st International Conference on Software Architecture (ICSA), 2024, DOI: 10.1109/ICSA59870.2024.00025