

***Ein Entwurfsmuster-basierter Ansatz
zur Systematisierung der
Wiederverwendung
sicherheitskritischer Systemlösungen
in automobilen Applikationen***

***A Design Pattern-based Approach for
the Guided Model-based Reuse of
Safety-critical System Solutions in the
Automotive Domain***

Maged Khalil

Geleitwort

Advanced Systems Engineering beschreibt die Entwicklung und das Betreiben intelligenter technischer Systeme als ganzheitliche Ingenieurdisziplin durch neue Perspektiven. Das ist die verbindende Leitidee unserer Forschungsarbeit. Die Forschung der gleichnamigen Fachgruppe am Heinz Nixdorf Institut der Universität Paderborn sowie am Fraunhofer-Institut für Entwurfstechnik Mechatronik IEM verfolgt das Ziel, die Innovationskraft von Industrieunternehmen zu steigern und sie durch innovative Methoden und Werkzeuge dabei zu unterstützen, die steigende Komplexität moderner Systeme zu beherrschen und Systems Engineering erfolgreich in die industrielle Praxis zu überführen.

Gerade in der Entwicklung sicherheitskritischer, softwareintensiver Systeme ist die Wiederverwendung bewährter Lösungen von enormer Bedeutung. In der industriellen Praxis erfolgt sie jedoch häufig opportunistisch und mit einem engen Fokus auf Implementierungsartefakte, wodurch Kontextinformationen, Annahmen und Begründungen verloren gehen. Es fehlen strukturierte Vorgehensweisen, geeignete Werkzeugunterstützung sowie Kriterien zur Bewertung der Vollständigkeit und Qualität von Wiederverwendungsaktivitäten. Dies erschwert eine konsistente, nachvollziehbare und effiziente Wiederverwendung.

Vor diesem Hintergrund entwickelt Herr Khalil mit seiner Dissertation **ein Entwurfsmuster-basierter Ansatz zur Systematisierung der Wiederverwendung sicherheitskritischer Systemlösungen in automobilen Applikationen**. Durch die Operationalisierung und modellbasierte Verankerung von Entwurfsmustern werden wiederverwendungsrelevante Informationen unmittelbar in bestehende Engineering-Artefakte und Entwicklungsumgebungen integriert. Der entwickelte Ansatz verbindet strukturierte Wissensrepräsentation, modellbasierte Methoden und praktische Werkzeugunterstützung zu einem kohärenten und konsistenten Lösungsrahmen, dessen Anwendbarkeit sowohl in wissenschaftlichen Fallstudien als auch in einer mehrjährigen industriellen Langzeitstudie überzeugend nachgewiesen wird.

Herr Khalil leistet mit seiner Arbeit einen klaren und eigenständigen Beitrag zur Weiterentwicklung der modellbasierten Wiederverwendung im Systems- und Safety Engineering. Die Dissertation zeichnet sich durch ihre methodische Konsistenz, ihre hohe industrielle Relevanz sowie ihre starke Verankerung in der industriellen Praxis aus. Besonders hervorzuheben ist die systematische Ausarbeitung eines anwendbaren Lösungsrahmens, der wissenschaftlich fundierte Methoden mit den Anforderungen realer Entwicklungsprozesse verbindet. Die Arbeit liefert damit wertvolle Impulse für die weitere Forschung und zeigt zugleich auf, wie wissenschaftlich fundierte Ansätze erfolgreich in industrielle Entwicklungsprozesse überführt werden können und hierdurch die praktische Etablierung eines systematischen, modellbasierten Systems Engineering in Unternehmen ermöglicht wird.

**Ein Entwurfsmuster-basierter Ansatz zur
Systematisierung der Wiederverwendung
sicherheitskritischer Systemlösungen in
automobilen Applikationen**

**A Design Pattern-based Approach for the Guided
Model-based Reuse of Safety-critical System
Solutions in the Automotive Domain**

zur Erlangung des akademischen Grades eines
DOKTORS DER NATURWISSENSCHAFTEN (Dr. rer. nat.)
der Fakultät Elektrotechnik, Informatik und Mathematik
der Universität Paderborn

genehmigte
DISSERTATION

von
M.Eng. Maged Khalil

Tag des Kolloquiums:
Referent:
Ko-referent:

24. April 2026
Prof. Dr.-Ing. Roman Dumitrescu
Prof. Dr. Andreas Vogelsang

Foreword

This dissertation is the result of more than 15 years of continuous work on this topic, beginning with the identification of the underlying problem during my time as a young R&D engineer in industry. It subsequently led to a focused research position at the fortiss research institute and the Technical University of Munich. This research path continued back in industry over the better part of the following decade, culminating in collaborative engagements with the Systems Engineering and Design Research groups at the Heinz Nixdorf Institute and the Fraunhofer Institute for Mechatronic Systems Design IEM in Paderborn, who helped to revive and give final shape to this thesis endeavor.

I am very grateful to Professor Roman Dumitrescu for supervising the completion of this dissertation, for guiding me through the necessary steps to bring this long-standing endeavor to its conclusion, and for providing the opportunity to formalize this contribution within an academic framework. I am especially indebted to you, Roman, for doing it all in the trusting, friendly, and very accessible way that is uniquely yours. Many thanks to Professor Andreas Vogelsang for supporting as co-examiner as well as to the entire examination committee.

The genesis of any project is often the hardest step, and for that, I would like to express my gratitude to Professor Manfred Broy, who initially took me on as a doctoral researcher and set me on this research path. I also sincerely thank Professor Alexander Pretschner for his support during a critical transitional period, and for encouraging me to consider the value of this work within the broader context of my professional journey.

The journey to this dissertation was non-linear and required perseverance through multiple stages of independent research and industrial application. I am grateful to all those who, at different points, provided the opportunity, encouragement, and formal support necessary to see it through to completion. I would especially like to express my deep gratitude to two individuals whose influence on my journey has been profound, though they are sadly no longer with us.

Bernhard Schätz offered encouragement, guidance, and a model of both intellectual rigor and kindness at a time when I most needed it during my early academic career. His ability to bridge deep formal expertise with a practical mindset, his generosity of spirit, and his collaborative support left a lasting mark on me, both personally and professionally, and his mentorship, offered even while facing his own personal battle with terminal illness, continues to inspire me.

I am equally grateful to Friedrich Scheerer, whose broad cross-disciplinary knowledge, engineering rigor, and fearless honesty were a constant source of encouragement and inspiration. His example confirmed for me that technical excellence, business success, and personal integrity can and must go hand in hand for any true leader.

I remain sincerely thankful for their guidance, friendship, and enduring impact.

Finally, I would like to thank my family and friends for their unwavering patience, support, and encouragement throughout this long journey.

To my wife, who had to listen to me talk, give up, rant, and circle back to this endeavor more times than either of us can count – thank you for your patience, your belief in me even when I

doubted myself, and for the stability and love that ultimately made it possible for me to find my way back to this work. The arrival of our beautiful daughter gave me the final push to move beyond hesitation and finish what I started, and for that, I am forever grateful.

I would also like to thank my brother and my parents, who endured this long road with me. My father – Professor and Dean Emeritus – may have been even more hurt than I was when I decided to abandon my PhD in 2019. Though he could not fully understand the emotional toll the journey had taken at the time, his and my mother’s lifelong commitment to scholarship and excellence remained a silent encouragement to pick up the work again when the time was right.

Without your support and belief in me, especially during the most difficult years, this dissertation would not have been completed.

Thank you.

Frankfurt am Main, Dec. 4th, 2025

M.Eng. Maged Khalil

Acknowledgments

The initial specification of this approach stems from and was carried out in the integrated development environment for the ITEA2 SAFE project and the co-led EUROSTARS SAFE-E project, with proof-of-concept implementations in the research CASE tool AUTOFOCUS3 (AF3). AF3 is the result of the hard work of many researchers and employees of the Software and Systems Engineering Department at the fortiss Institute (and previously the Software and Systems Engineering Chair at the Faculty of Informatics at the Technical University of Munich) in Munich, Germany.

COREPA and its successor projects represent the collective work of many dedicated colleagues at my former employer (at the time of this publication), a globally leading automotive Tier-1 supplier. I am sincerely thankful for their contributions and collaboration. I am particularly grateful for the fruitful collaboration with Alexandru Ipatiov, who, over many years, led the implementation of many of the changes and enhancements I recommended through applying my approach in practice. Without his support, the longitudinal industrial case study, which distinguishes this thesis from other more academically oriented evaluations, would have been a much more difficult undertaking. My sincere thanks also go to all survey participants whose honest feedback made the evaluation of the approach possible.

Zusammenfassung

Die Wiederverwendung betriebsbewährter Lösungen – etwa von Sicherheitsmechanismen oder Architekturdesigns – ist in der Entwicklung sicherheitskritischer Systeme weit verbreitet, um das Vertrauen in das Design zu erhöhen, den Entwicklungsaufwand zu reduzieren und die Einhaltung von Anforderungen zu fördern. Dennoch erfolgt diese Wiederverwendung nach wie vor weitgehend ad hoc und opportunistisch, da es an Prozessreife, strukturierten Anleitungen und geeignete Werkzeugunterstützung mangelt. Hinzu kommt das Fehlen expliziter Kriterien zur Bewertung von Vollständigkeit, Qualität und Reife der Wiederverwendung, was eine Bestimmung der tatsächlichen Wirkung und Effektivität entsprechender Aktivitäten erschwert.

Diese Dissertation entwickelt ein modellbasiertes Vorgehen zur systematischen Unterstützung der Wiederverwendung sicherheitskritischer, softwareintensiver Systemlösungen in der Praxis. Im Zentrum stehen die strukturierte Erhebung, Erfassung und Modellierung aller für eine korrekte Wiederverwendung erforderlichen Informationen – integriert in jene in der industriellen Praxis genutzten Werkzeuge und Artefakte. Daraus entsteht ein industrietaugliches Rahmenwerk, das eine konsistente und kontextbewusste Wiederverwendung im Entwicklungsalltag ermöglicht.

Auf Basis der nutzerzentrierten Prinzipien von Design Patterns (Entwurfsmuster, im Folgenden kurz „Patterns“) wird eine modulare, modellbasierte Pattern-Bibliothek definiert, die Entwicklungsartefakte und strukturierte Argumentationselemente verbindet und zusammen mit geführten Anweisungen in einem holistischen Ansatz integriert. Am Beispiel von Sicherheitsmechanismen im Automotive-Bereich wird hierfür eine wiederverwendbare Pattern-Vorlage entwickelt und in zwei Umgebungen instanziiert: einer domänenunabhängigen Forschungsplattform sowie einem automobilen Modellierungsrahmenwerk.

Die Methodik wird anschließend verallgemeinert und in einer mehrjährigen industriellen Langzeitfallstudie bei einem weltweit führenden Tier-1-Automobilzulieferer auf ein breit angelegtes Wiederverwendungsproblem angewandt. Eine systematische Expertenbefragung sowie werkzeuggestützte Reifegradbewertungen validieren die Wirksamkeit und Praxistauglichkeit des Ansatzes.

Die modellbasierte Erfassung von Patterns innerhalb ihres praktischen Kontexts bietet mehrere wesentliche Vorteile: sie verbessert die Charakterisierung des Problemraums, ermöglicht eine präzisere und praxisnähere Beschreibung der Wiederverwendungsschritte und stärkt die Abstimmung relevanter interdisziplinärer Artefakte – wodurch die Ko-Evolution von Systemdesignartefakten und Artefakten anderer Disziplinen (z. B. Sicherheitsnachweisen) gefördert wird. Zudem bietet der Ansatz einen Mechanismus zur systematischen Bewertung der Wiederverwendungseignung von Werkzeugketten als auch zur gezielten Identifikation und Schließung methodischer sowie werkzeugspezifischer Lücken.

Die Beiträge dieser Arbeit umfassen eine neuartige Operationalisierung von Design Patterns, ein domänenübergreifend anwendbares Wiederverwendungsrahmenwerk sowie validierte Leitlinien für Organisationen, die Wiederverwendung im industriellen Maßstab systematisch etablieren möchten, und sind somit sowohl für die Wissenschaft als auch für die Praxis von Relevanz.

Abstract

The reuse of proven technical solutions – such as Safety Mechanisms or architecture designs – is widely practiced in safety-critical systems engineering to improve design confidence, reduce development effort, and ensure compliance. However, such reuse remains largely ad hoc and opportunistic; lacking process maturity, structured guidance, or adequate tool support. In particular, the absence of explicit criteria for completeness, quality and maturity hinders the evaluation of reuse efforts.

This thesis introduces a structured, model-based approach for enabling systematic and context-aware reuse of system solutions. Central to the approach – and contrary to related work – is the focus on systematically capturing all reuse-relevant information in the exact artifacts and tools they will be reused with in practice. Using Safety Mechanisms in the automotive domain as a representative class of solutions, a reusable design pattern template is developed and validated. Building on the user-centric principles of Design Patterns, the approach defines a modular Model-Based Pattern Library, which integrates both development artifacts and structured argumentation elements. The approach supports both reuse *designers* and *users* through tailored workflows, and is instantiated in two development environments: a domain-independent research platform and a domain-specific automotive modeling framework.

The methodology is further generalized and applied to a large-scale automotive system solution reuse problem in a longitudinal industrial case study at a globally leading automotive Tier 1 supplier across multi-year iterative cycles, with proven positive impact validated through tool-supported maturity evaluations and a systematic expert survey.

The model-based representation of patterns within their practical context yields several key advantages: improved characterization of the problem space; a more natural and direct expression of reuse transformations; and a stronger alignment between the cross-disciplinary artifacts necessary for correct reuse – facilitating the co-evolution of system design and other discipline work products, such as safety assurance artifacts. Importantly, the approach provides a mechanism for assessing and improving the reuse readiness of toolchains, supporting prioritized targeted gap closure, capability extension and adoption decisions.

The contributions of this work are both theoretical and practical: a novel operationalization of design patterns, a reusable framework applicable across domains, and validated guidance for organizations seeking to scale solution reuse systematically and effectively. The resulting insights not only validate the proposed methodology but also offer actionable guidance for organizations striving to scale reuse effectively in complex, real-world engineering environments.

A Design Pattern-based Approach for the Guided Model-based Reuse of Safety-critical Solutions in the Automotive Domain

Table of Contents

Prologue.....	IX
1 Introduction.....	1
1.1 Motivation, Example and Scope	2
1.2 Problem Statement and Challenges	5
1.3 Solution Proposal	6
1.4 Targeted Advantages and Practical Benefits	7
1.5 Research Methodology.....	8
1.6 Structure of Thesis.....	10
1.7 Previous publications by the author	12
2 Systems and Safety.....	16
2.1 Research Methodology	16
2.2 Systems Engineering, Embedded Systems & Dependability	17
2.3 Safety-critical Systems	19
2.4 Safety Cases – Claims, Arguments and Evidence	31
2.5 Model-based Engineering.....	37
2.6 Summary	46
3 Design Patterns and Safety Mechanisms	47
3.1 Research Methodology	48
3.2 A Brief History of Design Patterns	49
3.3 Safety Mechanisms	51

3.4	Design Pattern Templates	53
3.5	Summary	60
4	MODPAL – The Model-based Design Pattern Library Approach	61
4.1	Research methodology	61
4.2	Our Solution in Context.....	62
4.3	Overview of Approach	64
4.4	Related Work.....	66
4.5	A Design Pattern Template for Safety Mechanisms	76
4.6	A Pattern Library for Safety Mechanisms	82
4.7	Artifact Reference Structure for Safety Mechanism Patterns.....	87
4.8	Instantiation Guideline and Artifact Mapping	99
4.9	Evaluating the Development Environments	103
4.10	Targeted Advantages and Benefits of our Approach	106
4.11	Summary	108
5	Instantiation in AutoFOCUS3 – a Research CASE Tool	109
5.1	Introduction to AUTOFOCUS3	110
5.2	Capturing Safety Mechanism Patterns in AF3	111
5.3	Implementation of our approach in AF3 and resulting Guidelines	116
5.4	Evaluation of Safety Mechanism Pattern Coverage in AF3	129
5.5	Summary	132
6	Instantiation in SAFE – an Automotive Safety Modeling Framework	134
6.1	The SAFE Project.....	134
6.2	Safety Case Modelling in SAFE.....	140
6.3	Safety Mechanisms and “Tactics” in the SAFE Meta-model.....	145
6.4	Evaluation of Safety Mechanism Pattern Coverage in the SAFE Platform	148
6.5	Summary	152

7	Instantiation in COREPA – a Generalized Industrial Case Study ..	154
7.1	Research methodology	155
7.2	Instantiation of the Pattern Library approach in Industry.....	156
7.3	Expert Survey – Exploring the approach’s multi-year impact.....	165
7.4	Summary	198
8	Conclusion, Limitations and Outlook	199
8.1	Summary of the Research	199
8.2	Contributions of the Thesis	201
8.3	Key Findings and Impact	203
8.4	Threats to Validity and Limitations	205
8.5	Outlook and Open Questions	206
8.6	Cauda	208
	Bibliography and List of references	209
	Appendices	227
A1	Appendix 1 – Additional Details on the Goal Structuring Notation – GSN	228
A1.1	Safety Case Elements	228
A1.2	Safety Case Element Relations and Possible Expressions	230
A1.3	Steps for building a safety case goal structure	230
A1.4	On using Safety Cases.....	231
A2	Appendix 2 – Classification of safety mechanisms at a logical level	233
A2.1	Initial categorization of safety mechanisms at a logical level.....	233
A2.2	Final categorization of safety mechanisms at a logical level	237
A3	Appendix 3 Additional Implementation Examples in AF3.....	240

A3.1	Comparison pattern.....	240
A3.2	Range Check pattern	248
A3.3	1-out-of-2-Voting pattern	255
A3.4	Homogeneous Duplex Redundancy pattern.....	261
A4	Appendix 4 – Survey Questionnaire	275
A5	Appendix 5 – Survey results.....	287
A5.1	Raw Responses	287
A5.2	Aggregated Numerical Scores	290
A5.3	Supplementary Visualizations (3D Clustered Charts by Segment) ...	291

List of Abbreviations and Acronyms

Acronym	Meaning	Description
ADL	Architecture Description Language	A language used to describe software or system architecture.
AF3	AutoFOCUS3	A research-oriented CASE tool for model-based development.
ASIL	Automotive Safety Integrity Level	ISO 26262 classification of risk levels for automotive safety.
AUTOSAR	AUTomotive Open System ARchitecture	A standardized automotive software architecture framework.
CBSE	Component-Based Software Engineering	An approach that emphasizes design and development through reusable components.
COREPA	COmponent REadiness Preparation by Architecture analysis	An industrial reuse initiative targeting the readiness of reusable components.
DSL	Domain-Specific Language	A specialized language tailored to a specific application domain.
DSE	Design Space Exploration	A process to evaluate architectural or system design alternatives.
EAST-ADL	Electronics Architecture and Software Technology – Architecture Description Language	A domain-specific language for automotive embedded system modeling.
EN50126	European Standard for RAMS	A railway-specific standard defining Reliability, Availability, Maintainability, and Safety.
FMEA	Failure Modes and Effects Analysis	A structured method for evaluating potential failure modes.

Acronym	Meaning	Description
FTA	Fault Tree Analysis	A deductive failure analysis method used in safety engineering.
HDR	Homogenous Duplex Redundancy	A fault-tolerance strategy using two identical channels/components.
IEC 61508	International Electrotechnical Commission 61508	A basic functional safety standard applicable across industries.
IMBSA	International Symposium on Model-Based Safety and Assessment	A conference focused on model-based safety analysis.
INCOSE	International Council on Systems Engineering	A global professional society for systems engineering.
ISO 12207	ISO/IEC/IEEE 12207	Standard for software lifecycle processes.
ISO 15288	ISO/IEC/IEEE 15288	Standard for systems lifecycle processes.
ISO 26262	ISO 26262	An international standard for functional safety in road vehicles.
ISO 26580	ISO/IEC 26580	Standard related to software and systems product line engineering.
ISO 42010	ISO/IEC/IEEE 42010	Standard defining system and software architecture description practices.
MBSE	Model-Based Systems Engineering	A systems engineering approach emphasizing formal models.
MDE	Model-Driven Engineering	A development paradigm focused on creating and exploiting domain models.
MODPAL	Model-based Design Pattern Application Library	The proposed pattern-based reuse framework developed in this thesis.
OSI	Organizational Support Index	A derived metric assessing perceived organizational support in the survey.

Acronym	Meaning	Description
SAF	System Architecture Framework	A domain-specific modeling framework from the SAFE project.
SAFE	Safe Automotive software architecturEs	A research project and resulting modeling framework for automotive safety.
SIL	Safety Integrity Level	A risk-based safety classification from IEC 61508.
SMP	Safety Mechanism Pattern	A reusable safety solution formalized as a design pattern.
SoC	System-on-Chip	An integrated circuit combining all components of a system on one chip.
SPES	Software Platform Embedded Systems	A German research initiative resulting in a comprehensive modeling framework.
SysML	Systems Modeling Language	A general-purpose modeling language for systems engineering.
TMR	Triple Modular Redundancy	A fault tolerance technique using three redundant systems and voting logic.
UAF	Unified Architecture Framework	A framework for enterprise architecture modeling.
UML	Unified Modeling Language	A general-purpose modeling language for software and systems design.

Prologue

37 seconds after launch on its maiden flight on June 4th 1996, the European Space Agency's (ESA) Ariane 5 rocket veered off course and was destroyed by its automated flight termination system. The failure resulted in the loss of over US\$370 million and dealt a serious blow to ESA's launch program. It remains one of the most infamous – and expensive – software-related failures in history [1].

The cause: a reused software module from the earlier Ariane 4 series. In Ariane 5, increased acceleration caused a horizontal bias value, computed as a 64-bit floating point number, to overflow the 16-bit signed integer used to store it – triggering a processor trap. In Ariane 4, efficiency constraints had led developers to omit overflow protection on some variables, including this one. These constraints no longer applied in Ariane 5 – but the assumption of safe reuse remained.

Ironically, the faulty software component was reused in its entirety, despite parts of it (including the one that failed) being unnecessary in the Ariane 5 context [1].

The Ariane 5 case highlights a deeper issue in safety-critical engineering: **the risks of reusing complex design solutions without full contextual understanding**. Despite being well known, this issue persists. This thesis addresses core aspects of the problem by proposing a structured approach to contextualized, verifiable reuse.

1 Introduction

Software-intensive systems performing safety-critical tasks are becoming ubiquitous; from advanced driver assistance and hybrid-electric drivetrains, to autonomous rail transport, high-precision medical equipment, and coordinated drone operations. This growing complexity is fueled by the rising number of integrated control units, communication interfaces, and software modules. As system complexity increases, so too does the risk of design and implementation errors, particularly when architectural reasoning becomes difficult to trace or validate.

This risk is amplified when new development methods are combined with fast-evolving technologies, and an increasingly intricate international standardization and regulation landscape. Addressing non-functional requirements such as safety in later development stages is notoriously difficult and expensive, as stated in [2], [3] and [4]. As a result, developers often turn to trusted solutions – components or subsystems reused from previous projects – in the hope of reducing effort, cost, and time, while increasing reliability [5]. As evidenced by the Ariane 5 incident, this doesn't always work well.

Reuse is not a panacea. The assumption that reusing previously validated solutions inherently and unconditionally improves safety is a widely held **fallacy**.

As Leveson [6] aptly states: *“Although the reuse of proven software components can increase reliability, it has little or no effect on safety. In fact, **reuse may actually decrease safety** because of the complacency it engenders and because the specific hazards of the new system were not considered when the software was originally designed and constructed.”*

A long history of safety incidents supports this claim:

- **Apollo 11 (1969):** A radar subroutine, inherited from earlier spacecraft and no longer required, overloaded the guidance system's processor. Only a priority-based scheduling design prevented mission failure [7].
- **Therac-25 computer-controlled radiation therapy machine (1985–87):** Legacy components reused from the previous Therac-20 caused massive radiation overdoses due to different usage contexts. At least two patients died [8].
- **UK Air Traffic System (1988):** Software reused from the US failed to handle zero-degree longitude, misplacing aircraft and disrupting radar coverage over the Greenwich Meridian [9].
- **F-16 Navigation in Israeli Airspace:** Systems failed at low altitudes over the Dead Sea – negative values for *absolute* altitude *above* sea level, a case never considered in the original US-based software, led to serious operational incidents [6].

These issues are not limited to the past:

- **London Airspace Shutdown (2014):** A legacy software check dating from the 1990s triggered a system-wide failure in the UK's air traffic control system, grounding flights across Europe [10].
- **Boeing 737 MAX Crashes (2018–19):** The MCAS system, adapted from previous Boeing aircraft without fully revalidating assumptions for the new aerodynamic profile [11], led to two fatal crashes [12].
- **CrowdStrike Global IT Outage (2024):** A faulty software update reused legacy configuration logic without full compatibility testing, affecting airlines, banks, and governments globally, with estimated economic impact exceeding \$10 billion USD [13].

The pattern is clear: safety is not a property of components in isolation, but of the system in its context. Safe reuse requires more than copying – it demands understanding the assumptions, constraints, and conditions under which a component was originally validated. Without this, even "trusted" legacy elements may fail.

As recently as 2023, studies confirm that the practical use of design patterns in the automotive industry remains fragmented and ad hoc, with a persistent disconnect between academic approaches and industrial reuse needs [14].

This thesis focuses on the systematization of reuse for complex system solutions. It began with a technically scoped, high-impact example: the reuse of safety mechanisms – a subset of system solutions with well-bounded safety properties. This scope enabled the structured development and evaluation of a novel model-based reuse methodology.

The method was first demonstrated across multiple modeling frameworks within the scoped domain. It was then generalized and applied at scale within a large industrial organization, supporting the reuse of complex system solutions through extensive tooling, training, and organizational adaptation.

While the thesis does not claim universal applicability, it provides a mature and transferable foundation for other large organizations seeking to implement systematic reuse. The approach, lessons learned, and accompanying recommendations offer a realistic path for incremental adoption in similarly structured environments.

1.1 Motivation, Example and Scope

The reuse of architectural designs, development artifacts, and entire code segments is widespread in engineering practice, particularly in well-understood domains. This also extends to the reuse of safety-related components, including documentation structures that support the fulfillment of safety goals. However, the reuse is mostly **ad-hoc** and **opportunistic** [15], with **loss of knowledge and traceability** and **lack of consistency**

and process maturity being the most widely spread and cited drawbacks [16], and with practitioners tending to **limit their focus to a central reusable artifact** (category) [17] – typically implementation-related, e.g., a piece of code, an algorithm or a design model – to the detriment of other information necessary to tell the whole story, despite being aware of this [18].

Practitioners often focus on the most visible or concrete element – typically a piece of code, a model, or a configuration file – while neglecting the surrounding rationale, requirements, and architectural assumptions. This bias is not merely procedural but **cognitive**: the tendency to isolate one element as “the solution” reflects well-documented psychological phenomena such as *focalism* or the isolation effect, which lead engineers to assign disproportionate weight to a central artifact while overlooking the broader context. This occurs even when, objectively, that item by itself has little or no intrinsic value *when considered independently* of the other interconnected elements [19].

Additionally, it is possible that practitioners view these artifacts in very prominent relief due to them being viewed as the culmination of the *intellectually creative* steps of the Engineering work¹, as opposed to the more mundane tasks of testing the developed systems or documenting the steps carried out. This would be a manifestation of investment bias. While these considerations have influenced the approach developed in this thesis, the cognitive aspects of this problem are beyond the scope of this work.

To make this problem tractable, this thesis begins by focusing on a well-defined class of technical system solutions: **Safety Mechanisms** in the automotive domain. According to ISO 26262 (Part 1, p.15) [20], a Safety Mechanism is a

“technical solution implemented by E/E functions or elements, or by other technologies (mechanical etc.), to detect faults or control failures in order to achieve or maintain a safe state.”

While originally designed to address safety, many of these solutions have relevance for broader dependability goals, including reliability, availability, maintainability, and security (RAMSS). Systems and Safety Engineering are explained in more detail in chapter 2, while Safety Mechanisms get their own section in chapter 3.

Safety Mechanisms represent an ideal scoped example: they are sufficiently complex to require the reuse of multi-disciplinary artifacts and supporting evidence – some of which spans technical, procedural, and even organizational domains – yet they remain contained enough to be addressable within a practical research framework, making them non-trivial yet feasibly manageable within this research scope.

As presented by the author in [21], a typical reuse scenario within our context would involve reusing an architectural measure such as **homogenous (hardware) duplex redundancy** (illustrated in the next figure) where a safety-critical signal is duplicated

¹ Such a focus evokes elements of what George Lakoff referred to as “*Prototype Theory*” [22].

across channels to increase reliability. However, even when previous artifacts or argumentation structures are reused, they are often treated as detached, prescriptive checklists rather than comprehensive knowledge containers.

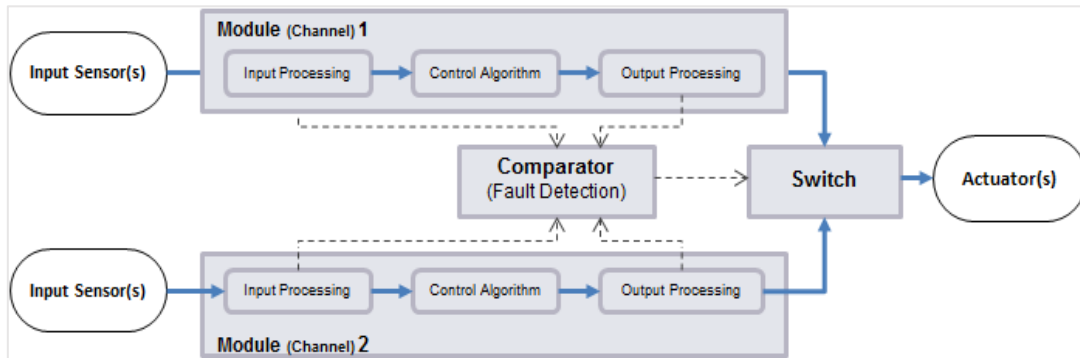


Figure 1: Homogenous (Hardware) Duplex Redundancy Pattern [21]

In reality, correct reuse of a Safety Mechanism such as homogenous redundancy requires significantly more information:

- The **requirements** it fulfills;
- the corresponding **logical architecture** and **deployment constraints**;
- supporting **tests**, **design rationale**, and **comparative trade-offs** (e.g., why two channels and not a 2-out-of-3 architecture?);
- **contextual constraints** such as the availability of a second hardware channel.

These considerations must be **explicitly captured and traceable**, not buried in developer memory or scattered across informal documentation. A 2019 comprehensive survey on safety pattern reuse [17] confirmed that while the literature focuses on reusable domain-specific assets, **cross-domain and cross-discipline integration** remains a major gap – especially where hardware, software, and organizational knowledge must interact; *exactly the scope of a complex system solution*.

Furthermore, although standards mandate early-phase safety analysis and architectural measures, these activities are rarely integrated into development environments in a structured, reusable way. Practitioners often lack tool support for **front-loading reuse-relevant information**, automating documentation, or embedding safety rationale into their architectural models [17].

What is needed is a way to encapsulate reusable technical solutions – including requirements, rationale, constraints, and evidence – in a **structured, tool-agnostic**, and **practical** approach. This approach must be:

- **Systematic** (to allow self-assessment and completeness checks);

- **lightweight** and iteratively **Scalable** (to avoid disrupting existing practices);
- **integrative** (to support real use inside actual development environments).

Implementing reuse in a **structured, repeatable and assessable manner** is a further problem, especially, since it is difficult to assess the quality or completeness of a reuse process, if there is no “**definition of done**”.

Most importantly, the approach needs to be usable in a practical setting; to capture and reuse knowledge regarding technical solutions within an engineering organization, to do so systematically, effectively, and without prescribing a tool or modeling language which practitioners are forced to migrate to – often the deal-breaker for many previously proposed solutions in research and practice, as will be discussed in the investigation of related works.

This thesis addresses these needs, with Safety Mechanisms serving as the **scoped pilot example**. They anchor the problem space and provide a realistic, high-impact setting in which to build and validate the proposed solution.

1.2 Problem Statement and Challenges

As summarized above, the reuse of complex system solutions suffers from the following recurring issues:

- a lack of systematicity;
- loss of critical contextual knowledge and traceability;
- process inconsistency and immaturity;
- poor integration with models, tools, and artifacts;
- narrow focus on isolated artifacts instead of system-level integration;
- inadequate tool support for capturing and reusing design knowledge.

These recurring issues crystallize into four interrelated challenges:

Challenge 1:

Systematically capture technical solutions² with critical contextual information for reuse in practice.

² We assume these are not isolated one-off solutions, but rather ones which form a set representing a class of solutions of a similar or related nature.

What information is required for correct reuse, and how can it be captured in a structured and repeatable manner?

Challenge 2:

Provide an efficient approach for solution reuse in practice, that improves the maturity, consistency, and effectiveness of solution capturing.

What is the “definition of done”? How do we guarantee a repeatable experience? Are we capturing our solutions well for reuse? How do we measure reuse maturity?

Challenge 3:

Integrate the approach seamlessly into practitioners’ development environments³.

How to do this within an organization? How do different tools and modeling choices affect the ability to capture reusable solutions?

Challenge 4:

Provide methodical guidance for evaluation and tool instantiation/selection support for approaches addressing Challenges 1 – 3

Do we have the right tools for reuse? How can teams assess their current toolchains and prioritize improvements for better reuse performance?

The main goal of this thesis is to investigate these challenges and provide a **systematic, tool-agnostic approach** to capturing a defined set of solutions (both as direct artifacts and contextual information) in a structured manner, targeting the direct and immediate practical reuse of a technical solution within an engineering organization. The approach must be tailorable to various development environments, without requiring any specific tool or modeling language.

1.3 Solution Proposal

To address the challenges of ad-hoc, fragmented, and tool-constrained reuse practices identified in the previous section, this thesis proposes a model-based, tool-agnostic approach to systematizing the reuse of complex technical solutions. The approach does not introduce a new modeling language or mandate the use of a specific toolchain. Instead, it aims to embed reuse-relevant information – such as rationale, constraints, and architectural dependencies – directly into the engineering artifacts already used by

³ We use the term development environment to refer to a practitioner’s (model-based) tool-suite; assuming the existence of a corresponding process description, method guide and development framework.

practitioners, enabling seamless, context-aware reuse. This is achieved via the novel concept of *operationalizing* Design Patterns⁴.

The method is piloted through the example of Safety Mechanisms in the automotive domain – solutions that are complex enough to require multi-disciplinary artifact integration, yet scoped enough to allow practical exploration. From this anchor, the approach is generalized to support larger-scale system solution reuse.

We call the approach the **MOdel-based Design PAttern Library (MODPAL)**. It consists of:

- A tailored **design pattern template** for structured reuse information
- An **artifact reference structure** that connects concepts, attributes, and artifact links
- **Guidelines** for instantiation, including roles for pattern *developers* and *users*.

The approach is instantiated and tested across multiple modeling environments, supporting flexible integration into different engineering contexts without requiring disruptive tooling changes.

1.4 Targeted Advantages and Practical Benefits

The proposed approach targets both immediate improvements and long-term strategic benefits for organizations aiming to improve the maturity and effectiveness of solution reuse.

We postulate the following four contributions, each corresponding to one of the key challenges outlined earlier:

- **Claim 1:** The approach enables **systematic capturing** of technical solutions and their reuse-critical attributes, moving beyond the current ad-hoc practices.
- **Claim 2:** The approach allows **assessment of how well tools support reuse**, identifying gaps in current tool capabilities.
- **Claim 3:** The approach supports the **prioritized extension** of existing tools to better support reuse practices.
- **Claim 4:** The approach enables **informed selection of modeling frameworks or tools** based on how well they support the structured reuse of complex system solutions.

⁴ Design patterns, originally introduced by architect Christopher Alexander and later adapted into software engineering; most famously by the Gang of Four [23], provide structured descriptions of proven solution *templates* for recurring problems. Chapter 3 of this thesis provides detailed information about this concept and its evolution.

These claims are empirically validated in the second half of the thesis:

- In Chapter 5, through a domain-independent instantiation in a research modeling environment.
- In Chapter 6, through integration with a domain-specific automotive safety modeling framework.
- In Chapter 7, through a longitudinal industrial case study demonstrating real-world application and organizational impact.

By addressing both technical and organizational aspects of reuse, this thesis provides a robust and extensible foundation for enabling systematic reuse practices in real-world engineering environments.

1.5 Research Methodology

Design problems in Systems Engineering, and especially in safety-critical applications, are rarely well-structured. Challenges stem from incomplete knowledge, conflicting stakeholder needs, evolving standards, infrastructure and organizations – and the integration of diverse technical domains. In this context, the selected methodology must offer a way to systematically identify the problem, explore its context, propose a support concept (e.g., a methodology, framework, or tool), and evaluate that support through both internal and practical validation.

This research follows the **Design Research Methodology (DRM)** developed by Blessing and Chakrabarti. DRM provides a structured and iterative framework for design research, aiming to simultaneously improve both the *understanding* of a design problem and the *development of support* that can improve the situation [24]. This dual purpose aligns well with the goals of this thesis, which addresses the complex and interdisciplinary challenge of solution reuse in safety-critical systems, with a focus on the automotive domain.

We selected this methodology, as, in our understanding, DRM is the most suitable approach for this thesis among the various design-oriented research frameworks, due to its balanced emphasis on both understanding the real-world complexity of design problems and developing validated support to address them. While Wieringa's Design Cycle [25] is well-suited for information systems and focuses on iterative problem-solving, it tends to prioritize the artifact creation process without deeply structuring empirical diagnosis or long-term evaluation. On the other hand, the Design Science Research Methodology (DSRM) by Peffers et al. [26] provides a clear stepwise procedure for artifact creation but is less well-equipped to handle the diagnostic and longitudinal aspects of industrially embedded design research. This latter aspect is better addressed by Consortium Research as described by Österle and Otto [27], which emphasizes collaborative industry engagement, yet lacks a robust framework for scientific validation

across exploratory, prescriptive, and evaluative phases. Most importantly, our thesis is concerned with complex systems design, leading us to Vaishnavi and Kuechler’s statement [28]: “Design Research *is research about design whereas Design Science Research is primarily research using design as a research method or technique.*”

In contrast, DRM explicitly supports a four-phase structure which aligns precisely with this thesis’s aims: to diagnose systemic reuse challenges, design an operational reuse framework, and validate it rigorously through both internal tool instantiations and a long-term industrial case study.

Blessing and Chakrabarti describe DRM as comprising four key phases [24]:

- 1) **Clarification of the Research Topic:** The problem is defined, its relevance to both academia and industry is justified, and research objectives are formulated. Success criteria and hypotheses may also be outlined. The aim is to ensure that the research addresses a *real* and *impactful* need, rather than an isolated technical challenge.
- 2) **Descriptive Study I (DS-I):** This phase seeks to understand the current situation in depth. It may involve literature reviews, observational studies, expert interviews, or empirical analysis to identify key factors that influence the problem and its potential solutions. The output of this phase includes requirements for the support that will be developed in the next phase.
- 3) **Prescriptive Study (PS):** In this phase, a solution is developed. This “support” may be a method, process, tool, or conceptual framework. The development is informed by the understanding gained in DS-I and aims to address the identified problems effectively.
- 4) **Descriptive Study II (DS-II):** The developed support is validated in practice. This phase assesses whether the solution is usable, useful, and effective in real-world scenarios. It often involves pilot studies, longitudinal deployments, or user feedback to evaluate the support against predefined success criteria.

This methodology was chosen because it ensures both scientific rigor and practical relevance. The integration of empirical inquiry with the development and validation of actionable support is especially important in this thesis, which targets practical reuse in complex, regulated development environments. DRM enables the research to maintain a strong link between theory, implementation, and evaluation; a necessary requirement for advancing both academic understanding and industry practice in safety-critical system reuse.

The research presented in this thesis spans almost 15 years from 2010 to 2024. References in the Clarification, PS, and DS-I phases (primarily 2010–2015) reflect the frameworks used and standards current at that time, to preserve historical and methodological

accuracy. Developments and updates relevant to the DS-II phase (2016–2024) have been incorporated where appropriate.

Due to the extensive longitudinal industrial case study used for the DS-II phase, our thesis can be most closely assigned a **Type 6** of the 7 DRM research type categories. The next section maps this thesis and its chapters to the DRM phases.

1.6 Structure of Thesis

The structure of this thesis follows the four phases of the Design Research Methodology (DRM) framework. Individual chapters correspond to a particular phase of the methodology, as outlined in the figure below. This structure ensures a rigorous and traceable development process – from identifying and analyzing a real-world problem, through the design of a practical solution, to demonstrating its effectiveness in both academic and industrial settings. This close alignment with DRM supports both the validity and practical relevance of the research findings.

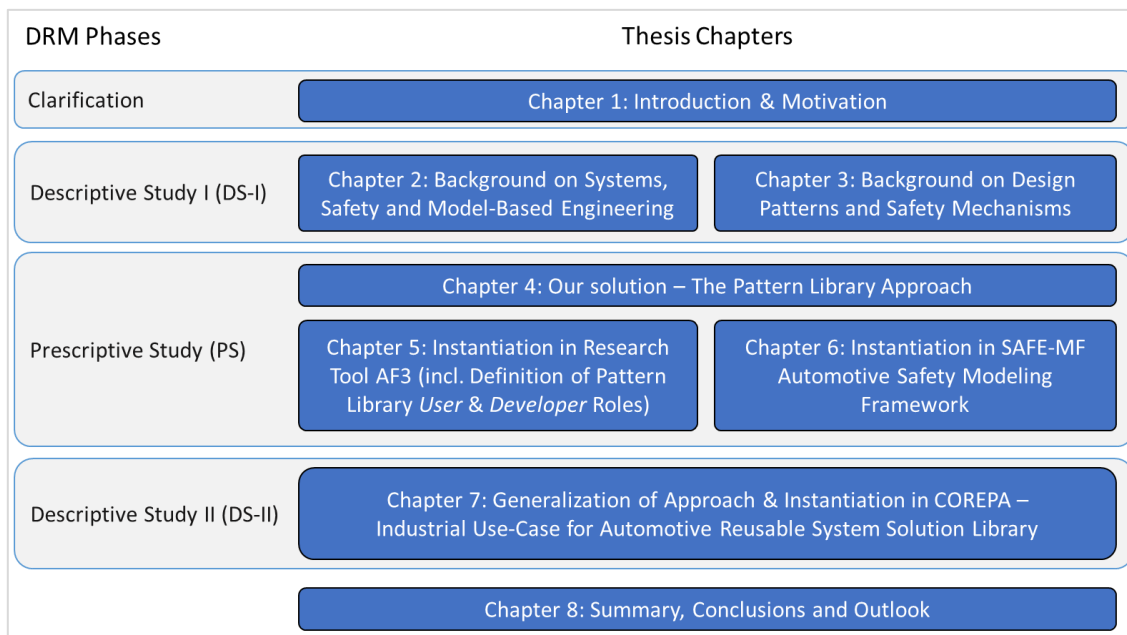


Figure 2: Thesis structure & mapping to DRM Phases

Chapter 1 – Clarification of the Research Topic

This chapter introduces and motivates the research. It outlines the core challenge of systematic solution reuse in safety-critical domains, supported by a motivating example. The research aims and claims are formulated, the thesis scope defined, and success criteria stated. In the context of DRM, this chapter represents the *Clarification* phase, establishing the need for a structured reuse approach and laying the foundation for the research work.

Chapters 2 & 3 – Descriptive Study I (DS-I)

These chapters provide an in-depth analysis and a comprehensive understanding of the problem space.

- Chapter 2 surveys the relevant state of the art and practice as well as industry standards, for systems engineering, safety-critical automotive applications, and model-based development.
- Chapter 3 shifts the focus to design patterns, safety mechanisms as a special design pattern category, and their relation to systematic reuse – analyzing existing templates, capturing documentation approaches, and highlighting the limitations of current pattern practices in safety-critical contexts.

Together, these chapters correspond to the *Descriptive Study I* phase, where the problem context is diagnosed in detail and requirements for a more effective reuse approach are identified.

Chapters 4, 5 & 6 – Prescriptive Study (PS)

These chapters describe the development and internal validation of the proposed solution.

- Chapter 4 introduces the Pattern Library approach for encapsulating reusable solutions, including a new template for Safety Mechanisms, an artifact structure, and guidance for instantiation.
- Chapters 5 and 6 demonstrate two implementations of the framework: first, in a general-purpose research CASE tool with targeted extensions based on gaps identified through evaluation; and then in a domain-specific pre-industrialization automotive modeling environment that inherently supports many of the reuse mechanisms.

These chapters collectively represent the *Prescriptive Study* phase, where the support is developed and evaluated in controlled environments.

Chapter 7 – Descriptive Study II (DS-II)

This chapter presents a long-term, large-scale industrial application of the generalized reuse approach. Over several years, the framework was instantiated and applied at a globally leading automotive Tier 1 supplier⁵, where it enabled systematic reuse of system architecture knowledge and the model-based synthesis of reference vehicle architectures. An expert survey provides evidence of its effectiveness, utility and impact. This

⁵ To maintain organizational confidentiality and focus on the generalizable aspects of the research, the company where the final instantiation was conducted is referred to throughout this thesis simply as "*the Company*."

corresponds to *Descriptive Study II*, validating the support in an operational industrial context.

Chapter 8 – Conclusions, Limitations and Outlook

This chapter summarizes the thesis, evaluates the validity of the research claims, and reflects on the overall contributions. It also identifies limitations and outlines areas for future research.

Appendices Overview

The thesis includes appendices that support the main chapters and can be read independently:

Appendix A1 provides a concise overview of the Goal Structuring Notation (GSN), which is introduced briefly in section 2.4.3, and used in this thesis. GSN is a structured notation for expressing safety cases. This appendix also adds notes on how safety cases are approached in this work.

Appendix A2 provides deeper details into the structuring work that went into implementing the safety mechanisms in AutoFOCUS3, demonstrating the completeness of the Safety Mechanism Pattern Library (discussed in Chapter 5), while **Appendix A3** presents additional Safety Mechanism implementation examples. Together, these additional implementation works were carried out in the context of a Master's thesis [198] supervised by the author and submitted in Sept. 2015, preceded by the publication of the author's contributions in 2014 at [196].

Appendix A4 presents the original questionnaire used to conduct the survey presented in chapter 7.3, while **Appendix A5** presents tables with the raw pseudonymized responses, aggregated intermediate results and additional visual synthesizations of the responses and results.

1.7 Previous publications by the author

Selected material presented within this thesis reflects contributions by the author which have been previously published in peer-reviewed conference papers, journal articles and research project deliverables.

They are listed in this section in chronological order.

The initial concept of a pattern-based approach for direct reuse was published in:

- Maged Khalil. (2013). "Pattern-Based Methods for Model-Based Safety-Critical Software Architecture Design: A PhD Thesis Proposal." In Software Engineering – SE Aachen 2013. ZeMoSS (Workshops). pp. 493-499. 2013. [183]

The initial concept for combining system description and development artifacts with safety argumentation artifacts into a consolidated **library for Safety Mechanisms** was developed and published:

- Khalil, M., Schätz, B., and Voss, S. (2014). "A Pattern-based Approach towards Modular Safety Analysis and Argumentation." Embedded Real Time Software and Systems Conference (ERTS 2014). Toulouse, France. Feb., 2014. [21]

That concept leveraged implementations from the ITEA2/EUROSTARS SAFE(-E) project deliverables, published in:

- M. Khalil and E. Metzker. ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D3.1.3 / D3.4 "Proposal for extension of Meta-model for safety-case modeling and documentation". The SAFE / Safe-E Consortium – www.safe-project.eu. 2013. [57]

Additionally, an earlier first proof-of-concept implementation was published in:

- Voss, S., Schätz, B., Khalil, M., Carlan, C. (2013). "Towards Modular Certification using integrated model-based Safety Cases". VeriSure 2013, Verification and Assurance Workshop, at CAV 2013. St. Petersburg, Russia. [200]

The concept for a reusable pattern library was further developed, **with** an initial version of **the Safety Mechanism Pattern Template, first implementations, and** the initial **pattern library** developer and user **workflows** published in:

- M. Khalil, A. Prieto and F. Hölzl. "A pattern-based approach towards the guided reuse of safety mechanisms in the automotive domain." In Model-Based Safety and Assessment – IMBSA 2014. pp. 137-151. Springer International Publishing. 2014. [196]

This leveraged implementations in the ITEA2/EUROSTARS SAFE(-E) project deliverables, published in:

- M. Khalil and E. Metzker. ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D3.1.3b / D3.4b "Updated proposal for extension of Meta-model for safety-case modeling and documentation". The SAFE / Safe-E Consortium – www.safe-project.eu. 2014. [197]

Additional implementations to demonstrate the completeness of the approach for covering known Safety Mechanisms from literature were carried out in the context of a Master's thesis [198] supervised by the author, and some are included in the appendices **A2** and **A3**, as additional support for the reader.

A more **mature form of the Model-Based Pattern Library approach**, including the current version of the **Safety Mechanism Pattern Template**, and an expanded pattern developer workflow, was published in:

- Maged Khalil. (2018). “Pattern libraries guiding the model-based reuse of automotive solutions”. 10th International Conference on Systems Analysis Modeling (SAM2018). Copenhagen, Denmark. 2018. [203]

The current form of the Model-Based Pattern Library approach (Chapter 4), with an expanded pattern user workflow, as well as **first reports of the results of the evaluation of implementations in AF3** (Chapter 5), the **SAFE framework** (Chapter 6), and the **first evaluations of generalizing the concept in industry** (Chapter 7), were published in:

- Maged Khalil. “Design Patterns to the rescue: guided model-based reuse for automotive solutions”. In Journal proceedings of the 25th ACM Pattern Languages of Programming Conference (PLoP2018). Portland, Oregon, USA. 2019. [184]

The generalization of the approach and its instantiation **in industry** (Chapter 7), led to the **identification** and prioritization of **reuse infrastructure gaps**, which were then **addressed in**:

- J. Eder, S. Zverlov, S. Voss, M. Khalil and A. Ipatiov. (2017). “Bringing DSE to life: exploring the design space of an industrial automotive use case.” ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017). Austin, Texas, USA. 2017. [217]
- J. Eder, A. Bayha, S. Voss, A. Ipatiov, and M. Khalil. (2018). “From Deployment to Platform Exploration: Automatic Synthesis of Distributed Automotive Hardware Architectures”. ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018). Copenhagen, Denmark. 2018. [220]
- J. Eder, S. Voss, A. Bayha, A. Ipatiov, and M. Khalil. “Hardware architecture exploration: Automatic exploration of distributed automotive hardware architectures.” In Journal of Software and Systems Modeling (SOSYM). Vol. 19. 911–934. 2020. [221]

These improvements leveraged earlier work published in:

- S. Zverlov, M. Khalil, and M. Chaudhary. (2016). “Pareto-efficient deployment synthesis for safety-critical applications in seamless model-based development.” In Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016). Toulouse, France. 2016. [206]

Finally, the **concept** for the repivoted **thesis** in its **current empirically-founded form** was published in:

- Maged Khalil. “Improving solution reuse in automotive embedded applications using a pattern library-based approach”. In proceedings of the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion 653–659 (MODELS 2019). [247]

All the presented work contained within the chapters of this thesis represents the original contributions of the author.

2 Systems and Safety

2.1 Research Methodology

This chapter constitutes a major part of the *Descriptive Study I (DS-I)* phase of the research, as outlined by the Design Research Methodology (DRM), with the second part being covered in chapter 3. It is grounded in an extensive, exploratory review of literature, standards, and domain-specific resources across systems engineering, safety engineering, and related disciplines. Rather than following a strictly systematic review protocol, this phase embraced a comprehensive, integrative approach, aiming to capture the breadth and depth of concepts, challenges, and best practices in the field. Together with the author's extensive industrial experience (prior to and after the beginning of this research work), the resulting synthesis forms the conceptual foundation for the subsequent design-focused investigations.

The first section introduces embedded systems and illustrates their most significant characteristics, including safety. The following section expounds on safety-critical systems and their specific traits as well as an overview of safety standards and their specifics. This is followed by an introduction to safety cases, the argument chains linking safety goals to evidence of their fulfillment; an important aspect of safety engineering demanded by many safety standards. The last section discusses Model-Based Software and Systems Engineering (MBSE) and its application to safety engineering.

Note on Reference Timeframes and Framework Versions

This thesis presents research activities spanning from 2010 through 2024. The Clarification, Descriptive Study I (DS-I), and Prescriptive Study (PS) phases were conducted between 2010 and 2015, while the Descriptive Study II (DS-II) phase — realized through a longitudinal industrial case study covered in chapter 7 — took place between 2016 and 2024.

References to frameworks, standards, and methods cited in the earlier phases are anchored to the versions relevant for, and used in, the corresponding works, and the state of the art in the 2010–2015 period. Updating these references retroactively would misrepresent the technological and methodological context in which the research was originally conceived, developed, and validated.

More recent developments, such as in Model-Based Systems Engineering, and Architecture and Systems Engineering Standards, have been incorporated, as they are directly relevant to the DS-II phase / the industrial case study. This approach ensures historical accuracy and preserves the coherence and integrity of the research progression over time.

2.2 Systems Engineering, Embedded Systems & Dependability

A **system** is fundamentally understood as an organized assembly of interacting components designed to fulfill specific objectives within a clearly defined context. The primary international standard for Systems Engineering, the ISO/IEC/IEEE 15288:2023 “*Systems and software engineering — System life cycle processes*” defines a system as

“a combination of interacting elements organized to achieve one or more stated purposes”[29].

This definition emphasizes the purposeful integration and interaction of elements to achieve designated goals.

Complementing this, the International Council on Systems Engineering (INCOSE⁶), in its *Systems Engineering Handbook* (5th Ed. 2023) [30], describes a system as

“an arrangement of parts or elements that together exhibit behavior or meaning that the individual constituents do not”.

This perspective highlights the emergent properties of systems, where the collective behavior or meaning arises from the interactions among the parts, surpassing the capabilities of individual components.

While the ISO 15288 standard does not provide a standalone definition of **Systems Engineering**, it outlines a comprehensive set of processes that constitute the discipline. These processes encompass technical, management, and agreement activities essential for the successful development and operation of systems. The standard serves as a framework for implementing Systems Engineering practices across various industries and applications.

INCOSE provides the following definition⁷ for Systems Engineering, which we will follow throughout this thesis:

“A transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems, using systems principles and concepts, and scientific, technological, and management methods.”

This definition underscores the collaborative and holistic nature of Systems Engineering, highlighting its reliance on systems principles and a blend of scientific, technological, and management methodologies.

⁶ International Council On Systems Engineering: a professional organization, founded in 1990, with over 26,000 members worldwide, that promotes collaboration, standards, and education in the field of Systems Engineering. www.incose.org

⁷ INCOSE. (2023). *Systems Engineering Definition*. International Council on Systems Engineering. Retrieved from <https://www.incose.org/about-systems-engineering/system-and-se-definitions/systems-engineering-definition>

Given these definitions, we can introduce Embedded systems, which are dedicated computers performing a predefined function, “embedded” within a larger mechanical and/or electrical system. Embedded systems are usually specifically built to carry out that task, as opposed to general-purpose computers, e.g., a desktop personal computer (PC), which can be used for many tasks. The specific nature of embedded system design is driven by two main factors: prior knowledge of the use case; that is, the exact function and usage the system is designed for, and resource limitations; the resources in question being computing power, power consumption, size and operating conditions, as well as price-per-unit. Depending on the application, embedded systems may also often have real-time computing constraints.

With the advent of digital computing using stored instructions sets in the 1950’s, many control operations requiring processing power and complex functionality were taken over by purpose-built computers. At first, and like many other innovations, this trend was driven by military applications – e.g., the D-17A missile guidance system [32] for the Minuteman I ICBM⁸. The move from transistors to integrated circuits in the 1960’s brought a steep drop in unit prices, heralding the availability of computing power for commercial applications.

Embedded systems currently cover a wide range of applications in many domains, from portable consumer devices, such as smart-phones and wearables, to large stationary installations, such as traffic lights or power plant supervision and control – and can be found in the depths of our oceans as well as in space. As embedded systems carry out specific tasks, design engineers optimize them to reduce costs and increase performance and reliability. To that end, embedded systems, though as a whole purpose-built, can rely on mass-produced components to benefit from economies of scale. The complexity may vary from a single dedicated controller, such as a digital watch, to large scale networked systems, such as cars, avionics and satellites. Software-intensive systems have become ubiquitous in our world, and hence the list is very long and continuously growing.

Embedded systems also commonly interact with their environments, continuously taking sensory input, providing control functionality and applying physical outputs through actuators. The interaction loop may also include human users, from whom the interaction with the environment is often hidden, instead limiting them to using dedicated input controls or graphical user interfaces (GUI). As part of this interaction, embedded systems often carry out critical tasks, whose malfunction or failure can have undesirable, and often dire, consequences. Governing a nuclear power plant, keeping an airplane in flight or a satellite in orbit, providing automobiles with safe maneuvering capability in adverse road conditions and ensuring the correct operation of an MRI⁹ and an ATM¹⁰ are a few examples of embedded systems carrying out such tasks. Dependability, defined in [2], [33] as “the system’s ability to avoid failures that are more frequent or more severe, and

⁸ Inter-Continental Ballistic Missile

⁹ Magnetic Resonance Imaging

¹⁰ Automated Teller Machine

outage durations that are longer, than is acceptable to the user”, is thus a very important aspect of embedded systems – one identified long ago [31].

There are various, mostly overlapping, definitions of dependability and which aspects it covers. We based the following sections on the concepts and taxonomy presented by Avizienis [33], listing the following aspects of the dependability matrix:

- Availability: timely readiness for correct service.
- Reliability: continuity of correct service.
- Safety: absence of catastrophic consequences on the user(s) and the environment.
- Integrity: absence of improper system alterations.
- Maintainability: ability to undergo modifications and repairs.

Security and Privacy, which cover access and information control, are also increasingly important aspects of embedded systems, and often added to the previous list or bundled with integrity. Safety is a primary focus of this thesis and will be discussed in more depth in the next section.

2.3 Safety-critical Systems

Safety is defined as “a system’s freedom from endangering human life or the environment” [34]. That is, *safety is a system property*, and as such safety-critical systems are those whose failure can lead to serious injury, loss of life, and/or significant property or environmental damage. These systems are thus subjected to a higher level of scrutiny and their design is a complex process, involving the integration of analyses and design strategies and techniques to fulfill two types of requirements:

- **Functional requirements:** cover the *functions* and *behavior* the systems is expected to perform.
- **Non-functional requirements:** represent *qualities* or attributes the system is expected to have, such as the dependability matrix introduced above.

This division can be misleading, as stated by Leveson [6] and van Lamsweerde [35], as it often affects the allocation of resources in software development projects, with the lion’s share going to fulfilling functional requirements that will “sell” the product. A false dichotomy is thus created, in which a project may either spend time and resources on non-functional issues; or be on time and within budget. The situation is further complicated by the fact that many of the aspects bundled under the “non-functional” moniker, despite not adding to the advertised functionalities of the new product, affect the functional system and software architecture by introducing new functions or mandating certain Design Patterns or constraints onto it. Case in point, many safety standards recommend (if not dictate) the early performance of analyses and introduction of architectural measures such as Design Patterns, e.g., the IEC61508’s recommendations on assuring independence in critical functions (IEC 61508, Part 1: 7.6.2.4) [36], as discussed in more

detail in section 2.3.3. The introduction of such measures at later development stages has proved to be a cumbersome and costly, if not downright detrimental, approach to systems development projects, as mentioned, for example, in [2], [3] and [4]. Because such analyses and architectural measures have to be conducted during the design phase of the system, they cannot be based on physical prototypes and implemented software, and, hence, have to rely on the design information in terms of requirements, structural design, functional specifications, and principled knowledge about the (nominal and potential deviating) behavior of the components used [37]. Functional requirements can be naturally expressed in discrete, logic-based formalisms. However, to express many non-functional requirements, designers need real-valued quantities to represent physical constraints and probabilities [38], which necessitate finding more adequate, context- and/or domain specific representations, as will be presented in section 2.5. Moreover, designers of safety-critical systems must ensure the correct integration of hardware and software components, which are usually developed independently from one another.

Keeping critical systems safe cannot be achieved by one technique, procedure or activity alone, but is rather a combination of all these aspects applied at various stages of the safety life-cycle, which stretches beyond the conception, design and development of safety-critical systems and into their operation, maintenance and eventual decommissioning. These activities are governed by standards, which share many commonalities, but must nevertheless address each domain's specific context and issues. The next section gives an overview on safety standards.

2.3.1 Safety Standards

Learning from mistakes, some of them quite costly, military and space programs were the first domains to collect techniques, procedures and activities into best-practices, guides and development manuals, which evolved into binding documents for the involved industries. Such standards were not only useful as gauges for product safety, but served as guidelines for the evolution of technologies and techniques. For instance, the U.S. Navy's conformance standards for testing computer systems and components, introduced in the early 1970's and used most significantly for early programming languages such as FORTRAN and COBOL, helped shape the development of high-level programming languages by identifying the challenges facing then current technologies, and led to significant convergence among the programming language dialects of the major computer vendors at the time [39].

In the ensuing years, various industries adopted their own guidelines and standards, often adapting those applied in neighboring domains to their specific use-cases. With origins in the process control industry, where Electric, Electronic and Programmable Electronic (E/E/PE) systems had steadily been taking over control functions from traditional hydraulic and mechanical systems, and targeting the consolidation of the many common aspects of safety engineering for E/E/PE systems, work began in the early 1990's on an international standard intended to cover basic functional safety for those systems and be

applicable to all kinds of industry. The result was the IEC 61508, first published in 1998, and titled “Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE)”.

2.3.1.1 IEC61508 – The E/E Meta-Standard

The IEC61508 covers the entire safety lifecycle – from concept through development and onwards till decommissioning – and targets the reduction of risks, calculated by combining the probability of a hazard occurring with its severity, to a tolerable level. While other technologies may be employed in reducing the risk, only those safety functions relying on E/E/PE systems are covered by the detailed requirements of IEC 61508.

Despite, or perhaps due to, its general applicability, the IEC61508 foresaw the provision of being further interpreted and tailored to suit specific projects or address industry sector-specific issues. Where sufficient need and momentum was reached, this manifested into standalone industry-specific standards, ultimately creating a landscape of IEC61508-derived or related industry-specific standards.

These standards share some common views on safety engineering:

- Zero risk is unachievable.
- Non-tolerable risks must be removed or reduced to acceptable levels.
- Safety has to be considered and planned from the onset.

An overview of the IEC61508 and related standards is given in Figure 3.

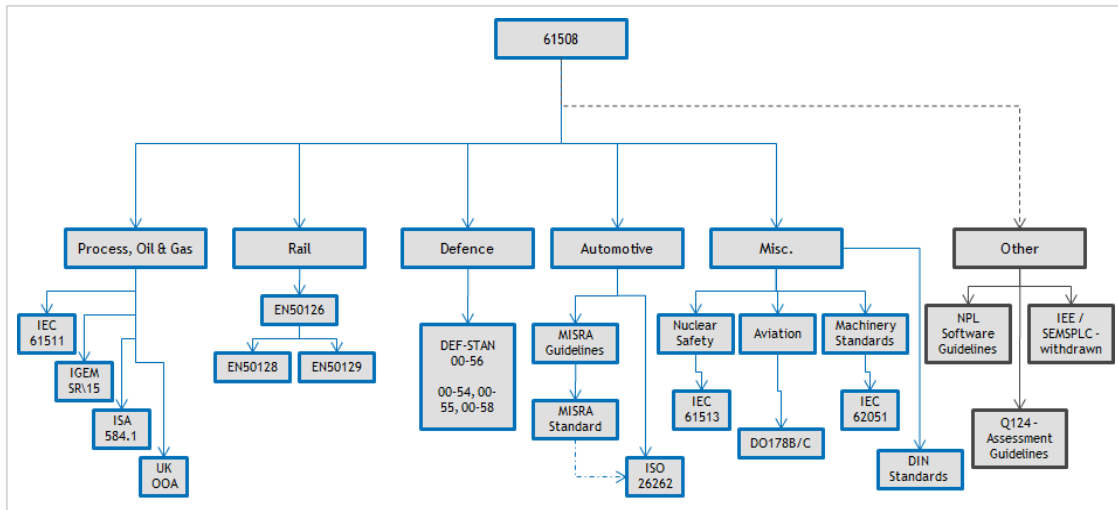


Figure 3: Overview of the IEC61508-related family of safety standards in 2015

Safety Integrity Levels

In its effort to consolidate concepts, the IEC61508 defined Safety Integrity as “the likelihood of a safety-related system satisfactorily performing the required safety function

under all stated conditions within a stated period of time”. Ironically, not all failures are created equal; the ramifications of a failure depend on the functions affected or hazards caused, such that some failures are nuisances, some are mild, whereas others are catastrophic. Furthermore, the very definition of what constitutes a “mild” or “catastrophic” result depends on the industry involved. This necessitated the creation of a quantifiable, universally understandable, metric to assess deviation from what is agreed upon to be tolerable risk. This “unmitigated” risk would have to be corrected by risk reduction measures. The unmitigated risk and the corresponding reduction measures were then categorized into levels. This essentially correlated the outcome of leaving the risk unmitigated with the orders of magnitude of the necessary risk reduction measures. The resultant Safety Integrity Levels (SIL), subdivided into 5 levels (0 – 4); ordered from lowest to highest, were thus a measure of both the unmitigated risk connected to a hazard and the effort necessary to mitigate said risk. Industry-specific interpretations of the IEC61508 tended to redefine SIL’s to better suit their domains.

Choice of Standards

For reasons of novelty and relevance (we use examples from the automotive industry), the ISO26262 is the standard we are basing our work on, and its definitions are the ones used throughout this thesis. One of the principal differences to IEC61508 is that the highest integrity level in automotive is roughly equivalent to SIL3. This determines the types of risk reducing measures relevant for automotive applications. Furthermore, automobiles are produced in large numbers and used many hours, which increases the importance of failure rates. As the work presented in this thesis spans over a decade, some of the relevant standards have been updated in the interim¹¹. We have nevertheless purposefully left the references to older versions of the standards unchanged in the sections where the research work was influenced by those standards at the time. Citations were updated to reflect the evolution of the work and the standards considered in it.

2.3.1.2 ISO26262 – Functional Safety for the Automotive Domain

Prior to the introduction of an automotive-specific safety standard, software development for safety-related systems in the automotive industry was primarily covered by the Motor Industry Software Reliability Association (MISRA) guidelines. Providing the first automotive industry-specific interpretation of the principles of the, then still emerging, IEC 61508, the MISRA guidelines, introduced in 1994, were conceived to provide recommendations for the development of embedded software in road vehicle electronic systems. Work on an automotive-specific tailoring of the IEC61508 continued, and would

¹¹ At the time this research work was beginning, the ISO26262:2011 was the most recent safety standard to be published for a major industry. It has since been superseded in terms of publication date by the DO-178C, the latest iteration of the avionics safety standard, published in 2012 and approved by the Federal Aviation Authority (FAA) in 2013. The second revision of the ISO26262, published in 2018, is the currently valid safety standard for the automotive industry. The 3rd revision is due in 2027.

culminate in the publication of the ISO26262 standard [20], titled “Road Vehicles – Functional Safety”, in late 2011¹².

The ISO26262 standard includes ten parts covering necessary activities and defining roles and responsibilities during the entire safety life-cycle, as well as recommending techniques and methods for development, and warning against known pitfalls. It addresses electrical, electronic, and software elements performing safety-related functions, as well as hardware requirements emanating from these components and the specification of the hardware-software-interface. Similar to other standards, it provides requirements for confirmation measures to ensure an acceptable level of safety is achieved.

Automotive Safety Integrity Level – ASIL

The ISO26262 also introduced automotive-specific safety integrity levels (ASIL), replacing the safety integrity levels (SIL) provided in IEC 61508, for a more domain-appropriate categorization of hazards and their risks. Like SIL’s, ASIL’s are also subdivided into 5 levels: (QM, ASIL A, ASIL B, ASIL C, ASIL D), ordered from lowest to highest risk. ASIL’s are determined by analyzing the probability, severity and controllability of a hazardous event. ASIL-D roughly maps to IEC 61508 SIL3. The automotive domain has no analogue to SIL4 hazards in its use cases. A more detailed overview of how ASIL’s are determined is provided in (Part 3: 7.4.3 Classification of Hazardous Event) and (Part 3: 7.4.4 Determination of ASIL and safety goals) of the ISO26262 standard [20].

2.3.2 Safety Assessments

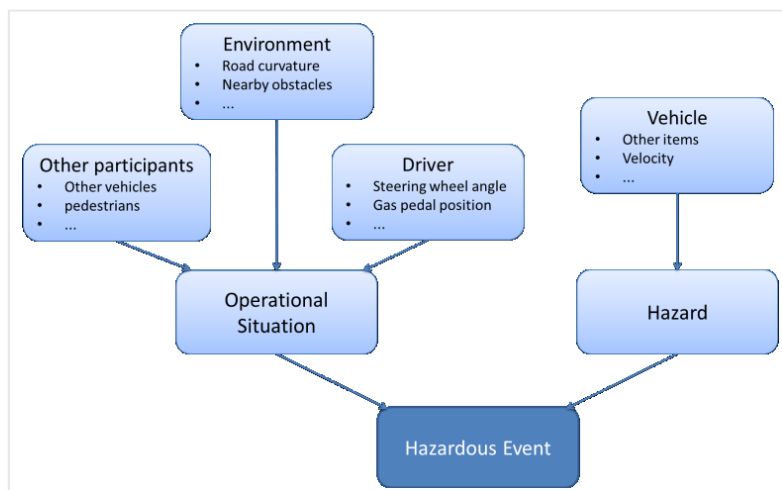


Figure 4: Description of Hazardous Event according to ISO26262 [40]

¹² The first 9 parts of the 10-part standard were released in November 2011, with Part 10 being released in late 2012. This version is used for most references in this thesis. Citations were updated to the 2018 version only if the relevant content had changed significantly.

Safety engineering targets the identification and management of hazardous events caused by hazards, defined by the ISO26262 as any “potential source of harm, caused by malfunctioning behavior” [20], occurring in combination with operational conditions, as depicted in Figure 4.

Malfunctioning behavior is caused by failures, which are the result of errors, which are in turn caused by faults. This is known as the Fault-Error-Failure Chain, explained in detail in the following section.

2.3.2.1 Fault-Error-Failure Chain

To understand this chain, some definitions are necessary:

- **Faults** are defects in the system, defined by the ISO26262 as an “abnormal condition that can cause an element or an item to fail” [20]. The presence of a fault – such as a memory bit flip altering the value of a stored variable – in a system may or may not lead to a failure, depending upon whether it causes an error, for example if the section of software containing the fault is ever executed.
- **Errors** are a deviation between the intended behavior of a system and its actual behavior inside the system boundaries, defined by the ISO26262 as a “discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition” [20]. Errors are invalid states which can propagate through the system and/or trigger other errors.
- **Failures** are instances where the system behaves contrary to its specification [2], defined by the ISO26262 as the “termination of the ability of an element to perform a function as required” [20]. Failures are observed at the system boundaries, and are basically errors which have propagated through the system to its boundary.

Having understood the mechanism of the fault-error-failure chain, it then becomes possible to take measures to break the chain and thus reduce risks. A hazard analysis and risk assessment (HA/RA) is performed to identify and categorize hazardous events and to specify safety goals and ASILs related to the prevention or mitigation of the associated hazards in order to avoid unreasonable risk [20].

As such, managing faults is one of the primary activities of safety engineering, with roots deeply anchored in dependability techniques. Various fault management techniques exist, as well as various ways to categorize them, as will be explained in more detail in the next section.

2.3.2.2 Fault Management

Fault management techniques can most aptly be categorized according to the development stage in which the technique is performed [33], [41]:

- **Fault Prevention:** deals with minimizing the number of faults introduced into the system during development, and can be accomplished by the use of good development and implementation methodologies and best practices.
- **Fault Removal:** aims at reducing the number of faults that were introduced into the system despite the Fault Prevention activities. This can be carried out into phases:
 - During development and before deployment through verification tests.
 - After deployment through inspections, failure records and maintenance and upgrade activities.
- **Fault Forecasting:** predicts likely future faults so that their effects can be circumvented or handled.
- **Fault Tolerance / Failure Management:** aims at preventing any remaining faults in the system (or external factors) from triggering failures by using mechanisms and design and development techniques for failure management.

Fault prevention, removal and tolerance are not alternative techniques but should rather be regarded as complementary approaches in an integrated fault management plan [42]. Fault tolerance and failure management techniques, as part of methods for reducing risks due to software failures, are the focus of this thesis, as will be highlighted in the next section.

2.3.2.3 Methods for Reducing Risks from Software Failure

One should differentiate between causes of failures in physical devices (e.g., random failures) and failures in software, which does not fail randomly. Software failures are systematic in nature, occurring, for example, due to erroneous implementation or design faults. Software safety is thus concerned with avoiding or protecting against these types of failures.

Safety standards, such as the IEC61508 [36] or ISO26262 [20], as well as literature sources such as [43], [44] identified and categorized methods of avoiding systematic software failures into:

- Safety analyses
- Development and design techniques
- (Re-)Using well-trusted designs, also known as Architectural Patterns

Safety Analyses

Inductive and deductive analyses are carried out on the system design to identify the causes of systematic failures and the effects of systematic faults. Examples of inductive analysis methods include Failure Modes and Effects Analysis (FMEA), Event Tree Analysis (ETA), and Markov modelling, while deductive analysis methods include Fault Tree Analysis (FTA), reliability block diagrams (RBD), and Ishikawa diagrams.

These methods each have their strengths and weaknesses, and combining them to yield better analysis and diagnostic coverage is standard practice. As an example, an FTA is a deductive, top-down approach, directed at analyzing the causes of failures in complex systems by identifying the initiating events and any activated faults. In contrast, an FMEA is an inductive, bottom-up analysis targeted at analyzing the effects of single component or function failures on subsystems or systems. While FTAs are a very good tool for demonstrating system robustness against single or multiple faults, they are not capable of identifying all possible initiating faults. On the other hand, an FMEA is capable of exhaustively cataloging initiating faults, and identifying their local effects. Analyzing multiple failures or their effects at system level is, however, not a strength of FMEAs. Furthermore, FTAs can consider external events, while FMEAs do not [45]. Consequently, FTA and FMEA are regularly used in complimentary fashion.

Development and Design Techniques for Safety

It is important to note the difference between defensive programming, which is an implementation technique; modular architectures, which is a design trait; and redundant deployment, which is an architectural pattern. The first two belong in this section, the latter in the following section.

Many of the development and design techniques included in this category can be found in “best-practices” or standardized development guidelines, and are also listed inside the respective sections of the safety standards. For instance, the ISO26262 [20] recommends the use of language subsets, strong typing, defensive implementation and unambiguous graphical representations, as well as the use of style guides and naming conventions and the enforcement of low complexity in its implementation guidelines (Part 6 – 5.4.7). It further specifically warns against using dynamic objects, global variables, unconditional jumps, implicit type conversions, recursions and hidden data or control flow (Part 6 – 8.4.4). If one is implementing in the C programming language, then such recommendations are covered by the MISRA C guidelines [46], [47], which can be applied by automated code-checkers to ensure conformance to these rules; a fact highlighted in the ISO26262 standard.

On the other hand, the IEC61508 [36] states “*From a safety viewpoint, the software architecture is where the basic safety strategy is developed in the software*” (Part 3). The ISO26262 – (Part 4) expounds on this, stating that first and foremost, architectural design should be modular, display an adequate level of granularity, and be simple [20].

It goes on to specify which properties of modular design are necessary for each ASIL, as seen in the following table, where (+) denotes recommended actions, while (++) denotes highly recommended.

Table 1: ISO26262 recommendations on modular system design properties [20]

Properties		ASIL			
		A	B	C	D
1	Hierarchical design	+	+	++	++
2	Precisely defined interfaces	+	+	+	+
3	Avoidance of unnecessary complexity of hardware components and software components	+	+	+	+
4	Avoidance of unnecessary complexity of interfaces	+	+	+	+
5	Maintainability during service	+	+	+	+
6	Testability during development and operation	+	+	++	++

This thesis focuses on the relationship between system safety and software-intensive system architecture elements, with an emphasis on modularity.

Architectural Patterns and well-trusted Designs

The concept of patterns or styles has been widely adopted in software architecture design in the past two decades. A pattern arises from the notion of reusing proven solution designs for similar problems. It defines components and connections, along with deployment constraints, that can be applied to software systems. Patterns have been identified for purposes varying from the very general, such as object-oriented programming paradigms [23], to the highly specific, such as patterns for deep machine learning [48]. The concept of patterns, its historical background and related works are covered in depth in Chapter 3, along with a comparison of relevant pattern *Templates*.

The ISO26262 (Part 4 – 7.4.3.4) clearly states, that to “reduce systematic failures, well-trusted automotive systems design principles should be applied. These may include the following:

- re-use of well-trusted technical safety concepts;
- re-use of well-trusted designs for elements, including hardware and software components;
- re-use of well-trusted mechanisms for the detection and control of failures; and
- re-use of well-trusted or standardized interfaces.”

This recommendation is, however, not without a caveat, as stated in the next point (Part 4 – 7.4.3.5). “To ensure the suitability of well-trusted design principles or elements in the new item, *the results of their application shall be analyzed and the underlying assumptions checked before reuse*” [20].

Thus, reuse is only recommended if the context and adequacy of usage within a new use-case are well understood and proven.

The ISO26262 (Part 6) defines different types of “mechanisms” – in fact, patterns – that can be used for fault avoidance, error detection, and failure management at the software architecture level. These include range checks and plausibility checks, but also external monitoring (e.g., software or hardware watchdog), control-flow monitoring, as well as diverse and redundant design [20]. ISO26262 refers to this class of solutions as *Safety Mechanisms*, and we will use that definition in this thesis.

Using such safety mechanisms has a high impact on the safety of the product as well as on the development processes and efforts involved, as discussed in the following section.

2.3.3 Impact of Architectural Patterns on Safety Engineering

Using architectural patterns is a widely spread method of dealing with problems in safety engineering [49]. As an example, let us consider cascading failures – one of the biggest challenges facing safety developers. They occur when the failure of a component or function triggers the failure of another, possibly unrelated, one. This becomes especially dangerous if the primary failure happens in a non-safety critical component, which then triggers a failure in a safety function; for example, the wiper blades prevent the brakes from working. Clearly, such a scenario is highly undesirable.

Yet if these components share common resources, e.g., they run on the same hardware, then it becomes difficult to prove the absence of cascading failures, also known as *freedom from interference*. Dependence and causality tests and failure analysis techniques become overwhelmed and inadequate with increasing complexity, especially if the cross-dependencies are not understood or realized by the developers and consequently are not included in the analyses and tests.

If freedom from interference cannot be proven, then the entire system housing the offending components inherits the highest safety criticality (worst setting) found in its constituent components. This is indeed mandated in safety standards, for example, the ISO2626 (part 9 – 6 Criteria for coexistence of elements) states that unless freedom of interference is proven, “*By default, when an element is composed of several sub-elements, each of those sub-elements is developed in accordance with the measures corresponding to the highest ASIL applicable to the element, i.e. the highest ASIL of the safety requirements allocated to the element*” [20]. In our example, this would mean developing wiper blade software to the same highly stringent standards as the braking software. This unnecessarily increases the complexity, effort and costs of development to a prohibitive level and should be avoided if possible. Architectural patterns offer solutions for such mixed-criticality setups.

2.3.3.1 Safety Architecture Patterns

Safety standards list various recommendations to solve this issue varying from mandating features in the software architecture to the explicit recommendation of architecture designs or patterns thereof. These solutions mostly revolve around the principles of software partitioning (separation of critical from non-critical software) and multiple redundancies (multiple versions, dissimilar or independent software). For instance, the IEC61508 states that the integrity requirements of a safety function can be satisfied by an architecture which combines parallel sub-systems of lower integrity, specifying that the allocation of integrity to these subsystems shall be carried out using appropriate techniques for the combination of probability, e.g., FTA and Cut-Set Analysis (Part 1: 7.6.2.4) [36]. This is referred to as decomposition. In the course of integrity allocation, systems which are treated as independent must satisfy the following set of requirements:

- They shall be functionally diverse (use of different algorithms to calculate the same results);
- They shall be based on diverse technologies (use of different type of equipment to perform the same result);
- They shall not share common operational, maintenance or test procedures;
- They shall not share common operational, maintenance or test procedures;
- They shall not share common parts or services whose failure could result in a dangerous mode of failure of all systems;
- They shall be designed so that the predominant failure mode for common support systems (e.g., power supply) is in a safe direction (i.e., fail-safe);
- They shall be physically separated such that foreseeable failures do not affect redundant safety-related systems.

If these requirements cannot be met, evidence should be provided that the probability of a dependent failure between the system elements is sufficiently low in comparison to the overall integrity requirements of the safety-related function (IEC61508, Part 1: 7.6.2.8) [36].

*Table 2: Criticality reduction by architectural measures in railway applications
EN50126 [50]*

Architecture	Most severe system failure condition / System development level	
	Catastrophic / A	Major / B
Partitioned Design	Level A for the partition Each partitioned portion can take a level corresponding to the most severe failure condition within it	Each partitioned portion can take a level corresponding to the most severe failure condition within it

Two dissimilar, independent design implementing a function	Level B for each portion	Level B for each portion
Two independent, partitioned designs implementing a function	Level A for the partition Level A for the primary portion Level B for the secondary portion	Level B for the partition Level B for the primary portion Level C for the secondary portion
Active monitor parallel design	At least one portion to level A The other portions to level C	At least one portion to level B The other portions to level C
Backup, parallel design	Primary portion to level A Backup portion to level C	Primary portion to level B Backup portion to level D

The railway safety standard EN50126 (Part 5) [50] went further and specified the potential safety integrity level reductions for several architectural patterns in an exemplary fashion for mixed-criticality systems. The examples, seen in Table 2, are highly helpful in understanding the impact of patterns on criticality, despite the fact that the mentioned patterns are all abstract variations on the redundancy theme. Note that hazards in the railway domain are described using the values A-B-C-D, from highest to lowest criticality, respectively.

The ISO26262 went even further and concretely suggested specific Safety Mechanisms for error detection and error handling at the software architecture level. A selection of the recommended mechanisms for error detection (Part 6 – 7.4.14) [20]:

- Range checks of input and output data
- Plausibility checks can be carried out against reference model of the desired behavior, as assertion checks, or by comparing signals from different sources
- Detection of data errors, e.g., error detecting code
- External monitoring, i.e., a watchdog – this can be provided by a dedicated ASIC or an external software component performing the watchdog role
- Control flow monitoring
- Diverse software design.

Having thus understood how architectural patterns and safety mechanisms can reduce criticality and efforts, benefitting safety-critical systems and software development, it is important to point out that the reductions specified in the table can only be justified if there is substantial evidence to support assumptions of independence (and in some cases dissimilarity) between the portions of these architectures. Providing evidence for the fulfillment of safety requirements and goals is the focus of safety cases, discussed in the next section.

2.4 Safety Cases – Claims, Arguments and Evidence

Safety assurance and certification methods vary considerably across industries, but also countries. Demonstration of compliance with rules and regulations is generally called an “assurance case”. If the goals to be met are safety related, then this demonstration is called a “safety case”. The purpose of a safety case is to present the argument that a system is acceptably safe to operate. This argument should demonstrate how the available evidence concerning the system can be reasonably and defensibly interpreted as indicating compliance with, and fulfillment of, the system safety requirements [16]. The safety case comprises all relevant information by and for all stakeholders, including the reasoning involved. This should then be collectively captured in an explicit (graphical or textual) form, and summarized in a report.

Depending on the product and location, assuring compliance with the governing rules and regulations usually involves demonstrating three types of activities:

- Adherence to design features and principles;
- Implementation of processes, organizational structures and policies; or
- Achieving some predefined goal or performance level.

A combination of all three types may prove necessary and is, in fact, the most prevalent form. The ratios are determined by the product under development, the industry, and the prevalent standards.

Based on these activities, a safety case presents “a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment”, whereby an argument is defined as “a connected series of claims intended to establish an overall claim” [51]. In attempting to persuade others of the truth of an overall claim, one must often make supporting claims. These claims may themselves then need further substantiation. This gives rise to a hierarchy of claims (representing a logical chain of reasoning) by which an argument is established.

To understand assurance or safety cases, one must understand their purpose, content, and context. On closer inspection, the first two types of activities listed above demonstrate following a predefined or *prescribed* rule or recommendation, reflecting *prescriptive* regulation. The third type of activity demonstrates achieving set targets, and thus represents *goal/performance-based* regulation. Types of regulation and their impact on safety cases are discussed in the next section.

2.4.1 Prescriptive vs. Goal-/Performance-based Regulation

The Flixborough explosion¹³ triggered the creation of the Health and Safety Executive (HSE) in 1974, which introduced earnest government oversight of safety in England, and

¹³ Chemical plant explosion in England, on 1 June 1974, which killed 28 people and seriously injured 36, out of a total of 72 people on site at the time. The cause was deemed to be shoddy maintenance.

shaped much of the ensuing regulatory activities worldwide. The term *safety case*, however, appears to have arisen as a result of the official inquiry report by Lord Cullen into the Piper Alpha disaster¹⁴. The Cullen report, published in 1990, was highly critical in its assessment of the state of safety in the industry. Among many other points, it concluded that safety assurance activities in the offshore oil industry were too superficial and generic, poorly scoped, overly mechanistic, and did not provide adequate proofs or consider interactions between components, systems and operators [52].

Most importantly, the report recommended the introduction of “safety cases”, emphasizing that regulation should be based around clear, measurable targets which would require that stated objectives be clearly and demonstrably met, rather than prescribing steps to be followed and documented [53], i.e., goal- and performance-based regulation instead of the hitherto predominantly prescriptive regulation. Such a change in perspective aimed at shifting responsibility for controlling risks from regulators to those who create and manage hazardous systems [54], and thus causing a paradigm shift in the mindset of the safety community; from demonstrating conformity to a standard, *to proving the safety of the system*.

This approach, adopted initially by the British Health and Safety Executive (HSE) [55] and applied widely to industries in that country, went on to influence other countries [56]. The ISO26262 reflects these changes in the focus of responsibility and features a high percentage of goal- and performance-based safety requirements. Analyses per ISO26262 can generally be categorized into *Audits* – which target *processes*, including activities, work products and documentation – and *Assessments*, which are concerned with evaluating the functional safety of the item, and hence focus on the characteristics of the *product*.

The types of evidence required and assurance arguments used depend on the type of regulation. The following section details types of arguments and evidence and the challenges in building a clear defensible safety case.

2.4.2 Goals, Arguments and Evidence

When dealing with prescriptive regulation, demonstrating compliance is usually quite straightforward. Goal- and performance-based regulation on the other hand, requires a more complex argument evaluation strategy, as well as changes in how the targets are specified and how the evaluation is performed. While the term “safety case” may be used in prescriptive regulation, “compliance” or “assurance case” are the more correct and predominant terms. “Safety cases” are more regularly used in a goal- and performance-based regulatory approach [56].

Safety cases have three fundamental building blocks; one must link *evidence* to the *goals* they prove the fulfillment of, via *arguments*. *Goals* are *claims* about requirements and

¹⁴ North Sea offshore oil and gas platform, which exploded in 1988, claiming 167 lives. At the time, it was the worst offshore oil disaster in terms of lives lost and industry impact.

constraints in compliance with standards. *Evidence* is proof of a solution via process or organizational compliance; analyses; tests, audits and inspections; reports and feedback; as well as logical, defensible rationales and concepts.

If the certification is based on following some product standard, e.g., cable material and thickness in a wiring harness, then inspections and material testing will build up the bulk of the *arguments* for the certification process. On the other hand, documentation, engineering artifacts and workflow analyses come to the forefront of certification, if the focus of the regulation is on processes.

Satisfying goal- or performance-based regulatory approaches may involve certain kinds of analyses, such as the use of specific types of probabilistic risk assessment to demonstrate “acceptable” overall failure rates. It may also allow include reasoning schemes that substantiate the claim of having achieved a particular performance goal.

Returning to the safety case definition given earlier, a safety case is “a documented body of *evidence* that *provides* a convincing and valid *argument*...”. This dual nature of safety cases, namely both a *documentation* of the fulfillment of a claim *and* the convincing *proof* that the solution is valid, is crucial. According to Weaver and Kelly [51], a commonly observed failing of safety assessments is that the role of safety argumentation is neglected. It is not uncommon for practitioners to flood the certification authorities with a large quantity of supporting evidence, e.g., hundreds of pages of fault trees or FMEA tables, with little done to explain how this evidence relates to the safety objectives. The (potentially overwhelmed) certifier can then only guess at any unwritten and implicit arguments. Both argument and evidence are crucial safety case elements that must go hand-in-hand. As Weaver and Kelly pointed out; arguments without supporting evidence are unsubstantiated, and therefore not convincing, while evidence without argument is unexplained, and hence not helpful in demonstrating how (or even if) safety objectives have been satisfied.

Additionally, certification may often be a one-time activity, performed once the development of a product is completed and immediately before – and most probably as a prerequisite to – market introduction. However, for complex systems with longer deployment life – examples include medical radiation therapy devices, chemical plants or aircraft – regular checks and oversight during operation may also be parts of the “continued worthiness” certification. Any changes to the original design – which served a basis for the initial certification – may necessitate partial or complete recertification, depending on the size of the change and the scope of the impact on the overall system.

In order for safety cases to be developed, discussed, challenged, presented and reviewed amongst stakeholders, and maintained throughout the product lifecycle, it is necessary for them to be documented clearly. The documented argument of the safety case should be structured to be comprehensible to all its stakeholders. It should also be clear how the evidence is being asserted to support these arguments [57].

Last but not least, complex projects generate a multitude of safety-related artifacts and even more non-safety-related ones. Establishing an intelligible argumentation path through this cacophony of artifacts is very difficult even for a system's developers. An external review of an argumentation chain under such circumstances becomes a nigh on impossible tasks, especially since simple tracing relationships – if they can be adequately maintained – are insufficient in describing the complex relations needed to describe how different artifacts come together to build an evidence chain arguing a claim [21]. Case in point, safety is a system property that can only be claimed within a specific usage context. It is part of the purpose of a safety case to define the context, within which safety is claimed [51]. This gives rise to the need for a dedicated “safety case”-specific view on the artifacts.

2.4.3 The Goal Structuring Notation – GSN

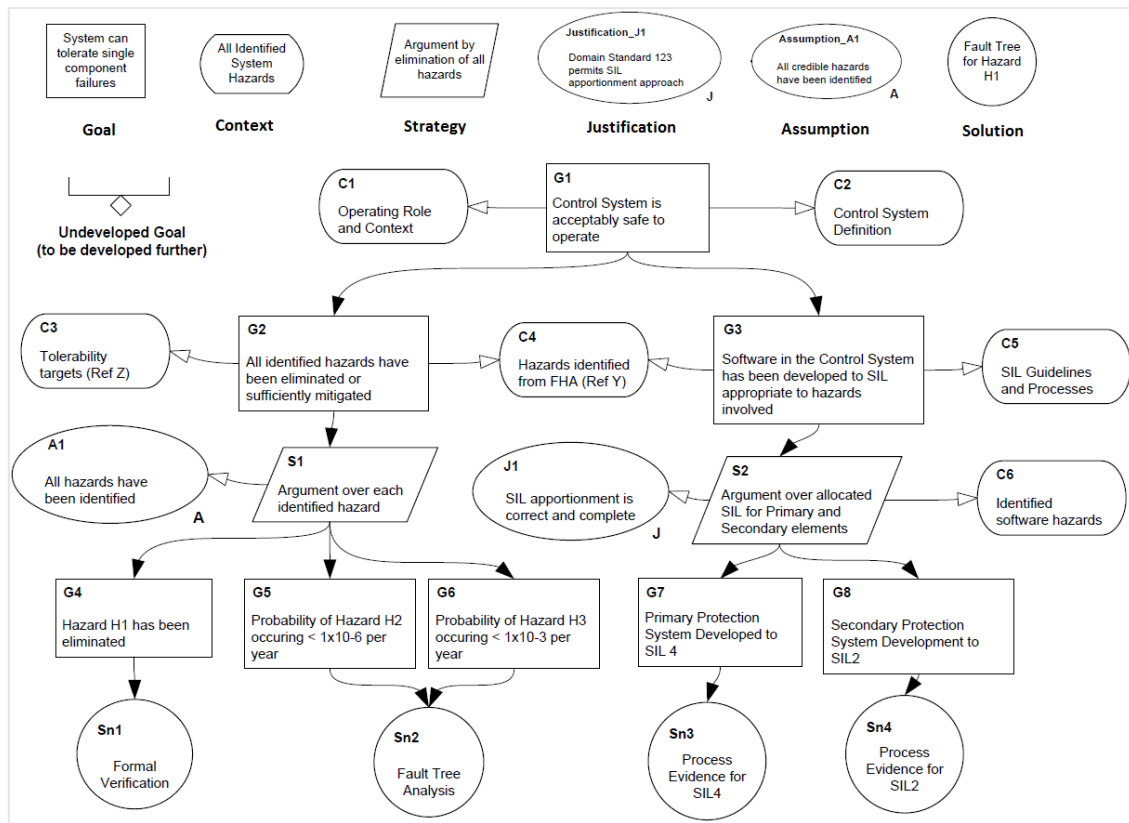


Figure 5: The Goal Structuring Notation – Elements and Example Argument Structure [58]

In our work we evaluated several approaches ways to express safety cases and settled on the Goal Structuring Notation (GSN), in its Community Standard Version 1.0 [58], as it was the most mature method appropriate for our needs. GSN is the result of long years of experience with the safety industry and the analyses of numerous argumentation documents, which was focused into a unified approach by the High Integrity Systems Engineering (HISE) Group at the University of York in England, spearheaded by John

McDermid and Tim Kelly. The GSN standard first appeared in November 2011¹⁵ and the specification document is readily available for free download. A short comparison with other approaches is provided in section 6.2, with more details provided by the author in [57].

GSN is a structured (yet not formal) notation, which provides graphically differentiated basic safety case element types, such as goals, evidence, strategy, and justification, as well as a clear description of necessary and allowed connection types between these elements. A selection of GSN safety case elements, as well as an example safety case argument structure are provided in Figure 5. A more detailed description of GSN is provided in Appendix A of this thesis.

2.4.4 Potential Limitations of Safety Cases

Safety Cases, when well-founded, -formulated and -documented, have been widely used in many industries with a good deal of success [51]. A safety case can thus be a powerful help for the development of complex systems. This, however, is not without a caveat.

A safety case is only as good as the information it encapsulates. Its basic elements – goals, arguments and evidence – are susceptible to misrepresentation and abuse, ultimately yielding a less-than-solid safety case.

Broy and Stolen [59] listed the misidentification, misinterpretation and misrepresentation of requirements, and the specifications based on them, as the biggest and most prevalent problems facing complex software-intensive projects in their critical early phases. Yet requirements and specification are the basis of every safety goal, without which important safety aspects will not be covered. In an experiment by Fischhoff et al. [60], a group of safety practitioners with mixed experience levels were presented with fault trees, from which significant parts had been purposefully removed. In a not entirely surprising, yet nevertheless alarming turn of events, both the novices and experts failed to consider the omitted information in their arguments, even though the experts could be expected to be aware of this missing information. Vicente and Rasmussen ran related experiments [61], in which expert participants were *knowingly* given partial information. The participants demonstrated a cognitive bias towards relying too heavily on the specification present – and failing to include other obvious information – despite full knowledge of its incompleteness. In essence, the report concluded that an incomplete specification proved more detrimental than having no specification at all.

Even with well-defined claims, building well-founded safety cases is a difficult task. Experiments have repeatedly shown that people tend to favor information which confirms their suppositions and aligns with their goals [62], and prefer “positive” testing, in which the search is biased towards evidence consistent with their current hypothesis and questions are phrased to receive affirmative answers [63]. This cognitive phenomenon is

¹⁵ The most current version at the publication of this thesis is Version 3, available at <https://scsc.uk/r141C:1>

known as *confirmation bias*. Engineers and developers wish to build safe systems and do so to the best of their abilities. This can, however, also mean that they build safety cases to show just that, instead of critical ones that attempt to demonstrate how the system could be unsafe [64]. Safety engineers should employ the exact opposite mindset, and try to find ways in which the system could be unsafe.

Even pioneers and champions of safety cases, such as Kelly, state that the mere creation and existence of a safety case “does not imply a safe system” [65], and highlight the need for rigor in drawing up argumentations, the integrity and validity of evidence, and the maintenance of the safety case throughout the safety life-cycle. The authors went on to confirm the threat posed by confirmation bias, and offered organizational actions to mitigate the negative outcome. They concluded that a “safety case should be thought of as another means towards safety and not as an objective”.

This is not surprising, as the use – or rather abuse – of safety cases had already been cited as a major contributor in accident reports. The Nimrod aircraft crash¹⁶ is a prime example. The 2009 inquiry report by Sir Haddon-Cave [66], was particularly scathing of the Nimrod’s safety case, calling it “... *a lamentable job from start to finish...*”, in which “... *the task of drawing up the Safety Case became essentially a paperwork and ‘checkbox’ exercise.*” The Nimrod report went on to make clear recommendations for the improvement of the safety case approach, centered on two main concepts:

- Safety Cases should be used within a rigorous, ongoing, and integrated development and operational process, and not just performed once and forgotten.
- Safety Cases are aids to the safety engineering process and thus a means to an end, rather than being an end unto themselves. “Compliance-driven” safety case documentation is worthless and should be avoided at all costs, as it gives a false sense of safety.

Leveson analyzed the Haddon-Cave report and others and reiterated the conclusion that the safety case regime should refocus on being a tool for identifying safety gaps, listing the following important characteristics for their correct usage [55]:

- Safety cases should be started early. If used properly, it can identify problems early in the concept phase and influence design decisions. This is opposed to the practice of using safety cases for a posteriori documentation of finished systems, providing a snapshot of the system “as-built”, instead of being a development aid to safety practitioners.
- Safety cases and their constituent components and structure should be monitored continuously throughout the safety life-cycle, to identify any changes in the design or usage of the system.

¹⁶ 2006 crash of a British Royal Air Force Hawker-Siddeley Nimrod reconnaissance aircraft over Afghanistan, which claimed the lives of 14 service members.

- To facilitate the monitoring and maintenance of safety cases, the safety case analysis and documentation should be included into the Systems Engineering approach and the system architecture description.
- The safety case description should be comprehensive, included as much information as necessary and possible, including system wide considerations.

Particularly the integration of safety case analysis and documentation into system architecture description in a holistic comprehensive approach is one of the main focus points and aims of this thesis. This integration requires incorporating multiple viewpoints on the system under development, each of which focus on an aspect or system property through an adequate representation thereof, which is often based on a mathematical model of the system. This is known as model-based engineering, as explained in the following section.

2.5 Model-based Engineering

The rising complexity of embedded systems, discussed in the first section of this chapter, and the fact that complex embedded systems typically include multiple components from different domains integrating diverse technical disciplines, makes it very difficult to manage system descriptions with any level of efficiency. Analyzing the interactions and dependencies of these systems is a necessary yet daunting task using traditional engineering methods. In order to use computational aides, such as computer aided software engineering (CASE) tools, one must first represent the systems mathematically, essentially creating a *model* of the original.

The many different notations used in the involved disciplines, combined with the system size and complexity make creating a complete representation of the entire system quite impractical. Instead, developers rely on representations of the system or parts of it, which focus on those aspects that need to be studied in each instant. A model is thus defined as “a simplified description, esp. a mathematical one, of a system or process, to assist calculations and predictions” [67]. Thus, a model, as discussed in [68], is a *depiction* of the original, a *reduction* focused on capturing relevant aspects, and it has a *purpose*, defining its usage.

Model-Based Engineering (MBE) can thus be defined as an approach to the design, development and analysis of software and systems, based on exploiting models and computer-based automation to reduce complexity and achieve improvements in both productivity and quality.

As surmised by Schätz [69], it is important to note that the focus – and strength – of model-based techniques lies not on the construction of a simplified description of one system or process; that is, modeling an *instance* of said system or process, in object-oriented parlance; but rather to abstract such specifics away and characterize a *class* description, allowing the reuse of concepts. At the risk of sounding circular, these classes

themselves can be – and often are – also models. Given the great promise that model-based techniques offer [70], MBE enthusiasts continuously tout the “everything is a model” principle [71]. To avoid confusion, it is thus necessary to provide some definitions for terms that will be used throughout this thesis. These definitions are in line with those introduced by the Object Management Group’s (OMG) standardization of their Model Driven Architecture (MDA) paradigm [72], as well as those provided in the SPES Framework [73] and its follow-up extensions.

Finally, we will introduce the INCOSE definition of MBSE (Model-Based Systems Engineering, which we will use as a reference for the practical applications of this thesis.

2.5.1 Modeling Vocabulary

One of the key strengths of modeling is the ability to capture concepts, rather than describe implementations. This aspect distinguishes it from implementation technologies such as source code, making it an excellent way to represent, understand and analyze systems [72]. To do this, models may take many forms and have many purposes. A *Domain Model*, for instance, may represent many aspects of a system; its business or hardware and software or any other domain-specific trait, while a *System Model* may include how the system’s many components interact to perform their function, as discussed by Broy and Steinbrüggen [74].

2.5.1.1 Models and Modeling Languages

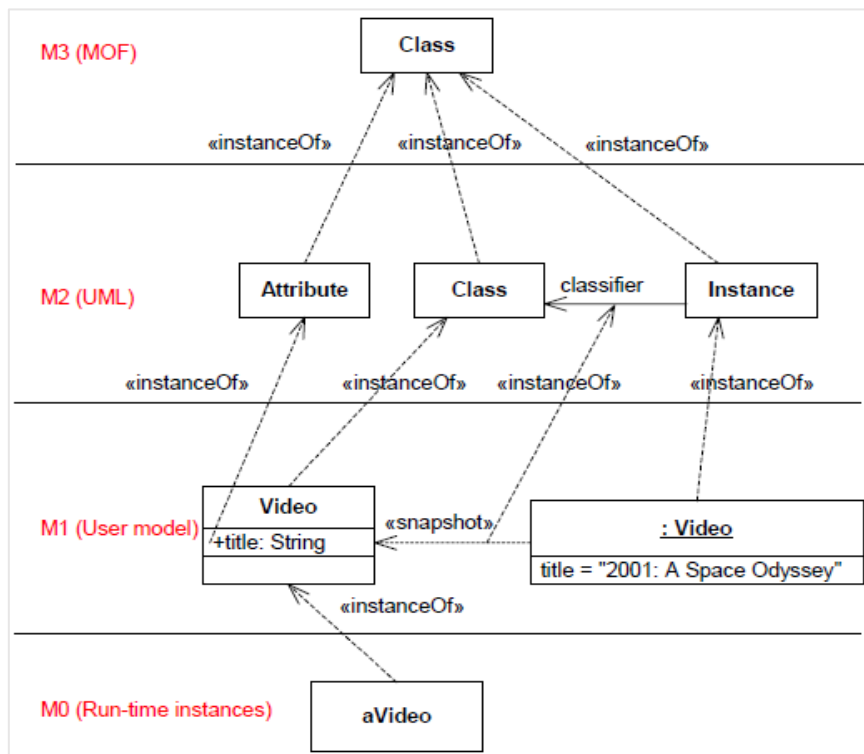


Figure 6: Modeling levels from the OMG MDA approach [75]

The terms, notations, semantics and structure as well as any integrity rules used to describe the model constitute its *modeling language*. Interestingly enough, more often than not, the best way to represent a modeling language is itself a model – this can be referred to as a *meta-model*. The meta-model defines the language used to describe the model, by defining what constructs the language employs and what their relationships are. One can abstract further onto the modeling language that defines and describes these constructs and relations; one speaks here of a *meta-meta-model*. This structure is shown in Figure 6, which explains the paradigms of OMG’s MDA approach using an example. *M0* represents the instance level, *M1* the model, *M2* the meta-model and *M3* the meta-meta-model, as defined in [75].

The salient point when dealing with different representations, is that the stakeholders need a common understanding of the model, so it can be useful. Thus, the model should be expressed in a language understood by the stakeholders and adequate for their domain.

2.5.1.2 Architectural Views

Following the essence of modeling paradigms, it is often useful to abstract away unnecessary information and arrive at a (partial) model capturing the system in a certain context, e.g., the privacy concerns of an electronic health care registry. Such a *view*, is a special representation of the system conforming to a *viewpoint*, which is a set of rules for the selection, creation and representation of an information contingent about the system, which addresses some particular stakeholder concerns. This definition, from the 2014 second version of the OMG MDA standard [72], aligns with similar concepts from the IEC Standard 1471 [76] (and its successor IEEE Standard 42010 [77]), which introduced a conceptual framework for architectural descriptions based on *architectural viewpoints*. Architecture is then defined as: “*The fundamental organization of a system, embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution*” [76]. Architectures can be described and specified using *Architecture Description Languages* (ADL). An example of ADLs, the EAST-ADL, is introduced and discussed in some detail in section 6.1.2.1.

We will especially be focusing on views and viewpoints, for which our thoughts are quite aligned with the form introduced by the SPES Modeling Framework (SPES MF) [73], and the definitions provided in the ISO42010¹⁷ “Software, systems and enterprise — Architecture description”. The SPES MF is an artifact-oriented approach (that is, it defines structures, concepts and content, rather than prescribing process and methods), supporting the seamless model-based software engineering of embedded systems. The SPES MF can readily be used to integrate software engineering and Systems Engineering activities, such as those set forth by the ISO/IEC 12207 standard for software engineering processes and ISO/IEC 15288 Systems Engineering processes standard, as described by Böhm et al in [79]. Among many other features, the SPES MF defines artifact types and

¹⁷ The earlier version ISO42010:2011 was used for DS-I and PS phases of the work done in this thesis. Citations were updated to the 2022 version [80] for the DS-II phase.

their relationships for different engineering concerns and different levels of system granularity. In the engineering of embedded systems, different stakeholders, e.g. requirements engineers, system architects, test engineers) have different interests in the system and its representation. The SPES MF introduces four different viewpoints to promote modularity and higher development quality by following the principle of separation of concerns; these are a *requirement*, *functional*, *logical*, and *technical* viewpoint, respectively. Furthermore, and to manage complexity, the SPES MF introduces system granularity layers orthogonally to these viewpoints. The granularity layers allow decomposition, such that tasks can be broken down to manageable size, essentially following a divide and conquer approach once more. The viewpoints and granularity layers mainly aim at structuring the artifacts created during the development¹⁸.

2.5.1.3 Model-driven vs. Model-based

Model-driven development (MDD) emerged as a generalization of the OMG MDA approach – first introduced in a white paper [81], then ratified by the OMG community in 2001 [82] – which aims at extending the use of modelling languages beyond design into programming. MDD relies primarily on models as the central artifacts for development, linking different views and their models (from the representation of business case through architecture description and right down to code generation) to one another through *model transformation*, which is defined as the generation of different models from an original model based on a transformation pattern. *Architecture*, in this context, is a representation of the system under development by means of a set of models, aiming to understand the system and stakeholder requirements and reach a design to satisfy these requirements [72]. Understanding the relations between models, and expressing them as precisely as possible – or formalizing these relations mathematically, if possible – is very important, because it enables correct reasoning about the architecture and its safety, towards the ultimate goal of achieving a “correct-by-construction” design.

Most importantly, model-driven approaches stress the seamless reliance on models as first-class entities through model transformation. Brambilla et al. [83], state this as being the most significant differentiating factor, defining model-based engineering (MBE) as a more watered-down, broader term denoting methods in which models play a central role in various process steps, yet seamless transformations across the process chain are not predominant. The fact that the OMG owns the copyright on model-driven architecture (MDA), model-driven development (MDD), and a host of related terms, for which they have supplied their own definitions, only adds to the confusion. In this thesis we will be using the term model-based to denote any development approach which relies primarily on models, without limiting the scope to those with seamless model transformations.

¹⁸ These concepts are of course not new; Parnas advocated a layered architectural pattern in 1979 [84], because it facilitates extension, contraction and restructuring of the architecture, achieved by using the ability to add or remove components defined at higher levels, which in turn use services provided by components at lower layers of the architecture.

2.5.2 Model-based Systems Engineering (MBSE)

This chapter provides a foundational understanding of MBSE and SysML, positioning them as key enablers of modern Systems Engineering practice. The following chapters will explore how MBSE methods are applied in specific domains and integrated into the systems development lifecycle.

2.5.2.1 Definition of MBSE

As with all topics in this space, there are several definitions used for Model-Based Systems Engineering (MBSE).

We will preferentially use the definition provided by the International Council on Systems Engineering (INCOSE), which has played a foundational role in defining and evolving MBSE practices. In its 2007 publication, *Systems Engineering Vision 2020* [85], INCOSE defined MBSE as:

"The formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases."

This definition underscores MBSE's role as a formal, lifecycle-wide method for improving clarity, consistency, and traceability in Systems Engineering, and highlights its role as a transformative approach in Systems Engineering that emphasizes the use of formalized models as the primary means of information exchange, design, and analysis (Models as first-class citizens). Rather than relying on traditional document-based approaches, MBSE promotes the integration of digital models throughout the entire system development lifecycle.

INCOSE updated and expanded this view in its 2020 publication *Systems Engineering Vision 2035* [86], envisioning MBSE as central to digital transformation across the engineering enterprise: *"The future of Systems Engineering is model-based, enabled by enterprise digital transformation."*

Further, it states: *"Model-based Systems Engineering, integrated with simulation, multi-disciplinary analysis, and immersive visualization environments is standard practice."*

This perspective elevates MBSE from a modeling technique to a comprehensive and integrative strategy for digital engineering.

2.5.2.2 Contrasting Definitions

Other authoritative sources¹⁹ provide complementary perspectives on MBSE.

OMG (Object Management Group) defines MBSE as:

"The practice of developing a set of related system models that help define, design, analyze, and document a system under development" [88].

This view highlights the practical utility of related models and the emphasis on standardization and tools, such as SysML.

In *"A Practical Guide to SysML"* [89] **Friedenthal, Moore, and Steiner** – pioneers of MBSE and its most widely used modeling language SysML – describe MBSE as:

"The use of models to support the engineering of a system, providing a more rigorous basis for capturing requirements, architecture, behavior, and verification".

This definition emphasizes the clarity and rigor models bring to complex system development.

2.5.2.3 SysML: The Standard Modeling Language for MBSE

The Systems Modeling Language (SysML) is the *de facto standard* modeling language used in MBSE practices. It is a general-purpose graphical modeling language designed to support the specification, analysis, design, verification, and validation of complex systems.

SysML originated as a profile and extension of the Unified Modeling Language (UML²⁰), tailored to address Systems Engineering needs, but hence evolved into its own language. The initiative to create SysML began in 2001, led by a consortium of industry experts and tool vendors and later formalized by the Object Management Group (OMG).

SysML continues to evolve to meet the growing complexity of engineered systems and to better align with digital engineering and MBSE principles. Table 3 provides a timeline of major events in SysML development.

¹⁹ NASA adopts a definition closely aligned with INCOSE's original vision: "The use of models to support system requirements, design, analysis, verification, and validation activities, beginning in the conceptual design phase and continuing throughout development and later lifecycle phases." [87]

²⁰ The Unified Modeling Language is a general-purpose visual modeling language that is intended to provide a standard way to visualize the design of a (software) system. It was developed in 1994-1995, and was adopted as a standard by OMG in 1997, and as the ISO/IEC19501 in 2005

Table 3: Significant milestones in SysML development

Year	Event/Version	Description
2003	SysML Initiative Formed	Led by the SysML Partners to create a UML-based modeling language for Systems Engineering.
2006	SysML v1.0	Official release by OMG; introduced support for requirements, behavior, structure, and parametrics.
2007-2017	v1.x Updates	Incremental improvements including better tool interoperability, bug fixes, and performance enhancements.
2017	SysML v2 Kick-off	OMG initiates the formal definition phase for SysML v2 to address limitations of SysML v1 and meet future digital engineering needs.
2019	SysML v1.6	Maintains backward compatibility while improving various diagrams and clarifying semantics.
2023	SysML v2 Beta	Major revision focused on semantic precision, scalability, and improved tool integration.
2025+	SysML v2 Release	Anticipated adoption phase for SysML v2, emphasizing interoperability, executable modeling, and web-native architecture.

2.5.2.4 Visualizing SysML: Diagram Taxonomy and Examples

SysML uses multiple new dedicated diagram types to capture useful information for Systems Engineering [88]. For example:

- A Block Definition Diagram (BDD) illustrates the system's structural hierarchy by defining blocks and their relationships. For instance, in an automotive system, a BDD can depict components like Engine, Transmission, and Suspension, showing how they relate to the overall Vehicle system.
- In contrast, an Internal Block Diagram (IBD) provides a detailed view of the internal structure of a single block, showcasing how its parts interact. Continuing with the automotive example, an IBD for the Engine block might display components such as Fuel Injector, Spark Plug, and Piston, along with the connections and data flow between them.

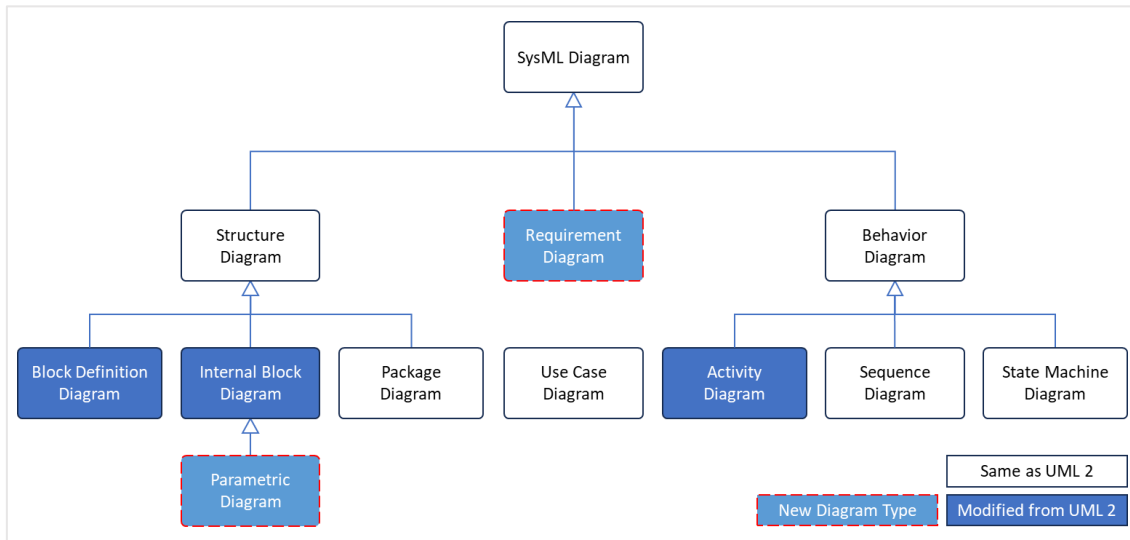


Figure 7: SysML primary diagram types

SysML comprises nine primary diagram types, seen in the figure above, and categorized as follows:

- **Structure Diagrams:** Block Definition Diagram (BDD), Internal Block Diagram (IBD), Package Diagram, Parametric Diagram
- **Behavior Diagrams:** Activity Diagram, Sequence Diagram, State Machine Diagram, Use Case Diagram
- **Requirement Diagram:** Captures system requirements and their relationships

SysML-based MBSE approaches provide a comprehensive capability for modeling various aspects of complex systems. SysML (v1.x) is used in the practical use-cases in this thesis, especially in the industrial use-case provided in chapter 7.

2.5.3 Model-based approaches to Safety Engineering

Traditionally, safety engineers had to synthesize information from numerous sources, including informal design models and requirements documents, in order to run safety analyses, such as FTA or FMEA. Unfortunately, such analyses are highly subjective and depend on the practitioner's skill and experience. Moreover, different safety engineers will often produce substantially different fault trees for the same system. A final version is usually only reached through a process of review and consensus building between the system and safety engineers [90]. Even then, there are no guarantees that the analysis results will be complete and consistent, or that they will remain current, as the analysis is essentially run on a parallel collection of data (that of the safety engineer's), that is not integrated in the system engineering approach and system architecture description.

Having understood the basic concepts of model-based engineering (MBE) and model-based Systems Engineering (MBSE), a quick reflection on the earlier sections detailing

safety-critical embedded systems readily yields numerous opportunities where model-based techniques can be applied to enable or greatly facilitate the many activities and analyses required for the development of safety-critical systems. For instance, in addition to a functional model of the system under development, “dysfunctional” or failure models can be introduced as well, allowing automated fault propagation analyses and greatly improving failure management. Indeed, the field of model-based problem solving [91] has generated theoretical foundations, prototypical solutions, and products for modeling artifacts and natural systems and for using models to solve a variety of tasks, with a major focus on automated diagnosis, but also solutions for failure modeling as well as for automated model-based software FMEAs, such as [92] and [93]. GSN itself is essentially a model-based approach, which provides a *safety case* specific viewpoint on the system under development.

Model-based development is increasingly popular in the development of embedded software systems in the automobile industry [94]. The information captured in these models is a treasure trove for safety engineers, yet safety analyses and argumentation are not well integrated into safety architectures, with the integration mostly limited to research tools [95].

Addressing the integration of safety and security in system architecture modeling is an important and popular research topic. For instance, Kharatyan et al. [96] present a metamodel enabling the combined analysis of these critical aspects. The proposed modeling language supports the synthesis and analysis of system architectures by incorporating safety and security considerations into a unified framework. This integrative approach allows for the early identification of potential failures and attack vectors, thereby enhancing the robustness of system designs in safety-critical applications.

The integration of safety aspects into the Systems Engineering approach, allowing model-based safety analysis, is of two-fold importance. First and foremost, the integration enables and facilitates the consistent performance of safety analyses *on the same artifacts used to develop the system*. Changes to the system architecture and design can be immediately analyzed, without the need for external – and potentially error-prone – synthesization of information. Secondly, thanks to architectural viewpoints and abstraction layers, the impact of changing requirements or implementation platform details can be decomposed down into manageable portions. This makes investigating and evaluating the fulfillment of safety goals a much more manageable task, as well as increasing the quality of safety cases, which can now directly incorporate the relevant development artifacts, in a structured manner that not only allows tracing changes, but can actually help engineers identify potential deficiencies in the safety argumentation. The integrative combination of these aspects allows for a practical *co-evolution* of System and Safety modeling artifacts, or more broadly, System Development artifacts and contextually-relevant System Description artifacts.

These points provide strong motivation for the approach presented in this thesis.

2.6 Summary

This chapter introduced embedded systems and highlighted their increasingly pervasive nature in everyday life, performing many functions with a potentially safety-critical impact. The concept of Model Based Engineering was introduced and its advantages expounded. The special nature of safety-critical applications was explained, with an overview of safety standards and their most relevant activities and aspects.

The reuse of well-trusted patterns, mechanisms and concepts has been shown to be beneficial and highly recommended, but only if the context of the reuse and the suitability for the new use-case can be established – and demonstrated in a safety case, which defensibly establishes the fulfillment of all underlying safety requirements. Such a demonstration would be greatly facilitated by employing model-based techniques, which integrate safety aspects into Systems Engineering approaches.

A framework for the modular reuse of safety mechanisms, with a provision for instantiation into various model-based development environments, and the potential for expanding this framework to more general types of complex system reuse problems beyond safety mechanisms, is the aim of this thesis.

The next chapter continues the motivation for this thesis, by introducing design patterns and their application to high-level architecture and design issues, as well as safety mechanism patterns and their documentation.

3 Design Patterns and Safety Mechanisms

A *pattern* is defined by the Oxford Dictionary²¹ as:

- *A repeated (decorative) design; an arrangement or design regularly found in comparable objects*
- *A regular and intelligible form or sequence discernible in the way in which something happens or is done:*
- *A model or design used as a guide (in needlework and other crafts); a set of instructions to be followed (in making a sewn or knitted item); a wooden or metal model from which a mold is made for a casting:*
- *An excellent example for others to follow.*

As the definition shows, patterns are as old as time; with design and solutions being copied and repeated for aesthetic as well as functional reasons. Interestingly enough, some of the most widely cited works on architecture design patterns in the software community are the works of a *building* architect, Christopher Alexander. In his book “The Timeless Way of Building” [97], Alexander argues that “*Beyond its elements, each building is defined by certain patterns of relationships amongst its elements*”. Alexander demonstrated how one could abstract away the particular details of a building, and then proceed to identify and capture those traits quintessential to the design – and its success, or lack thereof. These *patterns*, which he elegantly defined as a “*solution to a problem in a context*” in the same book, can then be used elsewhere.

With the advent of object-oriented programming – which, among others, championed the notion of reusable high-level concepts – in the late 1980’s and early 1990’s, the software community adopted Alexander’s concepts into what has since become known as *Design Patterns*.

Design Patterns can thus generally be defined as an abstract representation of a (trusted) solution for a general class of (repetitive) problems. Their main purpose is to support system architects and designers by not only lowering costs through not reinventing solutions but also increase reliability by reusing trusted solutions and, most importantly, increase quality by frontloading good design paradigms into the system development. Moreover, the concept of Design Patterns promotes good documentation of proven design techniques and thereby facilitates and simplifies communication between practitioners.

This chapter introduces the concept of Design Patterns, enumerating examples for different purposes, and discussing the capturing of patterns in Templates. This is followed

²¹ Oxford Dictionaries. “Definition of Pattern”. Oxford University Press.
<http://www.oxforddictionaries.com/definition/english/pattern>.

by introducing Safety Mechanisms, the solution pattern category defined in the ISO2626 automotive functional safety standard, which is the focus of this thesis.

The primary aim of our approach is to make Safety Mechanism reuse within an organization better (more systematic, repeatable, effective and efficient) by approaching Safety Mechanisms as if they were Design Patterns. We understand that Safety Mechanisms are – strictly speaking – not design patterns, and we are not primarily aiming at capturing a new kind of Design Pattern. We will use the structure provided by the design pattern template and general rigor of the process of defining a design pattern to improve solution component capturing for reuse in a practical setting.

3.1 Research Methodology

This chapter constitutes the second part of the *Descriptive Study I (DS-I)* phase of the research, as outlined by the Design Research Methodology (DRM), with the first part being covered in the previous chapter.

Since its introduction to Software Engineering in the late eighties, a vast body of work has been written about Design Patterns in Software and Systems Engineering. Our aim was to find Design Pattern Literature relevant to Software and System Design and Architecture in general – but with a special focus on safety-critical applications in embedded systems, and derive such attributes as are relevant for the capturing of safety-critical Design Patterns. These attributes would not only help capture Safety Mechanisms but serve as requirements for any model-based approach aiming at enabling the reuse of Safety Mechanisms in practice.

Our literature review was systematized by performing it in an approach adapted from systematic mapping studies as introduced in [98]. A systematic mapping study is a research methodology for reviewing and structuring a body of related work concerning a research area - we had successfully used this approach in a similar research endeavor involving a collaborative effort to identify requirements for a model-based Requirements Engineering tool in [99]

The purpose of our review was to identify commonalities in the resulting Design Patterns' Templates. A Design Pattern *Template* is defined as a construct for capturing Design Patterns by collecting their most relevant *Attributes*. Attributes are usually captured in the form of structured prose supported by diagrams. More details on Design Pattern Templates are given section 3.4.

Our search was organized in two phases. The first phase was a broad effort to collect works relating design patterns and safety and derive the common template attributes. In the second phase we focused on safety mechanisms – and any efforts to collect them and provide for their reuse in practice – and expanded to approaches targeting similar reuse problems.

We focused our search on three main digital sources for general literature: the IEEE Explore Digital Library, Google Scholar and the ACM Digital Library. We initially used search strings containing keywords such as “Design Pattern” or “pattern” in combination with either “safety”, “reliability”, “dependability” or “security” to identify generally relevant Design Pattern attributes. In the second phase we added the terms “library” or “repository” or “framework” or “approach” and “reuse” to identify related work. We also performed an extended manual search in the leading conferences on Design Patterns – Pattern Languages of Programming (PLoP); as well as its European siblings EuroPLoP and NordicPLoP – to increase our results and cover fundamental works. The results of the second phase, focusing on reuse especially in the context of system solutions and design patterns, are discussed in the next chapter.

The initial batch resulted in over a thousand papers, which we manually reduced via a process of elimination based on the relevance of the title. For the papers where the titles alone did not allow a decision, we read the abstract to determine their relevance. This reduced the works down to a little over one hundred, including several books. We then proceeded to prioritize these works based on precedence and citations, as we were more interested in earlier works, which usually influenced later publications. For instance, a very large number of Design Pattern papers published this millennium use the POSA Template (or variations thereof), which was introduced by Buschmann et al. in 1996 [100], and which is itself largely based on the earlier works of Riehle and Züllighoven [101].

This chapter concludes the *Descriptive Study I (DS-I)* phase of the research, as outlined by the Design Research Methodology (DRM). The next three sections give the most relevant results of phase one of our review.

3.2 A Brief History of Design Patterns

The idea of software Design Patterns was first suggested by Cunningham and Beck in 1987 when they proposed a number of software Design Patterns to describe elegant Smalltalk user interfaces [102]. Around that same timeframe, Coplien started to document what he called *idioms* for C++; essentially a programming-language-specific *Pattern Template*. This was used as an internal C++ guide at AT&T, where Coplien was working at the time, before being published in a condensed form in 1992 as part of a book on C++ programming styles [103], and eventually in a more expanded form in 1997 [104]. Sharing the “Alexandrian” heritage and discussing many of the same issues as Coplien, Coad independently catalogued patterns for object-oriented analysis and design in 1992 [105], with a focus on the concept of Design Patterns at a higher level of abstraction.

Parallel to these activities, Gamma started to document recurring design structures in object-oriented software development as part of his doctoral work in 1991 [106]. He went on to expand this work as part of the “Gang of Four” (Gamma, Helm, Johnson and Vlissides) [107], culminating in the 1994 publication of the first book dedicated to the

subject – “Design Patterns – Elements of Reusable Object-Oriented Software” [108]– a seminal work which firmly established Design Patterns as a relevant field in software development.

Much attention has since been given to Design Patterns. Several papers addressing patterns had already been presented between 1990 to 1993 at the ACM OOPSLA (Object-Oriented Programming Systems, Languages, and Applications) conference, with increasingly dedicated focus to the topic, and counterparts such as the European Conference on Object-Oriented Programming (ECOOP). 1994 also saw the organization of the first annual PLoP (Pattern Languages of Programs) conference, whose revised papers are normally published in book form, as part of the “Pattern Languages of Program Design” series, e.g., [109].

In 1996, Buschmann et al. published the book “Pattern-oriented software architecture: a system of patterns - POSA” [100], which included a catalog of relatively abstract solutions to common design problems and signaled increased interest in architectural patterns. The POSA template, largely based on the works of Riehle and Züllighoven [101], is currently one of the most widely used templates for software and system design patterns.

Appleton provided a good documentation of the genesis and ancestry of Design Patterns in Software in [110], which is a recommended read.

The adoption of the Design Pattern paradigm was not limited to software development. The concept was applied to hardware design and development, e.g., [111], and included importing useful elements of software development paradigms, e.g. importing object oriented techniques into System-on-a-Chip (SoC) design [112]; using UML to capture hardware design process patterns [113]; or using models to capture hardware Design Patterns for reuse [114].

Combining hardware and software, embedded systems – discussed in section 2.2 – were next in line. Building on his previous work in [115], Douglass introduced a noteworthy catalog of Design Patterns for embedded systems [116], which focused on real-time aspects such as concurrency, deployment, resource constraints and resource sharing, as well as reliability.

The establishment of Design Patterns as an important area of research encouraged its application to a variety of increasingly specialized and domain- and / or discipline-specific contexts; with efforts targeting everything from fault tolerant software designs [117]; to time-triggered embedded systems [118]; telecommunications [119]; information security [120], [121]; and redundant design for safety [122]; among many examples.

Last but not least, it is a natural leap of intuition from reusing a design principle, to reusing the corresponding argumentation structure used to show fulfillment of the design targets. McDermid and Kelly introduced the concept of safety case patterns [16], and Kelly went

on to pioneer and champion the idea in his doctoral dissertation [123] and through later collaborations, e.g., [51], culminating in the GSN standard [58], introduced in section 2.4.3. Based on Kelly's work on general safety case patterns, Weaver suggested a software-specific safety case pattern structure [124], Wu collaborated with Kelly on safety case pattern abstractions for failure management patterns in safety, which they dubbed "safety tactics" [44] (discussed in more detail in section 3.3), while Ye adopted the concept for arguing the use of Commercial-Off-The-Shelf (COTS) components in safety-critical development [125].

Whether one is documenting building design, or software design, each of these Templates had to document their Design Patterns. The documentation changed according to the design principles to be captured, the domain involved, and the characteristics they wished to highlight, yet followed common general principles. As Riehle and Zühlke stressed [101], *"patterns are not merely technical, but also capture the professional's domain specific understanding, guided by the underlying design metaphors"*.

The following section introduces Safety Mechanisms, the pattern category defined in the ISO2626 automotive functional safety standard, and the focus of this thesis. The section after that illustrates the documentation of Design Patterns.

3.3 Safety Mechanisms

The choice of what is and is not a pattern is a matter of perspective. *"One person's patterns can be another person's primitive building blocks"*, stipulated the Gang of Four [108]. For instance, while arguably also patterns, this thesis does not consider class encoding descriptions, such as linked lists and hash tables, or class relationship characteristics, such as "polymorphism". Similarly, we do not include high-level, system safety engineering argumentation design patterns, such as ALARP – As Low As Reasonably Practicable [126].

In this thesis we deal with the particular class of patterns known as Safety Mechanisms, which target fault and failure management, as recommended by functional safety standards and introduced in Section 2.3.3.

As a reminder, a Safety Mechanism is defined by the ISO26262 International Automotive Safety Standard [20] as a "technical solution implemented by E/E functions or elements, or by other technologies (mechanical etc.), to detect faults or control failures in order to achieve or maintain a safe state". Safety Mechanisms are concrete instances of a category of implementation-independent solution descriptions targeting safety. The solutions used in a "Safety Mechanism" are in fact not limited to safety but may also be useful for all manner of dependability or RAMSS – Reliability, Availability, Maintainability, Safety and Security – issues.

In his book on Design Patterns for real-time embedded systems [116], Douglass dedicated a chapter to safety and reliability patterns, such as "Triple Modular Redundancy" or

“Sanity Check”. Wu und Kelly described a very similar grouping of architectural designs or “Tactics” in [44] and proceeded to describe a design pattern and template for capturing them. Tactics capture the abstract principles or primitives of a design pattern. Not wishing to be bogged down by implementation details, they sought to capture the abstract notions behind safety mechanisms using the concept of abstract architectural themes. Pioneered by the likes of Riehle and Zühlke [101], who differentiated between concrete *patterns* and their overarching architectural *motifs*, and canonized by Bass et al. at the Software Engineering Institute (SEI) at Carnegie Mellon University as architectural *tactics* [127], the aim is to identify and collate the underlying primitives of a pattern independently of specific implementations, of which there are many variations. This makes the cataloguing easier, as there are obviously far fewer *tactics* than actual *Pattern* instances [128]. *Tactics* were proposed by SEI for six quality attribute categories (availability, modifiability, security, performance, usability, and testability) [127].

As a quick reminder of the Fault-Error-Failure Chain discussed in section 2.3.2, faults are defects or abnormal conditions in the system that can cause components to fail, for instance, a bit-flip in a memory register causing an erroneous memory address. When triggered, a fault will manifest as an error in the system, essentially a behavioral discrepancy or invalid states, which can then propagate further, causing more errors. Continuing the example, the fault results in an invalid address in a pointer variable. If the error reaches the system boundary, it causes a Failure, which is the inability to perform a function. In this case, a safety-relevant functionality crashes due to an invalid pointer.

A thorough survey of safety mechanisms which can be implemented in software was carried out in [130], based on recommendations from functional safety standards, such as the aforementioned ISO26262 [20] and NASA STD 8719.13 – Software Safety Standard [131], as well as Kelly’s work, but also on discussions with expert practitioners from the automotive industry. Building on the abstract categorization of safety “tactics” for failure management provided by Kelly and Wu in [44], the authors expanded the categories to include numerous patterns. These are described abstractly and divided into three groups based on separation of concerns, as seen in Figure 8 above. We will be basing our target pattern group on this collection and categorization.

The top left corner of Figure 8 comprises avoidance mechanisms. Since errors are the manifestation of faults, preventing them from occurring is only possible by avoiding their triggering faults. The center group targets the detection of any errors triggered by faults that could not be avoided. This group comprises two subcategories. Mechanisms which rely on previous system information for their functionality are called “stateful” error detection, e.g., control flow monitoring. Safety mechanisms which only need current information are fittingly labeled “stateless”. The group on the right comprises mechanisms to manage the errors – once they have been detected – and prevent them from causing failures.

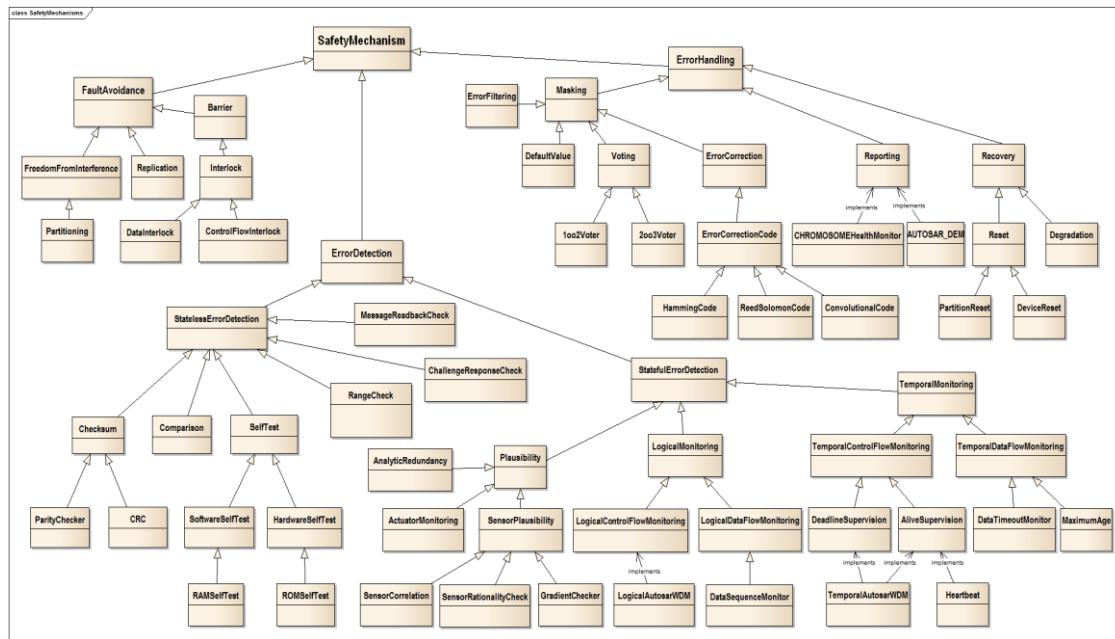


Figure 8: Safety Mechanisms' Categorization [132]

To understand the context of the Design Pattern documentation and approach presented in this thesis, some existing pattern documentation forms are presented and compared in the following section.

3.4 Design Pattern Templates

Alexander defined a basic construct for structuring pattern documentation, based on the triumvirate of *Problem*, *Context*, and *Solution*. The software community adopted this structure, and adapted it where necessary, by breaking down or decomposing these three essential pattern characteristics into more concrete ones. This was done on a case-by-case basis, to more accurately capture the significant aspects of the Design Patterns they were trying to describe. For instance, the *Context* could be broken down into separate entries for what elements play a part in the pattern, that is, its *Participants*, as well as entries for *Structure* and *Applicability*. Similarly, *Solution* descriptions could be decomposed into descriptions of the *implementation*, as well as how its *instantiation* works, supported by *examples*. The essential three building blocks – *problem*, *context*, and *solution* – can always be identified at the heart of all the various Design Pattern Templates.

The attributes are held together by the pattern's *name*, which should be unique and meaningful, ideally conveying the purpose and or intent of the pattern. The documentation includes a *structure* – that is, the *attributes* which capture each Design Pattern and their relation to one another, if any – and a *representation*, which is how the *attributes* are documented. The structure and representation of patterns is explored in the following subsections.

3.4.1 Design Pattern Structure

A quick survey of the first edition of “Pattern Languages of Program Design” – the Book collecting the accepted papers submitted to the PLoP Conference, introduced in the previous section – reveals how the software community adapted the newly established paradigm of Design Patterns. Wolf and Liu, who documented patterns for server/client frameworks [133], did so using *Problem* and *Solution* descriptions, with context information divided among the two attributes. Riehle and Zühlke analyzed functionality and usage patterns of software tools to facilitate tool construction and integration [101]. They added an explicit *Purpose* attribute, as well as *Compare*, which they used to capture interfaces and responsibilities. Rubel’s work on patterns for layered architecture [134], aimed at supporting a natural decomposition of system requirements into a layered architecture. He used mechanical control systems as an example, so he added the attribute *Forces* to the Alexandrian *Problem*, *Context* and *Solution*. He also explicitly added *Design Rationale* and a *Resulting Context* attribute to capture the impact of applying the pattern, as well as an attribute to document *Related Patterns*. Adams, on the other hand, worked on functional programming, and his function patterns [135] reflected this, replacing *Context* with explicit attributes for *Preconditions*, *History* and *Constraints*, and adding *Aliases*. Working on patterns in object-oriented database frameworks [136], Lajoie and Keller did away with both the *Context* and *Solution* attributes, in favor of a more detailed documentation in *Intent*, *Applicability*, *Description*, *Diagram*, *Motivation*, *Implementation*, *Category* and *Contract Example*, supported by *Discussion* and *See Also*. The Gang of Four (Gamma, Helm, Johnson and Vlissides) targeted general object-oriented concepts and paradigms, and subsequently sought to capture as many attributes as possible, in a comprehensive and explicit manner. Their Template [108] included 13 attributes: *Name*, *Intent*, *Also Known As*, *Motivation*, *Applicability*, *Structure*, *Participants*, *Collaborations*, *Consequences*, *Implementation*, *Sample Code*, *Known Uses* and *Related Pattern*.

Kelly, who included the tabular results of a similar survey in his doctoral thesis [123], aimed for a general methodology to capture safety case patterns. Motivated by their comprehensiveness and “freedom from explicitly software – especially (object-oriented) – attributes”, as well as the “explicit representation of *Applicability*”, Kelly adopted the pattern attributes suggested by the Gang of Four (GoF) nearly wholesale, replacing only *Sample Code*, with *Example Applications*. In his later collaboration with Wu on “safety tactics” [44], the attribute Template was whittled down considerably. Their Template [44] kept the attributes *Aim* (same as *Intent*), *Description*, *Rationale*, *Applicability*, *Consequences* and *Related Tactics*. It added *Side Effects* (to capture the impact on other quality attributes), as well as *Practical Strategies* and *Patterns* to list actual implementations.

Table 4: Comparison of Design Pattern Templates

Alexander [97]	Wolf and Lui [133]	Riehle and Züllighoven [101]	Adams [135]	Rubel [134]	Lajoie and Keler [136]	Gang of Four [108]	Kelly [123]	Wu and Kelly [44]	Armouath [122]
Name	Name	Name	Name	Name	Name	Name	Name	Name	Name
			Aliases			Also Known As	Also Known As		Other Names
Problem	Problem	Problem	Problems	Problem	Description			Description	Problem
		Purpose			Rationale / Intent	Intent	Intent	Aim	Abstract
					Motivating	Motivation	Motivation	Rationale	
Context		Context	History	Context					Context
			Preconditions	Forces	Applicability	Applicability	Applicability	Applicability	
			Constraints	Resulting	Diagram	Participants	Participants		
				Design	Discussion	Collaborations	Collaborations		
						Consequences	Consequences	Consequences	Consequences
								Side Effects	Implications
Solution	Solution	Solution	Solution	Solution	Implementation	Implementation	Implementation	Practical	Implementation
						Structure	Structure		Structure
					Contract Example	Sample Code	Example	Patterns	
						Known Uses	Known Uses		
					Category				Type
		Compare		Related	See Also	Related Patterns	Related Patterns	Related Tactics	

Armoush [122] cataloged various types of redundancy patterns (some of his own design) for reuse in safety-critical applications. Basing his Template on a mixture of earlier works and that of Kelly [123], he documented *Name*, *Other names* (same as *Also Known As*), *Context*, *Problem*, *Structure*, *Implementation*, *Consequences*, and *Related Patterns*.

Armoush also added *Type* (to differentiate between hardware-based, software-based, and mixed redundancy patterns), *Abstract* (which is essentially a combination of *Solution* and *Rationale*), as well as an attribute he named *Implications*, which aims to explicitly capture the “impact of a Design Pattern on the non-functional requirement (safety, reliability, modifiability, cost, and execution time)”. This is essentially analogous to the attribute *Side Effects*, introduced by Wu and Kelly in their “Safety tactics” work [44], although Armoush sought to concretize it, by expressing improvements to system reliability and safety, as well as cost, modifiability and impact on execution time. Analyzing what is arguably a closely-knit family of patterns, he chose to omit the *related patterns* attribute altogether.

Table 4 gives a comparative overview of the structure of the aforementioned pattern Templates. There are many other examples, yet the salient point is that regardless of the targeted Design Pattern, the structure of the pattern Template revolves around the Alexandrian three-points (Problem-Context-Solution), expanded and refined or consolidated to suit the domain and needs of the user.

Having decided which attributes they were going to include in their pattern Templates, hence capturing the *structure* of the Template, most pattern collectors stuck to Alexander’s form of documenting the attributes, i.e., the *representation*, as seen in the following subsection.

3.4.2 Design Pattern Representation

Figure 9 shows a pattern for rural planning in the vicinity of city limits, from Alexander’s book “A Pattern Language: Towns, Buildings, Construction” [137].

In the zone where city and country meet, place country roads at least one mile apart, so that they enclose squares of countryside and farmland that are at least one square mile in area. Build homesteads along these roads, one lot deep, on lots of at least half an acre, with the square mile of countryside or farmland behind the houses.

Prose

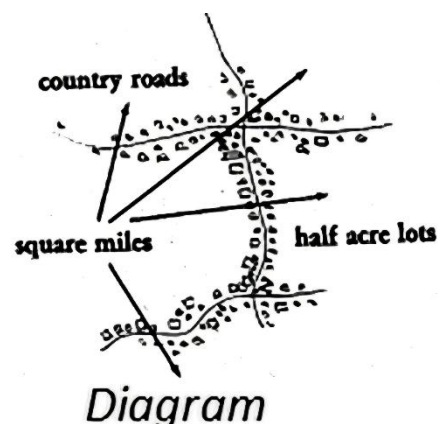
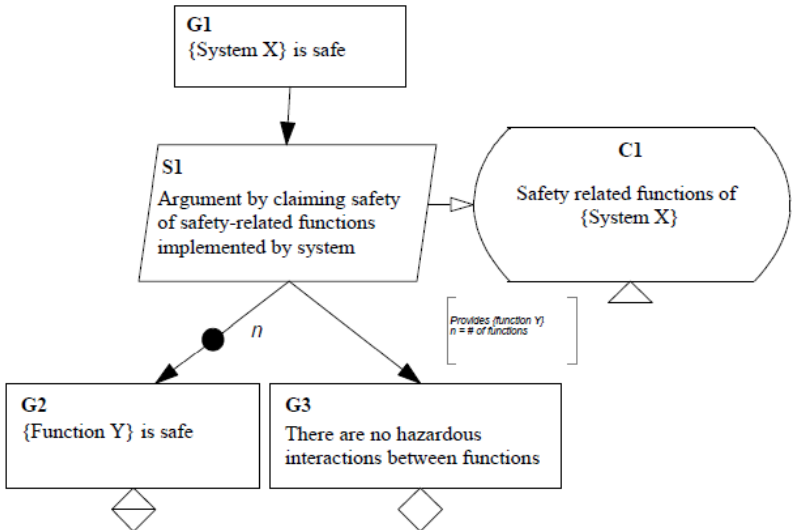


Figure 9: Alexander’s pattern for planning rural community plots [137]

As Kelly observed [123], Alexander’s architectural patterns and their various adoptions into software Design Patterns share a common representation; namely that the attributes are represented as *structured prose*, supported by *diagrams*, as seen in Figure 9. This documentation scheme did not change; only the kinds of attributes (as discussed in the previous section), and the type of diagram used to support them, varied with the domain and purpose of the pattern Template. Thus, the “Gang of Four” pattern Template for object-oriented programming [108] captured attributes textually in tabular form; that is, in *structured prose*, supported by *diagrams* in three different notations. These were, in turn, inspired by object-oriented paradigms; Class Diagrams and Object Diagrams, both based on the Object Modeling Technique (OMT) [138], and Interaction Diagrams – based on the Booch Method [139] as well as Objectory [140] – were used, with each pattern containing at least a Class Diagram.

Kelly himself followed suite, structuring the various attributes he used to document his safety case patterns in tabular form, with the attribute information captured textually. This is supported by graphical safety case notations using GSN, essentially a *diagram*. The next table shows a pattern documentation from Kelly’s doctoral thesis [123].

Table 5: Functional Decomposition Safety Case Pattern from Kelly’s Doctoral Work [123]

Functional Decomposition Pattern			
Author	Tim Kelly		
Created	22/02/99 01:56	Last Modified	22/02/99 02:36
Intent	The intent of this pattern is to argue the safety of a system by appeal (sic) to the safety of the functions implemented by that system.		
Also Known As	Functional Safety “Divide and Conquer” Pattern		
Motivation	The motivation for this pattern is the need to decompose a high-level goal (that is difficult to substantiate ‘as-is’) into sub-goals that are hopefully easier to address.		
Structure	 <pre> graph TD G1["G1 {System X} is safe"] --> S1["S1 Argument by claiming safety of safety-related functions implemented by system"] S1 --> C1["C1 Safety related functions of {System X}"] S1 -- "n" --> G2["G2 {Function Y} is safe"] S1 --> G3["G3 There are no hazardous interactions between functions"] G2 --- D1{ } G3 --- D2{ } C1 --- D3{ } D1 --- P["Provides (function Y) n = # of functions"] D2 --- P D3 --- P </pre> The diagram illustrates the Functional Decomposition Safety Case Pattern. It starts with a goal G1: "{System X} is safe". This goal is supported by a strategy S1: "Argument by claiming safety of safety-related functions implemented by system". S1 is supported by a context C1: "Safety related functions of {System X}". S1 is also supported by two sub-goals: G2: "{Function Y} is safe" and G3: "There are no hazardous interactions between functions". G2 and G3 are supported by a context D1: "Provides (function Y) n = # of functions".		

Participants	G1 Defines the overall objective of the pattern S1 Presents the strategy (functional decomposition) adopted to support G1 C1 A list of functions performed by System X that could impact system safety required to expand S1 G2 Expresses safety claim for each identified safety-related function (over items in C1) G3 Goal required to validate the approach adopted
Collaborations	• C1 introduces the safety related functions of System X that then form the basis for constructing G2 goals. • Goal G3 is necessary to support the implication that solving the goals G2 is sufficient to support G1.
Applicability	This is a very general pattern and, as such, has a wide applicability. In order to apply the pattern, it is necessary to instantiate C1 (Safety Related Functions of System X). C1 should identify the list of all functions of the system that have been identified as having a possible impact on system safety. Functional Hazard Analysis is a possible approach to identifying safety-related functions from the list of all system functions.
Consequences	After instantiating this pattern, a number ($n+1$, where $n = \#$ of functions) of unresolved goals will remain: • G2 (n of) • For each function it is necessary to support this claim that the function is implemented safely, and appropriate measures have been taken to mitigate or eliminate risks associated with the function. • G3 • To support the functional decomposition of the overall goal of safety into sub-goals of safety over each function it is necessary to support this claim of independence - i.e. there are no hazards generated by the interaction between functions.
Implementation	In implementing this pattern, it is first necessary to instantiate C1 (identify the list of system functions) Possible Pitfalls: • Attempting to decompose G1 in sub-goals over functions (G2) without adequately supporting the claim of independence between functions (G3) • Inappropriately instantiating C1 with simply safety functions rather than safety-related functions. The distinction being that safety functions are those functions that are obviously concerned with achieving safety (e.g. protection

	mechanisms) and safety-related functions are any functions in the system that have been identified, e.g. through Functional Hazard Analysis, as potentially posing a safety hazard. Safety functions are a sub-set of safety-related functions.
Example	This goal structure shows the instantiation of the pattern for an aero-engine controller. The top goal (G1) has been instantiated to refer to the system in question. S1 (the argument strategy) remains unchanged. C1 is instantiated to refer to a Functional Requirements Document (FRD) that clearly identifies the main functions of the engine-controller. These functions have then been used as the basis for putting forward the claims G2, G3 and G4, each expressing a goal of safety for a separate functional area. (There are more functional areas than those included in this example). Beneath this argument, it is then necessary (although not shown here) to support each of these functional safety claims together with the independence claim – G5.
Known Uses	Engine Controller for the PT390 Engine (Ref SJ/3.2/97)
Related Patterns	Hazard Directed Argument Pattern: a pattern that can be applied at a similar level in an overall safety argument, but which breaks down an overall system safety goal by introducing (and claiming safety against) the list of system hazards.

More than a decade later, this method of documenting patterns still holds. For instance, Armoush’s 2010 work on redundancy patterns [122] also captured attributes using structured text, supported by schematics or *diagrams* to capture the pattern structure. An example for such a structure diagram for the “Triple Modular Redundancy” pattern is shown in Figure 10.

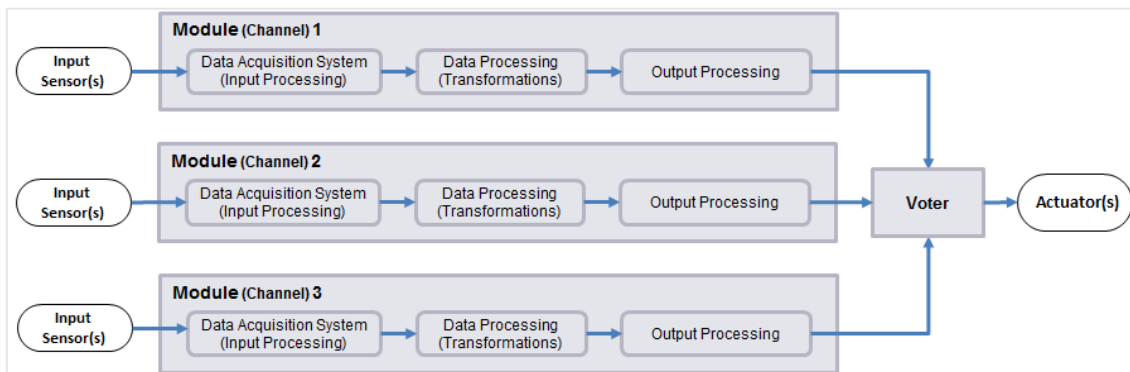


Figure 10: Structure Diagram of Triple Modular Redundancy Pattern used by Armoush [122]

3.5 Summary

When cataloging specific pattern classes, the authors tend to hand-pick design pattern attributes based on relevance, but also arguably their ease of documentation. More importantly, pattern authors are mostly more interested in identifying purportedly new design patterns, and describing them as generically as possible – independently of implementation or technology – so they can be added to the general wealth of Engineering, Technology, and Computer Science knowledge. In contrast, we are more interested in the practical aspects of capturing and reusing system solutions in practice.

This chapter gave an overview on the related work in design patterns for safety and introduced the design pattern category – Safety Mechanisms – we will be using as an example to scope our approach.

The next chapter will introduce our approach, contrast to related work and define the core elements, concepts and constructs necessary for our solution.

4 MODPAL – The Model-based Design Pattern Library Approach

The development of safety-critical systems is largely driven by trusted designs, proven solutions and best practices, with reuse playing an increasingly significant role in the development of complex systems. The previous chapters laid the foundational concepts and bounding constraints needed for mature reuse of system solutions in practice.

This chapter investigates the four challenges presented in the introduction chapter and proposes a solution to address them – in the form of a model-based approach for reuse in practice, with step-by-step guidance for its instantiation into the practitioner development environment, and a simple evaluation scheme to measure reuse maturity and guide technical capability enhancements. The approach is then validated by multiple case-studies demonstrating its feasibility and claimed benefits in the following chapters of this thesis.

We will begin by briefly presenting our research methodology and then recap our problem statement and research challenges, before contrasting related work, explaining why the topic remains unresolved.

4.1 Research methodology

Chapters 4 through 6 constitute the *Prescriptive Study (PS)* phase of the Design Research Methodology (DRM), according to Blessing and Chakrabarti, which we are following in this thesis. In this phase, a solution is developed based on insights from the Descriptive Study I (DS-I), presented in the previous 2 chapters. Chapter 4 presents the design of a novel reuse framework for safety-critical systems, including a formalized safety mechanism pattern template, artifact structure, and instantiation guidelines. Finally, an evaluation scheme to measure the impact of the approach is defined.

Chapters 5 and 6 provide an initial evaluation of this support in academic and controlled pre-industrialization environments, demonstrating its fundamental feasibility and functionality. These evaluations – one using a generalized CASE tool extended with reuse-focused features, and another within a domain-specific automotive modeling framework – serve to iteratively refine the solution and validate its technical applicability prior to industrial deployment.

While an initial formalization of the proposed reuse framework using a set-theoretic representation is developed in this chapter (section 4.7), the formalization was not pursued further in the evaluation phases. The formal models served primarily as conceptual guidance for how to instantiate the approach in two distinct modeling environments – the research-oriented MBSE/CASE tool (AF3) in chapter 5 and a domain-

specific automotive safety modeling framework (SAFE-MF) in chapter 6 – but were not central beyond this aspect to the empirical investigations. This decision was based on two considerations. First, the utility of a full formal proof of applicability within these prototypical environments was judged to be limited in terms of external validity and practical impact, particularly since the frameworks themselves were still under academic or pre-commercial development. Second, applying a formalized evaluation in the industrial case study – intended to be conducted over an extended time period and hence involving evolving tooling infrastructure and organizational processes – would have been impractical, if not infeasible, due to the dynamic, multi-stakeholder, and large-scale nature of the real-world setting.

Our approach aligns with established guidance in design research methodology, which emphasizes the importance of contextual relevance, utility, and rigorous evaluation of artifacts through iterative application in real-world environments. In particular, Hevner et al. [141] argue that evaluation in *context*, i.e., using empirical methods such as case studies, field studies, and expert assessment, is not only valid but essential for demonstrating the practical value of a designed artifact in complex domains like engineering or information systems, while Iivari [142] argues that design research must balance formal rigor with practical utility.

Similarly, the DRM methodology we are following in this thesis supports a pragmatic approach in which different types of rigor (formal, empirical, conceptual) are appropriate at different phases of the research process [24]. As such, we believe that the decision to prioritize empirical and longitudinal evaluation over continued formalization reflects a methodologically sound, context-aware research strategy aimed at maximizing practical relevance without sacrificing academic integrity.

4.2 Our Solution in Context

A short personal note: when the search for practical solutions to my “reuse in practice” problem began (over 15 years ago, while still working in industry), I was struck by a recurring theme in academic work: many proposed approaches were elegant in principle but took place in highly sanitized, idealized settings; what Admiral Rickover famously derided as “*paper reactors*” in his 1953²² memo. These solutions had limited scalability or applicability in real engineering environments.

A deeper issue soon became apparent: most approaches were tightly bound to a specific modeling languages or development framework. Their “complete” solutions were constrained by what their chosen environments *could express* – and therefore, what they

²² This link (<https://whatisnuclear.com/rickover.html>) provides a reproduction of Admiral Rickover’s famous ‘paper reactors’ memo, from June 5th, 1953. The original was published in The Journal of Reactor Science and Technology, Volume 3, No. 3 (which is classified), but reproduced many times, e.g. in Congressional testimony.

could capture. This reflects a form of cognitive bias known as Maslow’s Hammer²³ [143]. In this context, both researchers and engineers frequently contort their tools to fit the problem, then view the results as satisfactory – even when the tool may be a poor match for the domain.

A clear example of this phenomenon is the proliferation of UML profiles (*Computer Science’s Hammer*) across literature, used to represent virtually every type of engineering activity; regardless of UML’s actual fitness for the task. Similarly, in industrial practice, I have observed the widespread (and problematic) misuse of rapid-control prototyping tools like Simulink in series software development for automotive systems. In such cases, the expressiveness of the tool effectively becomes the ceiling of what knowledge can be documented – and eventually, even perceived – by its users.

This thesis aims to break that cycle through a three-pronged approach:

- 1) **Tool-Agnostic Structuring:** We do not restrict ourselves to any single modeling language or development environment. Instead, we promote the use of **domain-adequate, context-rich models**, as these inherently support more complete and accurate knowledge capture.

- 2) **Problem-First Thinking:** We encourage practitioners to shift their mindset from “*What can I capture with my tools?*” to “***What must be captured to support correct reuse?***” The focus is on the actual problem space, not the limitations of any particular toolchain.

- 3) **Informed Tool Evaluation and Extension:** We equip practitioners to critically assess their tool environments: Are they capturing everything required for the quality they aim to deliver? If not, how can tools be extended or adapted? This enables *systematic, prioritized improvement* of reuse capabilities over time.

These three principles form the foundation of the approach presented in this thesis. The method is piloted in the scope of **Safety Mechanism Patterns (SMPs)**. Within that scope, we target the **encapsulation of reusable technical artifacts and supporting argumentation**, with the goal of enabling correct, efficient reuse.

Crucially, we define **what information must be captured** – rather than letting tool limitations define what can be captured. This tool-agnostic mindset allows us to define a pattern template in section 4.5 that is **unconstrained by any specific framework**, and which can subsequently be used to evaluate and improve modeling tools and environments. Our underlying hypothesis is that **the richer and more context-adequate a modeling framework is for the originally developed solution, the more comprehensively it can support its reuse**, and the higher the quality of the resulting product.

²³ Where every problem begins to look like a nail *because the practitioner only has a hammer*.

These principles are motivated by the four recurring challenges identified in the reuse of complex system solutions (previously introduced in Chapter 1):

- **Challenge 1:** The lack of a structured way to capture solution artifacts and their critical context,
- **Challenge 2:** The absence of maturity models or definitions of reuse completeness,
- **Challenge 3:** Difficulty integrating reuse methods into existing engineering toolchains, and
- **Challenge 4:** The lack of support for evaluating and extending tools to close reuse gaps.

Using the example of Safety Mechanisms, we will address Challenge 1 and Challenge 2 with the definition of a novel Design Pattern Template in section 4.5, and the subsequent definitions of the MODPAL approach in this chapter. Challenges 3 and 4 are addressed by definitions provided in this chapter, and validated using the use-cases in chapter 5 and chapter 6. Finally, chapter 7 demonstrates the ability to address all challenges for a generalized practical reuse problem in an industrial setting at scale.

4.3 Overview of Approach

We aim to bridge the gap between the systematic, well-structured (yet generic) representations used to capture patterns in design pattern literature, and the many needs of actually reusing the pattern in a practical setting, where domain models – often content-rich – provide an excellent opportunity to capture the pattern information in the same language and context it is going to be reused in. Most importantly, we wish to counteract the ad-hoc nature of current pattern reuse in practice, by using the structures known from design pattern literature and general rigor of defining a Design Pattern, as seen in the previous chapter, to derive a novel, systematic approach for pattern reuse in practice; the *Operationalization of Design Patterns*.

To do so, we approach reusable system designs (e.g., Safety Mechanisms) as if they were Design Patterns. This strategy enables more systematic, repeatable, and effective reuse within an organization. While Safety Mechanisms are not themselves design patterns, they serve as a well-scoped and complex enough class of solutions to demonstrate our method. We thus adopt the rigor of defining a Design Pattern Template to drive the systematization of reuse in practice.

However, merely defining yet another Design Pattern template does not solve the reuse problem. Our contribution lies in developing a holistic approach that leverages model-based capabilities to support tool-enabled reuse within a Systems Engineering context, aligned with ISO 15288 [29]. Given the software-intensive nature of modern embedded

systems, our method integrates established software engineering paradigms: Model-Driven Engineering (MDE) [144], Component-Based Software Engineering (CBSE) [145], and reuse repositories as presented in [146] and [147].

Our approach provides a library²⁴ of such solution patterns²⁵, which captures all the relevant pattern information, allowing the identification and comprehensive (re-)use of suitable architectural measures and designs, as well as their corresponding components and safety cases, facilitating a cumulative if not compositional argumentation about the achieved safety goals.

This chapter defines the structure of such a library and presents an approach, which facilitates the reuse of technical solutions, through the scoped example of Safety Mechanisms, by encapsulating all the information necessary for correct reuse in a pattern library with tool support. Our approach not only adds new attributes to the ones known in Design Pattern Template literature but captures the information from these attributes into the used models themselves – contrary to the generic representations found in Design Pattern literature. This decreases reliance on the hitherto typically verbose capturing of Design Pattern attributes and reduces ambiguity, avoids the common pitfalls of overlooked or misinterpreted textual descriptions, and grounds pattern reuse in practical development artifacts.

We call our solution the MOdel-based Design PATtern Library (MODPAL) approach. Throughout the remainder of this thesis, we will use the abbreviation MODPAL or the shorthand "Pattern Library" to refer to this solution.

Deeply rooted in the user-centric nature of Design Patterns, our solution can be summarized as:

- 1) A Design Pattern Template for Safety Mechanisms.
- 2) The Pattern Library Approach, composed of:
 - a. An *Artifact Reference Structure*, which defines the relevant set of concepts, attributes and relations to each other as well as to the development artifacts they are usually linked to in practice, as well as their representation forms and dependencies to be specified during instantiation into any development environment.

²⁴ A Library in Software Engineering is defined as a collection of non-volatile resources, such as pre-written code and subroutines, classes, configuration data, documentation, message templates, etc., used by computer programs. In the context of Object-Oriented Programming, a library is a collection of implementations of behavior, written in terms of a language, that has a well-defined interface by which the behavior is invoked.

²⁵ While our approach does build on the concepts of a software repository, we consider the term library to be more appropriate to our approach, as we focus more on defining the collection of Design Patterns and their *behavior and impact on the system*, and defining the interfaces and instantiation (i.e., invocation) mechanisms (*see definition above*), and providing these as a resource to practitioners, than on the Software Engineering mechanics of an actual software repository, which we leave to each practitioner to solve as they see fit.

Instantiating this structure within an existing development environment allows practitioners to capture solutions within their own tools, avoiding information loss in model-to-text transformations.

- b. *The Pattern Library Approach Guidelines*. Together with the Artifact Reference Structure, these offer the core modeling paradigms of MODPAL, including step-by-step guidance for the instantiation and tailoring of the approach into a development environment, as well as guides for both a *Pattern Developer* and *User* roles, instantiated into a domain-independent development environment in the first case study.

3) The *MODPAL instantiation Case Studies* (presented in Chapters 5 through 7), which validate the feasibility of the approach, support the evolution of its concepts and guidelines, and demonstrate practical instantiations.

This chapter introduces the MODPAL approach using simplified examples. We begin by defining a new Design Pattern Template for Safety Mechanisms. We then present the structural model of the Safety Mechanism Pattern, detailing new attributes and operations to illustrate its formalization potential and to define the basic Artifact Reference Structure. This is followed by a workflow for instantiating patterns into development environments and performing reuse evaluations. Finally, we introduce the evaluation scheme applied in the case studies in subsequent chapters.

Given the foundational role of this chapter in establishing the technical basis of our work, the Related Work section will follow, contrasting our approach with other efforts and situating it in the broader academic landscape, before introducing the new Safety Mechanism Pattern Template in section 4.5.

4.4 Related Work

This section is structured as follows:

We first distinguish our focus from previously discussed related work in Design Patterns, particularly in terms of *how* we propose to capture and reuse design solutions – directly in engineering artifacts, rather than structured prose and diagrams.

We then review prior work focused on **model-based collection and reuse of safety mechanisms** or similarly scoped design solutions.

Finally, we examine approaches that are **comparable in breadth and ambition**, integrating design patterns into MBSE frameworks for knowledge retention and systematic reuse.

4.4.1 Patterns in Safety, and the Case for a Different Pattern Representation

As discussed in detail in section 3.2, the concept of Design Patterns originated in architecture, with Alexander's pattern language [137] first articulating them as a way to capture proven solutions in context, before they were popularized in Software Engineering by the "Gang of Four" in the 1990s [108]. It was less than a decade before it had been adapted by Kelly to the topic of Safety Cases. Although providing multiple useful generic argument strategies, Kelly's catalog [123] merely represents a cross-section of useful solutions for unconnected parts of arguments. Weaver specifically developed a safety pattern Template for software [124], with the crucial difference being that this Template not only specifically targeted software aspects of the system under development, but the set of included patterns were also specifically designed to connect together in order to form a coherent argument. The solution was, however, not generic and a number of weaknesses have since been identified with Weaver's pattern Template, as mentioned in [148]. First, the patterns take a fairly narrow view of assuring software safety, focusing on the mitigation of known failure modes in the design. Mitigation of failure modes is important, but there are other aspects of software assurance which should be given similar prominence. Second, issues such as safety requirement traceability and mitigation were considered at a single point in Weaver's patterns. This is not a good approach; as it does not reflect the building up of assurance relating to traceability and mitigation over the decomposition of the software design. Finally, Weaver's patterns have a rigid structure that leaves little scope for any alternative strategies that might be needed for novel technologies or design techniques.

Ye's patterns for COTS components in safety provide some interesting advances on Weaver's, including the introduction of patterns for arguing that the evidence is adequate for the assurance level of the claim it is supporting [148], a general weakness in the safety case pattern regime. Ye's work notwithstanding – or that of others who have addressed the problem, like Rushby [149] – the problem remains unsolved, with Habli and Kelly stating: *"Although representing an argument using a notation such as GSN clearly disambiguates the structure and elements of the argument, it **cannot ensure** that the argument itself is 'good' or sufficient for its purpose."* in a 2014 position paper advocating the formalization of GSN [150].

Most importantly, these approaches all focus on documenting safety case templates for known patterns, whereas the safety case templates themselves are the target. While still quite useful as a prescriptive aide to guide developers to what the solution argument structure should look like, safety case templates lack (by definition) the link to the actual artifacts supporting the argumentation elements.

Vast lists of safety patterns were compiled, for example, by Douglass in [115] and [116], as well as Pullum [151], Koren and Krishna [152], and Hanmer [153]. These Templates focused on general descriptions of the patterns. Armoush's work [122], while capturing

many practical aspects of redundancy patterns, does not offer any support for the practical application of the patterns in system architecture models in practice.

Similarly, the many Templates that have been presented in Chapter 3 have a common purpose; namely, the generic documentation of patterns relevant for a domain or contextual use. The authors analyzed their domains, identified relevant trends and themes, and sought to document their findings as pattern Templates, presenting an important addition to the wealth of engineering and computer science knowledge. The pattern representations as *structured text* supported by *diagrams* – as introduced in section 3.4 – are well suited for this purpose, because they are independent of technology or implementation. This is useful for an academic or research context, where the purpose is to define a new pattern and explain its usefulness in a generic manner, understandable to all.

But what about the actual immediate use of patterns or reuse of systems solutions in a practical environment, i.e., within an engineering organization?

This is the focus of this thesis, and the reason that we intend to capture the needed information (defined in the attributes) of system solutions (initially scoped by the example of safety mechanisms) using the actual development artifacts, and the same tools, which will be used to develop the target system (and reuse it) in practice.

4.4.2 Related work combining Patterns and Model-driven approaches for Reuse

With the advent of powerful modeling languages like UML in the late 90's and SysML in 2006 (see section 2.5.2.3. for more details), it was only a matter of time until Design Patterns were combined with the power of model-based techniques. In fact, Douglass himself documented his first patterns for real-time embedded systems using UML [115]. The following decade saw an explosion of capturing patterns using models. The examples are abundant: using UML to capture hardware design process patterns [113] or to document patterns for software product lines [154]; or using models to capture hardware design patterns [114], yet the fact remains, that the focus was on leveraging the capabilities of model-based techniques for the documentation of the patterns, rather than capturing them for reuse in a practical setting.

An interesting solution more in this direction was presented by the OPENCROSS Project in form of the Assured Reliability and Resilience Level (ARRL) approach [155]. In essence, ARRL leverages Quality of Service (QoS) paradigms, arguing against Safety Integrity Levels (SIL) and seeking to supplant them with ARRLs, which essentially transform components into a compositional framework, where “ARRL level components carry a contract and the evidence that they will meet this contract given a specific set of fault conditions”. This should allow for a safer reuse of components.

While the notion has its charm, there are several limitations to this approach:

- It assumes that the information is available in the models or components it is applying ARRL to; this is often not the case.
- It assumes that a transformation of the information, contracts and guarantees is possible into ARRL, without loss of information. In practical settings, this is not a trivial assumption.
- It requires the adoption of a new framework, which will undoubtedly meet very high resistance from industry, as well as substantial and not entirely drawback-free changes to existing and established safety standards and practices.

On a more pragmatic note, Denney and Pai [156] offer automation support for the generation of (partial) safety cases, by considering three inter-related tables – namely one each for hazards, system requirements, and functional requirements – as idealizations of the safety analysis and development processes. The argument structures are then automatically populated by predefined mappings from the tables. While the approach is useful for structuring requirements and argumentation, is for the most part formalized, and has functioning tool support [157], it does not directly link design architectures or product artifacts in its function; or address patterns; or how the aspects needed for reuse are covered; nor does it show how the method may work in other fields.

Armengaud [158] proposed a conceptually similar approach for automating the assembly of safety cases. His approach is based on linking ISO 26262 activities and work products – for instance, evidence such as design and test artifacts – by means of information models. However, the author does not address how these models structure the safety case argument; nor does he address the reuse of patterns or safety mechanisms in safety.

A noteworthy contribution is presented in [159], where the authors developed a UML profile, supported by tabular “role descriptions”, which facilitate the description of and practical reuse of safety patterns. The approach is however, not without its shortcomings. It does not show how all the information of a pattern is captured in the model and does not support implementation. It also does not support argumentation or safety case documentation. In fact, the approach only targets architectural reuse, yet as the authors put it “the rules described in this work are not enough to ensure the application of a safety pattern in an architectural model”.

More promising (and pragmatic) work in this direction was that of the HISE Group of York University, presented by Graydon et al. in [160]. In it the authors present an approach based on the automatic generation of assurance cases from design artifacts using model weaving. In essence, the authors defined model transformations to weave various elements relevant for arguing the safety of architecture design and product information from the AADL²⁶, with a meta-model of GSN, to generate a safety case argument

²⁶ The Architecture Analysis and Design Language (AADL), is an ADL standardized by the SAE (Society of Automotive Engineers) in 2001. The AADL was developed for avionics and initially known as the Avionics Architecture Description Language. It continues to be developed for avionics applications but has not really found much support in the automotive domain, where complementary European-driven concepts like the EAST-ADL and AUTOSAR have received higher acceptance and practical adoption.

structure. One of the main advantages of the approach is that the weaving models are bidirectional; allowing the generation of updated argument structures to match changes in product information, but also allowing the reflection of changes in the argument structure back to the product and architecture design. This facilitates the co-evolution of architecture design and product information with the assurance case.

The approach is not without limitations. It is restricted to development environments using (or based on) AADL, which is not widely adopted in the automotive domain; and it does not address implementation artifacts or issues. Furthermore, the approach does not address patterns but is focused on the generation of assurance cases. The approach can, however, be used to support safety patterns, by adapting it to maintain the co-evolution of the system design and product information side with the argumentation side of a safety pattern library element. In that capacity it is most useful when applied to abstract high-level patterns. For concrete Safety Mechanisms, such as those targeted in this thesis, it would be much simpler and more precise to directly link the assurance case elements to the design and implementation artifacts supporting them. In that scope, the York Model-weaving approach would be too general in its current form, and on the other hand, attempts to make it more precise would probably be over-kill and counter-productive. Its strengths simply lie elsewhere.

More closely related are the works of Hauge [161] and Hamid [147]. Hauge introduces a pattern language for safety design – called Safe Control Systems (SaCS) - with which he proceeds to capture known approaches towards a safe system design. These include both product-based as well as process-based safety concepts enabling a safe-design. Hauge's selection of what a "safety concept design" is builds on the work of Habli and Kelly in [129], in which they introduce safety case patterns for generic safety concepts, i.e., repeatable argument patterns, using GSN. The SaCS language provides good coverage of safety concept design, as demonstrated in the analysis given in [161], and expanded on in [162]. Hamid introduces a holistic approach for model repository-centric reuse of solutions for complex applications. His approach provides a methodology and tool-support for developing model repositories, supporting two categories of users: "reuse" producers and "reuse" consumers, and is demonstrated via a preliminary prototype that captures security and dependability pattern models. Hamid builds on his previous work in [163], in which he introduces an approach he refers to as "SEMCO (System and software Engineering with Multi-Concerns support)", which is an integrated repository of modeling artifacts aiming at enabling the co-evolution of models covering various engineering concerns, e.g., safety, security.

Hauge's work targets basic "patterns" in safety concepts, such as "Hazard Analysis", "Risk Analysis", and "Establish System Safety Requirements" and how to combine them into composite "Safety Requirements" patterns. Our work focuses on the practical reuse of the category of technical solutions collected under the term "safety mechanisms", as well as the generalization of the concept for other types of model-based reuse. While Hamid's work offers a holistic and precise approach to building modeling repositories.

Its reliance on model transformations, while supporting a more precise definition of his holistic model repositories, can be somewhat cumbersome and inflexible to a practitioner, and this reason made us opt for the relative flexibility of model-weaving. More importantly, both Hauge and Hamid have come up with their own modeling languages and tools. Similar to the ARRL and related approaches presented here, this assumes that practitioners are willing, or even capable, of completely abandoning their domain/industry-established tools in favor of an academic suite with little to no commercial support. In contrast our approach provides a blue-print for practitioners to tailor and instantiate it into their own tool suites for their own exact purposes, while enabling them to evaluate the adequacy of those tools for their individual purposes and providing a guide for eventual tool extensions/replacements.

Equally interesting for the broader perspective of combining multiple research threads for reuse in a safety context is the work of Barbara Gallina. Her 2015 publication, “*Towards Enabling Reuse in the Context of Safety-Critical Product Lines*” [164], introduces the "Anti-Sisyphus" framework, which aims to facilitate reuse across process lines, product lines, and safety case lines in safety-critical systems engineering. This approach seeks to reduce the time and cost associated with creating safety cases by systematically identifying and managing commonalities and variabilities in processes and products, thereby enabling the reuse of safety arguments and evidence across different product variants. The approach leverages modeling techniques such as feature diagrams for product lines, SPEM 2.0 for process lines, and Goal Structuring Notation (GSN) for safety case lines to enable semi-automatic generation of safety cases tailored to specific product and process configurations.

While both Gallina's work and this thesis emphasize the importance of reuse in safety-critical systems, they differ in focus and application. Gallina's framework is primarily conceptual, addressing the reuse of safety artifacts at a high level without delving into the operationalization within specific MBSE tools or industrial contexts. In contrast, this thesis presents a practical methodology for embedding reusable design patterns directly into MBSE environments, validated through longitudinal industrial studies. It emphasizes the direct reuse of complex system solutions within an organization, focusing on tool integration and organizational adoption, areas not extensively covered in Gallina's publications.

Indeed, a 2015 initial survey of patterns in safety-critical development by Gleirscher and Kugele [165] demonstrates that most research work on patterns tended to focus on clear target types of problems; engineering step (requirements, design, implementation...), artifact category (argumentation pattern, safety / security pattern, design pattern – like redundancy...), as well as domain abstraction (e.g. software, hardware...), and used these to organize the results.

The authors expanded on this work substantially in 2019 [17], presenting a comprehensive survey of design and argument patterns used in the assurance of safety-

critical systems. Their work aims to systematize the body of knowledge around reusable safety patterns, particularly in *specification*, *design*, and *argumentation* (similar to our earlier work), to identify gaps and future directions in safety engineering research.

They conclude that while safety patterns are widely employed across the development of safety-critical systems, their **use remains largely inconsistent**, particularly in terms of formalization, structure, and evaluation. Decomposition and behavior patterns emerge as the most frequently applied types, especially within software-intensive systems, typically supporting design strategies such as redundancy, separation, and monitoring. Argument patterns—though increasingly researched—are still not well integrated with the corresponding design patterns, with most focusing on compliance, confidence building, and process-based assurance. The authors note that many patterns are domain-specific, and cross-domain reuse is rare, limiting broader applicability. Furthermore, the systematic connection between safety and security remains underexplored. Importantly, the survey identifies a lack of rigorous evidence demonstrating the practical effectiveness of safety patterns in strengthening safety and its assurance. To address these limitations, Gleirscher and Kugele propose a taxonomy linking safety tactics to pattern categories and call for future work to develop standardized, tool-supported libraries of reusable patterns that can span disciplinary and domain boundaries [17].

We agree, a holistic *Systems Engineering* approach is needed. The following section investigates related works and the intersection of Design patterns and Model-Based Systems Engineering (MBSE).

4.4.3 Related work integrating Patterns with MBSE frameworks

In *Systems Engineering*, early advocates like Cloutier and Verma explored how patterns could benefit system architecture. They published the results of a survey in 2007 [166], proposing a *Systems Pattern* template, and observing that “*Systems Engineering patterns operate at a higher level of abstraction than Alexander’s original building patterns*”.

Particularly notable in Cloutier’s body of work is his investigation into applying architectural patterns and model-based system engineering (MBSE) principles to manage complexity in systems engineering projects, providing foundational insights into architecting approaches suitable for complex, integrated environments. Like the work presented in this thesis, Cloutier recognizes the value of patterns as reusable knowledge structures to address complexity and enhance reliability in systems engineering [167]. However, while Cloutier emphasizes patterns primarily in the context of capturing and maintaining know-how from an architectural perspective to aid in systems design and complexity management, this thesis specifically operationalizes patterns within a model-based framework tailored explicitly toward systematic reuse of safety-critical system solutions. The distinctive contribution of this thesis lies in its explicit formalization of pattern templates and artifact structures to enable direct and practical reuse within an organization – an area not covered in Cloutier’s published research.

Moreover, Cloutier’s work on the applicability of patterns tends to address broad-scale, domain-independent complexity in system architectures and emphasizes theoretical exploration and conceptual applicability, as seen in [167] and [166], as well as the role of patterns as Know-How carriers combined with MBSE to enhance its adoption among practitioners; with Cloutier pointing out that, the same way engineers reuse their knowledge based on previous experience, Alexander and his co-authors "*did not invent these patterns, they came from observation and testing; and only then were they documented as patterns*" [167].

Conversely, the contribution of this thesis is more practically embedded: it explicitly addresses domain-specific constraints (i.e., automotive safety-critical systems) and emphasizes practical instantiation and multi-year industrial validation. By operationalizing pattern reuse through MBSE and evaluating it systematically through expert surveys and longitudinal industrial studies, the current research provides tangible support for engineers working under stringent safety and regulatory requirements; thereby extending beyond Cloutier’s general conceptual treatment of patterns.

This is not to say that Cloutier and Verma’s research did not influence or benefit this thesis. Their work with Muller on introducing the concept of *Reference Architectures* [168], combined with that of Bass et al. [169], and Nagakawa et al. [170], was foundational for the approach used to instantiate our method in the industrial use case. More details on this aspect are provided in section 7.2.2.

Similarly to Cloutier and Verma, Haskins argued that writing and sharing patterns could expand Systems Engineering to address not just technical but also social aspects of systems [171]. These foundational works established that capturing architectural and process “know-how” in pattern form can foster knowledge reuse across projects; elevating the level of abstraction of Design Pattern capturing. Over the past decade, the INCOSE community has formalized this idea as *Pattern-Based Systems Engineering* (PBSE) – an approach to leverage MBSE models for reuse by treating proven sub-systems and practices as configurable patterns. Schindel and Peterson introduced PBSE at an INCOSE symposium, showing how MBSE techniques could be “leveraged” by patterns to streamline system family development [172]. In parallel, researchers sought to formalize Systems Engineering patterns. For example, Pfister et al. proposed a *Design-Pattern Metamodel for Systems Engineering*, enabling pattern specification in a modeling tool [173]. Likewise, Ernadote applied ontologies to define [174] and reuse Systems Engineering patterns, demonstrating pattern-based approaches for system design with formal semantic grounding [175]. These efforts all aimed to enhance MBSE by embedding expert knowledge into MBSE artifacts so engineers can reuse validated solutions instead of reinventing them, thus improving consistency and reducing design effort. In summary, the community consensus, as projected early by Cloutier and Verma [166], is that patterns can complement other reuse tactics by codifying an organization’s *tacit know-how* (design heuristics, best practices, preferred solutions) in a transferable form.

On a more general note, Husung’s research, particularly his 2016 introduction of “solution patterns” within the context of MBSE [176], and its 2023 follow-up focusing on their systematic application in the example of a load cell [177], aligns conceptually with our core idea of capturing reusable engineering knowledge to improve development efficiency. However, while both efforts emphasize repeatability and traceable model-driven engineering, Husung’s work remains largely illustrative and example-driven, focusing on small, well-scoped engineering contexts and demonstrating the feasibility of MBSE-integrated reuse through structured patterns. In contrast, our work advances this concept significantly by tackling reuse at scale across complex safety-critical systems, embedding this within real industrial toolchains, and validating it through longitudinal industrial studies.

Extending that context, we reference the works of Dumitrescu and his research group. In [178], they advance the field of pattern-based systems engineering (PBSE) by developing a methodology that identifies and structures *solution patterns* from existing mechatronic systems. This approach facilitates the meaningful reuse of design knowledge in the development of intelligent technical systems. By employing a matrix-based modularization technique within a model-based systems engineering (MBSE) framework, the method externalizes tacit knowledge and represents functional chains across disciplines. Their structured reuse strategy has been validated through application to systems such as cooperating delta robots, demonstrating its practical utility in complex system design contexts.

Husung’s more recent work investigates collaborative MBSE practices using SysML v2 and dataspaces to facilitate distributed engineering across organizational boundaries [179]. While that work extends the relevance of MBSE into inter-organizational contexts, the focus of this thesis remains internal to a single industrial organization. Specifically, it addresses internal reuse challenges related to siloed development, loss of relevant information, and difficulty managing emergent behaviors in increasingly integrated automotive systems.

The most directly related work to our generalized concept is that of Quentin Wu (University of Lorraine) and collaborators Gouyon, Levrat, and Boudau. Their 2020 publication "*Use of Patterns for Know-How Reuse in a Model-Based Systems Engineering Framework*" [180] addresses the challenge of reusing engineering knowledge within Model-Based Systems Engineering (MBSE), aiming to improve efficiency amid increasing system complexity and competitive market pressures. Their approach evaluates existing reuse strategies such as Commercial Off-The-Shelf (COTS) components and set-based design (as used at Toyota), ultimately advocating for Design Patterns – structured, reusable model templates – as the most effective mechanism for capturing and transferring expert know-how.

Their primary contribution is the "Mining-Maturation-Implementation" (MMI) methodology, initially developed by Wu in his 2019 PhD thesis [181]. MMI outlines a

structured approach: mining past projects for reusable solutions; generalizing them into abstract, tool-independent patterns using a custom meta-model; and implementing them in new contexts via transformation into tools such as Capella. Illustrative figures in their publications show pattern definitions and how they are instantiated across projects. This reliance on a custom meta-model introduces a parallel modeling layer, requiring a dual translation effort: first into the custom model, then into a target tool via dedicated transformations. Potential gaps in the model transformations due to the varied expressivity of practitioners' tools are not systematically addressed. While domain-adequate in principle, this approach remains disconnected from evaluating or extending existing tooling, and follows the broader academic trend we have detailed in the past pages of defining successful reuse in idealized modeling environments rather than integrating into the practitioners' actual toolchains.

In contrast, our approach is explicitly tool-agnostic. We do not prescribe or require migration to a new modeling framework or meta-model. Instead, we define what reuse-critical information must be captured and evaluate how existing tooling environments support this. This allows practitioners to extend their own environments in a targeted way – prioritizing gaps and tailoring the solution to their organizational context.

Like our approach, Wu et al. emphasize the importance of avoiding tool lock-in and promote reusable pattern libraries across modeling environments. They report that pattern-based reuse helps engineers focus on design logic, not syntactic details [182]. The aerospace case study – reusing generalized electrical system architectures - *qualitatively* demonstrates benefits such as improved design speed and consistency.

While Wu et al. contribute a rigorous conceptual framework with academic validation of pattern-based reuse in MBSE, our thesis diverges notably in both focus and integration scope. Wu et al. emphasize theoretical generality and MBSE enablement across environments, while our research targets **direct reuse of complex system solutions** within a single large-scale industrial context. Our principal innovation is the embedding of reuse-critical information into existing engineering artifacts via structured attribute templates. This allows reuse to occur natively within the practitioner's own tools, without additional translation steps or model projection.

Moreover, our work includes extensive **longitudinal industrial validation** within a leading automotive Tier-1 supplier. Over several years, our method has captured and reused complex vehicle system solutions across architectural views (functional, logical, technical, etc.), demonstrating measurable gains in design consistency, reduced redundancy, and deployment efficiency. These outcomes go beyond the largely qualitative evaluations presented by Wu et al, by demonstrating real-world feasibility and impact quantitatively validated by a systematic expert survey.

Finally, it is important to highlight that the foundational ideas and methodological contributions presented in this thesis significantly predate Wu et al.'s publications. Specifically, the core methodological framework for operationalizing design patterns

developed by the author and presented in this thesis – developed incrementally since its initial conceptualization in 2013 [183] and published in a mature form at the Pattern Languages of Programs (PLoP) 2018 conference [184] – was first to explicitly introduce the operationalization of Design Patterns and leverage it to embed design pattern information directly into engineering artifacts within practitioners' MBSE environments with the goal of enabling and enhancing systematic direct reuse in practice.

In summary, Wu et al. provide a thoughtful and structured academic framework for MBSE pattern reuse. While their contributions offer valuable conceptual framing, our work is distinguished by its earlier development, deeper integration into tool-supported environments, and direct industrial validation. In a subsequent 2021 publication [185], Wu et al. introduce an *organizational* maturity model that assesses and enhances reuse readiness, covering aspects such as training, governance, and culture. These elements complement our technical method and are revisited in Chapter 7 in the context of our longitudinal case study, where we reference Wu et al.'s maturity model as a potential complement to our approach for organizations aiming to scale reuse capabilities effectively.

4.5 A Design Pattern Template for Safety Mechanisms

In this thesis section we deal with the particular class of patterns known *Safety Mechanisms*, introduced in detail in previous sections, which we will use to initially scope our approach.

When cataloguing specific pattern classes, such as the work on redundancy patterns by Armoush [122], the authors tend to hand-pick design pattern attributes based on relevance but also arguably their ease of documentation for the pattern collector. In contrast, we included all possible attributes relevant to the solution class we want to reuse, to leverage the wide variety of models available in practice, especially domain-specific ones, in order to most adequately capture specific information. This additionally underscores our postulation that the more detail-rich and context-adequate or domain-specific the data-model behind a tooling/modeling environment, the higher the number of attributes one can capture in the models and the better the quality and degree of attribute capturing for reuse.

In light of the nature of *Safety Mechanism Patterns* (SMP) as discussed in section 3.3, and referring to the various relevant attributes we expounded on in section 3.4.1 – and contrasted in the previous section – we chose the following thirteen attributes for our SMP Design Pattern Template:

- **Name**
- **Intent**
- **Motivation**

- **Applicability**
- **Structure**
- **Participants**
- **Collaboration**
- **Consequences**
- **Implementation**
- **Implications**
- **Usage Classification**
- **Example**
- **Related Pattern**

Each of these attributes is explained in detail in the next section, including what the capturing of the pattern in practice in the automotive industry entails. This is followed by a discussion of the criteria that went into the choice – as well as exclusion – of attributes.

4.5.1 Safety Mechanism Pattern Attributes

This section covers the characteristics selected for capturing the safety mechanism patterns. The characteristics are identified by their name, with an abbreviation provided in brackets after each name.

Name (n)

The name of a pattern should not only be unique but should capture the purpose and actions of a pattern in an easily identifiable manner. If the pattern is new and not already well-established in the practitioner community, then special care should be given the choice of name, so that practitioners can then use it to quickly and unambiguously communicate the concept and principle of their safety mechanism pattern, leading to the name establishing itself in their community.

In practice: The reuse mechanism should provide means to store the pattern name. This is easily done for automotive implementation artifacts, but this attribute should not only be attached to the implementation artifacts; it should traceably extend to all other artifacts relevant for the pattern reuse.

Intent (i)

Reflects the purpose of the pattern. For instance, the pattern may allow the plausibilization of a signal or provide conformity to a regulation. The pattern developer should use this attribute to provide a clear understanding of what the pattern does to any potential users.

In other words, what is the purpose of the pattern? What does it actually do?

In practice: The purpose of Safety Mechanism Patterns is fault and error management. The Intent should be captured in the pattern requirements, but also supported by practical information reflected in a dysfunctional view – capturing the malfunctioning behavior of components – as well as through failure propagation analyses. Furthermore, there should be a possibility to link the *Intent* description with an accurate hazard description – according to the ISO26262 standard, as seen in Figure 4 – as well as to the safety case.

Motivation (m)

While the *Intent* gives purpose and thus covers what the mechanism actually does, it may still not be clear how this contributes to the fulfillment of the safety mechanism's primary goal. This attribute thus covers how a design pattern will solve a problem and hence the rationale behind using it. For instance, an Alive Counter is a mechanism that appends an incremental value to signals before sending them. This is its *Intent*. Its *Motivation* however, is to make sure that the receiver does not get outdated data due to sender or communication failure.

This attribute supports practitioners to understand – and thus correctly apply – the pattern to suit their needs. It also constitutes the main argument as to why the mechanism is useful in its context, making it a strong contributor to the pattern's safety case.

In practice: The rationale of using a Safety Mechanism Pattern is best described using a combination of requirements, argumentation (safety case), and practical information based – again – on a malfunction description and failure propagation analysis.

Applicability (a)

Provides a description of the situation and conditions to which the Safety Mechanism can and should be applied, as well as counterexamples. For Safety Mechanism Patterns, this particularly includes structural constraints, as well as any assumptions or contextual information necessary for the correct application of the pattern. In addition, it may also be helpful to include information providing guidance on when to apply the pattern. A good example would be Armoush's Decision Trees [122] for selecting the appropriate kind of redundancy pattern, depending on system design.

In practice: How well the applicability of a pattern is captured is directly related to how well the development environment captures and provides contextual information. In the absence of such a capability, much of this attribute's information will be documented using whatever implementation description is available, aided by textual entries.

Structure (s)

Provides the representation of the elements of a pattern and their relationship (preferably in graphical form). This includes the development components as well as their relation to the requirements they should fulfill, the context in which they are stated and the specifications they implement. Elements of the structure should be labeled clearly to increase the understandability of the pattern.

In practice: The structure of a pattern's implementation artifacts can be documented relatively well using the graphical tools available today, albeit with varying degrees of precision and detail. Linking these artifacts to requirements and other elements in domain-specific context is the problem.

Participants (p)

This attribute provides a clearer picture of the objects involved in the function of the Safety Mechanism. As such, it plays a complementary role to the *Structure* attribute by providing a description of each of the safety mechanism pattern elements, including potential instantiation information. The element descriptions should clarify their role within the overall safety mechanism pattern.

In practice: Capturing the *Participants* of a pattern, as well as its *Structure*, is heavily dependent on the tool or development environment offering a framework which includes all the relevant elements necessary to describe the pattern.

Collaborations (cl)

This attribute shows how the pattern's various elements (for example, sources of contextual information, argument structure, requirements, logical components) collaborate to achieve the stated goal of the pattern. Moreover, this attribute focuses on capturing any links between the pattern elements that could not be captured clearly and explicitly or were not the focus of the *Structure* attribute. Capturing this information explicitly is essential for the correct use of the pattern by safety practitioners.

In practice: To capture the *Collaborations* of a pattern's elements, the development environment must not only offer a framework that allows linking these elements, it must also provide a means to show how the interaction occurs. This could be covered, for example, by a logical behavior description of the implementation artifacts, reinforced by malfunctioning behavior description and failure propagation analysis, and held together using a safety case, which describes how all these elements fit together and collaborate.

Consequences (cn)

Captures information pertaining to the instantiation and deployment of the pattern, as well as to the impact it may have on the system. This characteristic particularly includes which

activities need to be carried out when applying the pattern. This includes technical instantiation of the pattern, and the establishing of links to other elements as well as any process activities that need to be carried out by the safety engineer. The purpose here is to ascertain that the user knows exactly what the pattern does and how it is used.

In practice: This attribute requires a mechanism for storing and reusing the pattern, which can leverage all the capabilities of the development environment in use.

Implementation (im)

This attribute describes the implementation and should include its possible failure modes or dysfunctional behavior as well as any constraints or possible pitfalls. It may also include additional information about the application of the pattern. For instance, the order in which the pattern elements are developed.

In practice: Capturing all details of this attribute will rely heavily on the tool capabilities.

Implications (ic)

This attribute covers the impact of applying the pattern on non-functional aspects of the system. This may include traits such as reliability, modifiability, cost, and execution time.

In practice: This attribute is trickier to capture, because non-functional aspects are traditionally more difficult to capture [35] and have less tooling support. A few integrated design tools – especially those incorporating aspects of software Product Lifecycle Management (PLM) – can handle some, not all of the aspects of this attribute. For instance, when analyzed in 2014, PREEVision²⁷ could analyze and document component costs, perform simple reliability analysis and evaluate some modifiability metrics for software and system architectures. To handle the other aspects of *Implications*, the development environment would also have had to cover timing analyses and failure propagation at least. Some of these capabilities have since been introduced.

Usage Classification (uc)

This attribute categorizes the safety mechanism patterns according to their usage in context of fault/error/failure mitigation techniques (avoidance, detection, handling) [44] as depicted in Figure 8 and discussed in section 3.3.

In practice: This attribute is best captured by a dedicated element as part of a failure propagation / safety analysis tool support.

Example (e):

²⁷ MBSE Tool addressing Engineering Analyses & development tasks for automotive E/E Architectures
<https://www.vector.com/int/en/products/products-a-z/software/preevision/>

Examples are an important aid for practitioners to better understand the pattern, especially if visual. Their significance increases with the abstractness of the pattern. This section should thus provide explicit examples that illustrate the instantiation of the pattern, which is especially important if the other characteristic descriptions do not provide such an aid. This attribute is optional, used only where needed. Multiple examples should be used to depict diverse instantiations of the pattern if needed, whereas if only one example is provided, then it should depict the most typical instantiation.

In practice: Examples are rarely used; practitioners rely instead on graphical diagrams which are easier for others to understand.

Related Patterns (rp):

This section provides a listing of safety mechanism patterns that are related to the one being captured. This may, for instance, include patterns that are derived from this pattern.

In practice: Truly capturing related patterns in a way that makes this attribute useful requires a capable development environment, and a pattern reuse mechanism capable of leveraging these capabilities.

4.5.2 Choice of Template Attributes

This section discusses the choice of characteristics, focusing especially on why some attributes typically found in other related Templates were left out of ours.

Most of the related work we have reviewed follows one of two patterns:

- 1) **General Patterns:** the authors define a Template of general attributes (e.g., the POSA template [100]) used to describe any Design Pattern, generalizing their experience with their pattern, which often happens in a specific setting. While adding to the general knowledge about design patterns, this approach leaves a good deal of room for interpretation in applying the patterns to any specific setting, and does not cover the role of development tools in the quality of attribute coverage.
- 2) **Specific Patterns:** In contrast, the authors of such Patterns define a framework to tackle an issue, and then define an attribute Template to match it. When defining Templates for specific pattern classes the authors tend to hand-pick design pattern attributes based on relevance, but also arguably their ease of documentation within the framework predefined by the authors.

In contrast, we chose to include any potentially relevant attributes. This was done to leverage the concept of using the wide variety of models, especially domain-specific ones, available to practitioners, to most adequately capture specific information. This also reinforces the proposition presented previously, namely that the quality of attribute information capturing is directly related to how detail-rich and context-adequate or domain-specific a modeling framework is.

The characteristics described in the previous section describe the information that should be captured in order to define a Template a safety mechanism properly and allow for its reuse. These were chosen primarily to cover the attributes of Safety Mechanisms as analyzed in section 3.3, and regardless of the documentation and representation technique, instead focusing on information that should be captured for correct reuse.

As for the common attributes we omitted:

Also Known As

This attribute records alternative names that are used for the pattern.

While useful for documenting a pattern which may recur under different monikers in various organizations or across multiple domains, this characteristic was purposefully left out of our Template, because we target the practical use of the pattern within a development environment. In that context, the ambiguity implied by multiple identifiers for the same pattern is strongly discouraged.

Known Uses

This attribute typically captures known uses of the pattern. As with the Example section, this aims at increasing the understandability of the pattern for practitioners.

For safety mechanism patterns, this information should be captured in the Rationale, Motivation and Collaborations attributes, with any remaining details covered by an example, making this attribute superfluous. If needed, the information about what product designs use instances of a particular Safety Mechanism should be provided by the traceability mechanisms of the practitioner's development environment, into which Pattern Library approach has been instantiated.

The next section gives more details on the structure of the pattern library.

4.6 A Pattern Library for Safety Mechanisms

Using a simplified categorization of safety mechanisms according to the most common fault/error/failure management subtypes (avoidance / detection / handling) we define a pattern library covering known solutions and architectural measures /constraints in a seamless holistic model-based approach with corresponding tool support, introduced by the author in [196]. The pattern library comprises the minimum set of elements needed for correct reuse, i.e. the requirements the pattern covers, the specification of how one plans to implement it and the architecture elements / measures / constraints required as well as the supporting safety case template, and may include deployment or scheduling strategies, which would then be integrated into existing development environments. This enables an early analysis of hazards and risks, which is recommended by many safety standards. Subsequently, fault types can be matched both to probable hazards but more

importantly to the problem categories they fall into or are most similar to, from a system architecture design viewpoint. Combining this with known architectural constraints and patterns for solving them, we can thus reason about which types of architectural patterns are relevant for the system under analysis. The fault types, along with their requirements, are bundled with solution arguments, comprising components, their architectures, deployment plans and schedules, into pattern libraries, which are rounded up by the corresponding safety-case templates or skeletons to provide argumentation for achieving the goals.

Underlying the approach is the consistent use of patterns from the categorization of hazard types, over the abstract modeling of the respective safety concepts, and down to their implementation in the system architecture description, with a focus on providing argument chains in a seamless model-based environment, allowing the co-evolution of integrated safety and system architecture models.

In this chapter, we mainly focus on defining the structure, mapping and instantiations of elements from a library of reusable patterns, including the data models and workflow required to support seamless integration into a model-based development environment. Automation via tool-support is essential to achieve the intended advantages, and is thus the focus of the evaluators investigated in the next chapters.

4.6.1 Pattern Library Elements

Figure 11 gives a sketch of a generic library element comprising development artifact categories – requirements, specification, and implementation description, side by side with the supporting argumentation artifacts. The left part of the schematic gives possible development artifact elements of the reusable pattern, while the right part shows the corresponding argumentation artifacts comprising a safety case skeleton – shown exploded for clarity – with connections to the relevant development artifact categories. The argumentation artifacts are displayed exemplarily in a graphical notation, in this case GSN [58] – introduced in section 2.4.3. Both sides together comprise the pattern library element, the building blocks of which are the four artifact categories requirements, specification, implementation and argumentation.

While the left part of the schematic, comprising the various development artifacts necessary for reuse, can entirely originate in one seamless tool – as will be shown in our first evaluator in chapter 5 – this is not necessary. As long as the artifacts' relations are well-defined (see section 4.6.2), the development artifacts may reside in multiple repositories and be in varying formats; the binding element is the safety case shown on the right hand side, which gives the story and rationale behind the reusable pattern, supported in-tool by the pattern library mechanism. This specific aspect is particularly important for reuse in a safety-critical development context, and forms the cornerstone of our approach.

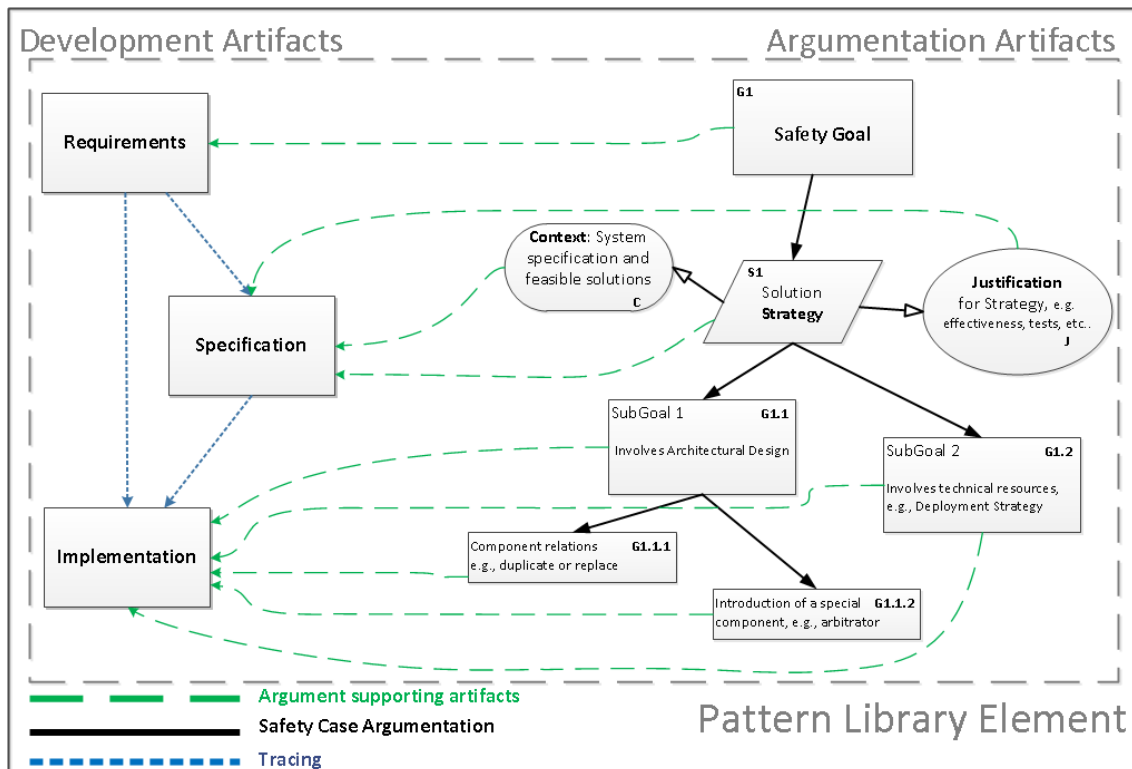


Figure 11: Generic Pattern Library Element, reproduced from [21]

Depending on the development environment used, the relation between artifact categories will differ according to the definitions provided for how the different category views (if available) interact. The pattern library mechanism can then be in general terms interpreted as a weaving model, which binds the pattern elements together and allows their reuse. Additional mechanisms for the instantiation, also depending on the development environment, may be necessary. Based on the formal nature of the frameworks we made the first instantiations in, the categories *Requirements*, *Specifications*, and *Implementations*, are understood to be complete – covering both sides of the V-Cycle; i.e., Requirements, for instance, are formalized enough in their formulation to allow direct validation against them. This idealized understanding does not carry to the industrial case-study presented in chapter 7.

In order to investigate the instantiation of the approach, it is necessary to first provide some basic definitions of the pattern library and its elements. We start with an overview and motivation for capturing patterns in their development artifacts.

4.6.2 Using models to capture Patterns

Model-based development has been increasingly popular in the development of embedded software systems in the automobile industry over the past two decades [94], which provides an excellent opportunity for capturing and linking *product* safety information. As such, capturing patterns using models, optimally the actual development

artifacts of the system under development, is much more practical and preferable to doing so in *text* and *diagrams* in this context. There are several reasons:

- Using the models from the product's development environment guarantees that safety practitioners base their analyses and activities on the same artifacts as their colleagues developing the system, i.e., the co-evolution of system and safety models.
- The different models in a product's development artifacts are linked to each other, if not directly by transformation, then at least within the development process, for instance, through work-products and activities. Thus, much of the information is purportedly captured in the existing models.
- If this is not the case, then it is a good question to ask why not? Is this known to the developers and system engineers? How does this impact the quality of the final product?
- If the information is indeed captured in the models, then why transform it to texts and diagrams, only to have to transform it back again for reuse? This is essentially a back-and-forth model-transformation, which always risks loss of information.
- Last but not least, free-wheeling information in textual lists – not directly linked to implementation artifacts – has a good track record of being ignored.

Using the models also supports linking the argumentation structures with the architecture design and product information supporting them. This last aspect is very important, as:

- it increases the traceability of the product and system design information and understandability of the argument structure;
- and more importantly,
- it allows a co-evolution of the argument structure with the system design.

This allows an early identification of deficiencies in the argument, as well as facilitating documenting and maintaining the safety case throughout the entire product life cycle.

As a quick reminder of the model-based architectural paradigms introduced in section 2.5.1, to achieve separation of concerns, it is useful to avoid the construction of large monolithic models, which are by definition difficult to handle, maintain or reuse. This simultaneously implies that these different models or model views – if not initially thus defined – will require integration into a final coherent model. This linkage between the different models may be provided by three means:

- **Model transformation:** where there are clear and precise transformation models between the different development views, such as those introduced by the OMG's MDA [72] approach, as presented in section 2.5.1.
- **Model-weaving:** while the concept is not really new, the term itself has no universally accepted definition as yet, but [71] defines it as the operation of setting fine-grained relationships between models or meta-models, with a focus on the associated execution semantics for the application domain under consideration.
- **Modeling frameworks:** such as the SPES MF [73] or UAF²⁸[186], leverage both model transformations and model-weaving to offer predefined links between their different models, or modeling views, for different stakeholders (separation of concerns) as well as granularity layers (divide and conquer), as also introduced in section 2.5.1.

Model-weaving is especially useful to describe relations between different models (or views) with no predefined relations, for instance; to link previously unrelated artifacts from different views for a new purpose. The models to be linked may be unrelated models addressing separate concerns; e.g., linking a requirements model from one tool with an architecture model from another tool with an argumentation model in a third tool to express safe design. This particular notion is especially useful, as it describes the disparate tool landscape of today's automotive domain.

It is important to note that transformation and weaving are different approaches, even if a weaving model may in some cases resemble or even be a transformation model. For each separate model or particular view of a system, there may exist a meta-model representing a domain-specific language dealing with model's particular concerns. Transformations between or from these models may be difficult to define, while model weaving links permit describing the aspects both separately and in combination. As described in [187], transformation differs from weaving mainly in:

- **Parity:** A transformation commonly takes one model as an input and produces another model as an output. This holds even if multiple input and output models are considered. Model weaving links two input models using a weaving meta-model. The combined result is the output.
- **Automaticity:** Transformations are essentially automatic operations, while in weaving the user may need assistance through additional heuristics or guidance.

²⁸ UAF is a standardized framework developed by the Object Management Group (OMG) to support enterprise architecture modeling across various domains, including defense and commercial sectors. It integrates elements from existing frameworks like DoDAF, MODAF, and NAF to provide a comprehensive approach to enterprise architecture.

- **Variability:** Transformation conforms to a fixed meta-model (if there is a transformation language) or at least a transformation model. In weaving, on the other hand, a new weaving meta-model is created for each particular purpose or application, so there exists in essence no canonical standard weaving meta-model.

By leveraging the “everything is a model” principle, model weaving offers several advantages, as summarized in [188]:

- Specialized weaving models can link all the necessary information and relationships between the considered models, thus avoiding unnecessarily constructing large meta-models for capturing all the aspects of a system.
- Furthermore, the separate meta-models describing the concerns of separate model can continue to be developed independently, while maintain the interconnections by weaving models, essentially creating a “lattice of meta-models” [71].

In fact, there are great similarities in the foundational concepts of model-weaving, model federation, and data fabrics. All share a common goal of connecting disparate artifacts to achieve an integrated whole. Model-weaving treats models as primary engineering artifacts – in MBSE, “*models are first-class citizens of the engineering process*” [189] – and weaves separate models together via explicit links. This *models-as-first-class* paradigm directly inspired the concept of model federation in digital engineering, which similarly links heterogeneous domain models to form a unified system-of-systems view while each model remains in its native format [190]. In parallel, data fabrics in data-driven engineering and machine learning provide a unifying architecture that “*stitch[es] together multiple heterogeneous data sources and types*” into a single abstracted layer [191]. Both model federation and data fabric approaches emphasize interoperability and integration across heterogeneous systems, enabling information to flow consistently across diverse models or data silos [192]. The result is a cohesive, federated environment in which diverse models or datasets can coexist, interconnect, and collectively support complex engineering analyses – a perspective that balances model-driven, digital, and data-centric paradigms in modern MBSE.

Our approach is not limited to one kind of model-linkage, although our evaluation examples focus on modeling frameworks, supported by model-weaving, as will be discussed in further in later sections.

4.7 Artifact Reference Structure for Safety Mechanism Patterns

We begin with a quick reminder of the attributes we defined in section 4.5.1 for the Safety Mechanism Pattern Template, and their symbols. Based on this, the following sections will explain and define the approach and its use.

The Safety Mechanism Pattern is described using the following attributes {Name (N), Rationale (R), Motivation (M), Applicability (A), Structure (S), Participants (P), Collaborations (CL), Consequences (CN), Implementation (IM), Implications (IC), Usage Classification (UC), Example (E), Related Patterns (RP)}.

4.7.1 Basic Definitions of Safety Mechanism Patterns

A quick observation of the above attribute list for Safety Mechanism Patterns reveals that each **attribute** ($N, R, M, A, S, P, CL, CN, IM, IC, E, RP$) is actually an information **tuple**. It contains both the attribute's name or identifier – a string – *and* the information content of the attribute. The information content may vary depending on the form of expression, degree of detail, level of abstraction, completeness of information and so on. As we will later want to evaluate, merge or compare the information content of attributes, we will consider this content information to be represented as a set – whose content is abstract – for simplicity.

Hence, a **Safety Mechanism Pattern** (SMP) is defined as an abstract object whose content can be retrieved by the operator *decomp*, as follows:

$$decomp(SMP) = (N, R, M, A, S, P, CL, CN, IM, IC, UC, E, RP)$$

Equation 1: Basic Definition of Safety Mechanism Pattern (SMP)

where $N, R, M, A, S, P, CL, CN, IM, IC, E, RP$ are the aforementioned **attribute tuples**. The information content can only be precisely described once the approach has been instantiated into a development environment, which then determines the actual content of the set – and its form. Two operations are possible for each attribute:

- *id*(attribute): which returns the name of the attribute as a string, and
- *content*(attribute): which returns the information content of the attribute as a set.

We can then expand the definition to the following general form:

Let C be a set of atomic contents, and S' be the set defined by the following equation:

$$S' = P(C)^{12}$$

The set S of Safety Mechanism Patterns (SMP) is the set of all $s \in S'$ such that s is finite, and for all $n, i, m, a, s, p, cl, cn, im, ic, uc, rp \in P(C)$, such that $s = (n, i, m, a, s, p, cl, cn, im, ic, uc, rp)$, the sets $n, i, m, a, s, p, cl, cn, im, ic, uc$ are not empty. For this set, we introduce the following notations:

$$name\#(s) \stackrel{\text{def}}{=} n$$

$$intent\#(s) \stackrel{\text{def}}{=} i$$

$$applicability\#(s) \stackrel{\text{def}}{=} a$$

$$motivation\#(s) \stackrel{\text{def}}{=} m$$

$$structure\#(s) \stackrel{\text{def}}{=} s$$

$$participants\#(s) \stackrel{\text{def}}{=} p$$

$$collaboration\#(s) \stackrel{\text{def}}{=} cl$$

$$consequences\#(s) \stackrel{\text{def}}{=} cn$$

$$implementation\#(s) \stackrel{\text{def}}{=} im$$

$$implication\#(s) \stackrel{\text{def}}{=} ic$$

$$usageClassification\#(s) \stackrel{\text{def}}{=} uc$$

$$relatedPattern\#(s) \stackrel{\text{def}}{=} rp$$

Equation 2: Basic Notations of Safety Mechanism Patterns (SMP)

In the above definition, the following specific notations were introduced:

- $name\#(s)$ denotes the information content of the name attribute of the set s , as defined in section 4.5.1.
- $intent\#(s)$ denotes the information content of the intent attribute of the set s , as defined in section 4.5.1.
- $motivation\#(s)$ denotes the information content of the motivation attribute of the set s , as defined in section 4.5.1.
- $applicability\#(s)$ denotes the information content of the applicability attribute of the set s , as defined in section 4.5.1.
- $structure\#(s)$ denotes the information content of the structure attribute of the set s , as defined in section 4.5.1.

- $\text{participants\#}(s)$ denotes the information content of the participants attribute of the set s , as defined in section 4.5.1.
- $\text{collaboration\#}(s)$ denotes the information content of the collaboration attribute of the set s , as defined in section 4.5.1.
- $\text{consequences\#}(s)$ denotes the information content of the consequences attribute of the set s , as defined in section 4.5.1.
- $\text{implementation\#}(s)$ denotes the information content of the implementation attribute of the set s , as defined in section 4.5.1.
- $\text{implication\#}(s)$ denotes the information content of the implication attribute of the set s , as defined in section 4.5.1.
- $\text{usageClassification\#}(s)$ denotes the information content of the usage classification attribute of the set s , as defined in section 4.5.1.
- $\text{relatedPattern\#}(s)$ denotes the information content of the related pattern attribute of the set s , as defined in section 4.5.1.

Having introduced these basic definitions, we can proceed to investigate pattern relations more precisely, starting with how patterns can relate to each other, as discussed in the next section.

4.7.2 Restructuring to capture relations between Design Patterns

Recalling the attribute representations from section 3.4, *Related Patterns* are traditionally documented as textual lists. Yet an analysis of the Safety Mechanism Pattern class addressed in this thesis, and discussed in detail in section 3.3, led to the conclusion that the attribute *Related Patterns* is not random or without structure but could be expressed by object-oriented class and object relations, greatly improving the precision and usability of the attribute information as well as traceability between related patterns.

This conclusion is not unsubstantiated. Zimmer introduced the necessity of defining relationships between related patterns in [193], and Noble built on this work in [194], defining a number of relationships between system design patterns such as *uses*, *refines*, *used by*, *combine*, and *sequence of*.

Kelly and Wu arrived at a similar result in 2004 [44], when they used object-oriented terms to describe the properties they observed for their safety *tactics*:

- Safety tactics can be *refined*. They can exist at various levels of abstraction.
- Safety tactics are not necessarily *independent*. The application of one tactic may assume that another tactic has already been *defined* and *applied*.
- Safety tactics can be *combined*.

In contrast to our work presented here, the definitions provided previously for these relations are at a very abstract level, which does not support an expressiveness for detailing what parts of the related patterns are *refined*, *combined* or *used by* other patterns. A precise modelling of pattern relationships is thus not possible.

We continued this line of thought with the aim of more precisely describing the relations. Our analysis led us to deduce that Safety Mechanism Patterns were specifically related to each other – that is, there exists a dependency – in one of two manners. The formalization follows, but the simple explanation is:

- **Specialization**: Patterns with similar purposes using the same basic design principle can be collected under a generalized description, in an “Is-A” inheritance-type class relation. Inversely, abstract descriptions of safety mechanism patterns can then be further specialized into increasingly more specific descriptions based on several criteria, such as context or implementation technology. These subclasses retain all the Design Pattern attribute information of the parent class and add to them as suitable or necessary.

A clear example is *Redundancy*. It can be described and its attributes can be captured at a very high level (using duplicate or equivalent sources of information to increase reliability and/or overcome single channel vulnerability). It can then be further specialized according to implementation technology (HW-based, SW-based, Mixed) and system context (availability of resources, such as control flow monitoring, or a dedicated watch-dog) as presented in [122] and discussed in chapter 3.3.

- **Inclusion**: Design Patterns may include other Design Patterns in a special form of “Has-A” object relationships, which includes attributes of both composition and aggregation. As such, a Composite Design Pattern consists of a full attribute Template specifying a complex function, which is nevertheless incomplete and thus not usable without the aggregation of attributes from the included or used pattern.

A straightforward example is also given by *Redundancy*, which features redundant sources of information that require some form of *Selection* (switching, voting, comparison, etc.). *Selection* is a category of design patterns in its own right. It is, however, also included in any description of *Redundancy*. However, while the Design Pattern *Selection* can exist on its own, *Redundancy* cannot be meaningfully defined without first defining *Selection*.

We could, in fact, argue the existence of a third type: **Combined Relation**. Revisiting the *Redundancy* example, we find that it, in fact, represents a situation in which both **Inclusion** and **Specialization** relations can occur simultaneously. We need to define these two basic relations first, in order to be able to correctly define combined ones.

Thus, instead of listing *Related Patterns* (RP) textually, we will capture them in two distinct attributes leveraging the **Specialization** and **Inclusion** rules. The first attribute,

Specialized Pattern (SP), expresses a connection between the pattern we are describing and the pattern it is a specialization of. The second attribute, *Included Pattern (IP)*, refers to the pattern(s) without which our pattern cannot be coherently and fully described.

Let C be a set of atomic contents, and S' be the set defined by the following recursive equation:

$$S' = P(C)^{12} \times P(S') \times P(S')$$

The set S of Safety Mechanism Patterns (SMP) is the set of all $s \in S'$ such that s is finite, and for all $n, i, m, a, s, p, cl, cn, im, ic, uc \in P(C)$, $ip, sp \in P(S)$, such that $s=(n,i,m,a,s,p,cl,cn,im,ic,uc,ip,sp)$, the sets $n, i, m, a, s, p, cl, cn, im, ic, uc$ are not empty.

For all $s \in S$, and all $n, i, m, a, s, p, cl, cn, im, ic, uc \in P(C)$, $ip, sp \in P(S)$, such that $s=(n,i,m,a,s,p,cl,cn,im,ic,uc,ip,sp)$, we introduce the following notations:

$$name\#(s) \stackrel{\text{def}}{=} n$$

$$intent\#(s) \stackrel{\text{def}}{=} i$$

$$applicability\#(s) \stackrel{\text{def}}{=} a$$

$$motivation\#(s) \stackrel{\text{def}}{=} m$$

$$structure\#(s) \stackrel{\text{def}}{=} s$$

$$participants\#(s) \stackrel{\text{def}}{=} p$$

$$collaboration\#(s) \stackrel{\text{def}}{=} cl$$

$$consequences\#(s) \stackrel{\text{def}}{=} cn$$

$$implementation\#(s) \stackrel{\text{def}}{=} im$$

$$implication\#(s) \stackrel{\text{def}}{=} ic$$

$$usageClassification\#(s) \stackrel{\text{def}}{=} uc$$

$$includedPattern\#(s) \stackrel{\text{def}}{=} ip$$

$$specializedPattern\#(s) \stackrel{\text{def}}{=} sp$$

Equation 3: Extended Notations of Safety Mechanism Patterns (SMP)

Revisiting Equation 2, and replacing the attribute *Related Patterns RP*, with the two attributes *Specialized Pattern SP* and *Included Pattern IP*, yields the extended definitions for *SMP*, shown in Equation 3, where:

- The first eleven notations remain the same as provided in Equation 2.

- $includedPattern\#(s)$ denotes the information content of the included pattern attribute of the set s .
- $specializedPattern\#(s)$ denotes the information content of the specialized pattern attribute of the set s .

Similarly, revisiting Equation 1 yields the following extended basic definition for *SMP*.

$$decomp(SMP) = (N, R, M, A, S, P, CL, CN, IM, IC, UC, E, SP, IP)$$

Equation 4: Extended Basic Definition of Safety Mechanism Patterns (SMP)

Using the simple notations provided in Equation 3, it is possible to determine whether a pattern is *atomic* or *composite*. **Atomic** patterns do not include other pattern and thus their attributes form a consistent and complete description of a Safety Mechanism Pattern on their own. **Composite** patterns are understood as “not-atomic”, and cover all three pattern relations described in the previous section (*inclusion, specialization, combined*). Hence:

Let $s \in S$, s is an atomic SMP iff:

$$includedPattern\#(s) = \emptyset$$

$$specializedPattern\#(s) = \emptyset$$

Equation 5: Simple definition of a single atomic pattern

To be able to perform evaluation and comparisons beyond an individual pattern, we will need to update this definition using the attribute operations (*id, content*) defined in the previous section. Where Π_v denotes the projection on an attribute v of the content of a Safety Mechanism Pattern, we redefine atomic patterns in general as follows:

$$\forall SMP: atomic(SMP) \Leftrightarrow \left(content \left(\Pi_{IP}(decomp(SMP)) \right) \right) = \emptyset$$

$$\forall SMP: composite(SMP) \Leftrightarrow \neg atomic(SMP)$$

Equation 6: Generalized definition of atomic patterns

For composite patterns, the case is more complex. Since a pattern may now rely on other patterns to carry out its task and completely relay all the information concerning its functionality, we can no longer solely rely on the decomposition of the information carried within the pattern to describe its entire functionality. We thus introduce a new operator (*attr*) which returns the set of attributes that comprehensively and consistently

describe a Safety Mechanism Pattern (*SMP*). If our pattern is atomic, this becomes a trivial case, as this set is the list of attributes contained within the pattern itself.

We can thus provide a bounding preliminary definition of the *attr* operator, limited to atomic patterns, and expressed by:

$$\forall SMP: \text{atomic}(SMP) \Leftrightarrow (\text{attr}(SMP) = \text{decomp}(SMP))$$

Equation 7: Initial definition of "attr" operator

To reach a more general definition of the *attr* operator – one valid for *composite* patterns as well – we first have to understand how attribute information is shared between composite patterns, necessitating a closer look at the **Inclusion** and **Specialization** relations.

4.7.2.1 Inclusion Relation

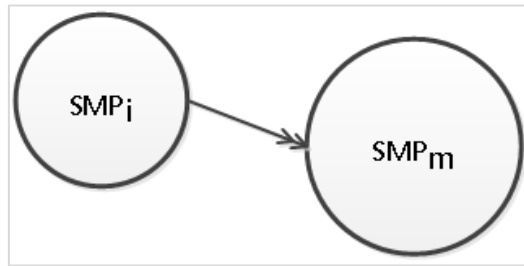


Figure 12: Simple representation of design pattern **Inclusion** relation

We express the Inclusion relation by a double headed straight arrow, as seen in a simple graph-like representation in Figure 12. In this depiction, the **Inclusion** relation means that the composite pattern SMP_i uses another Design Pattern SMP_m . The composite pattern SMP_i consists of attributes which build on corresponding ones from SMP_m and can only have meaning if the Included Pattern SMP_m exists. Thus, a complete and consistent description of SMP_i will *aggregate* information from SMP_m . The **Inclusion** relation can thus be generally defined as:

$$\forall SMP_i, SMP_m: SMP_i \Rightarrow SMP_m \Leftrightarrow \text{content} \left(\Pi_{IP}(\text{decomp}(SMP_i)) \right) \ni SMP_m$$

Equation 8: General definition of Pattern Inclusion relation

To define what the **Inclusion** relation means for the individual pattern attributes, we must define how the aggregation of information in the attributes of composite patterns is performed. It is important to note, that not all attributes of the composite pattern are aggregated with their corresponding ones from the included pattern. The *Name*, *Example*,

Usage Classification, *Specialized Pattern*, and *Included Pattern* attributes are by nature unique to the composite pattern and come only from SMP_i . The aggregation of information content from the remaining attributes is expressed by a new operator (*aggr*), defined as:

$\forall SMP:$

$$aggr_{IP}(SMP, v) = \bigcup_{smp \in content(\Pi_{IP}(decomp(SMP))) \cup \{SMP\}} content(\Pi_v(decomp(smp)))$$

for all $v \in \{R, M, A, S, P, CL, CN, IM, IC\}$

Equation 9: Definition of aggregation of content in Pattern Inclusion

Using this definition, we now define another operator (*comp*) which creates the aggregated composite attribute tuples:

$$\forall SMP: comp(SMP, v) = (id(\Pi_v(decomp(SMP))), aggr_{IP}(SMP, v))$$

for all $v \in \{R, M, A, S, P, CL, CN, IM, IC\}$

Equation 10: Definition of the "comp" operator

Based on the previous two definitions, we can proceed to capture the attributes of a composite pattern by expanding the initial definition of the operator (*attr*), introduced in Equation 7 and bounded for atomic patterns, to a more general form, expressed as:

$$\begin{aligned} \forall SMP: decomp(SMP) &= (N, R, M, A, S, P, CL, CN, IM, IC, UC, E, SP, IP) \Leftrightarrow \\ attr(SMP) &= (N, comp(SMP, R), comp(SMP, M), comp(SMP, A), comp(SMP, S), \\ &\quad comp(SMP, P), comp(SMP, CL), comp(SMP, CN), comp(SMP, IM), \\ &\quad comp(SMP, IC), UC, E, SP, IP) \end{aligned}$$

Equation 11: Generalized definition of "attr" operator

Note that the definition of the (*aggr*) operator, given in Equation 9, means that if the pattern SMP is atomic, the aggregation will return only SMP 's attributes. Applying this to the general form of the (*attr*) operator, given in Equation 11, reduces the definition back to the same expression as the form for atomic patterns, as given in Equation 7.

4.7.2.2 Specialization Relation

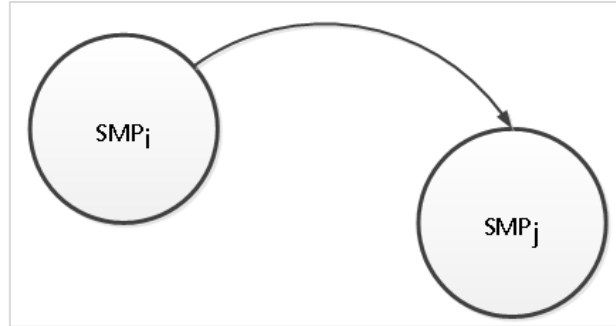


Figure 13: Simple representation of design pattern **Specialization** relation

The **Specialization** relation, depicted by a curved arrow in the graph representation shown in Figure 13, means that SMP_i is a more general pattern of SMP_j or, vice-versa, that SMP_j is a specialization of SMP_i . The **Specialization** relation is defined by:

$$\forall SMP_i, SMP_j: SMP_i \sim SMP_j \Leftrightarrow content \left(\Pi_{SP}(decomp(SMP_i)) \right) \ni SMP_j$$

Equation 12: General definition of Pattern Specialization relation

The relation means that a Child-Pattern initially inherits all the attributes of its Parent-Pattern. If we assume that the Child-Pattern's attribute information is consistent (in a non-formal sense), then inherently it cannot contain information contradicting the Parent-Pattern. Furthermore, Child-Patterns must contain more information than their Parent. Thus, the **Specialization** relation between two patterns SMP_i and SMP_j implies attribute relations expressed by the following equations, where the first term expresses that all information in the parent must be included in the child, while the second term infers that the child must contain at least one attribute with more information content than its corresponding counterpart in the parent pattern – otherwise they would be one and the same.

$$\begin{aligned}
& \forall SMP_i, SMP_j: \\
& SMP_i \sim SMP_j \Leftrightarrow \left(content \left(\Pi_{\delta}(decomp(SMP_i)) \right) \subseteq content \left(\Pi_{\delta}(decomp(SMP_j)) \right) \right) \\
& \quad \text{for all } \delta \in \{R, M, A, S, P, CL, CN, IM, IC, UC, IP\} \\
& \quad \textbf{AND} \text{ there exists } v \in \{R, M, A, S, P, CL, CN, IM, IC, UC, IP\} \\
& \quad \text{such that} \\
& SMP_i \sim SMP_j \Leftrightarrow \left(content \left(\Pi_v(decomp(SMP_i)) \right) \subsetneq content \left(\Pi_v(decomp(SMP_j)) \right) \right)
\end{aligned}$$

Equation 13: Definition of attribute content relationships for Pattern Specialization

4.7.2.3 Combined Relations

Despite having consistent ancestors, highly specialized children of the constituent Design Patterns of a **Complex Pattern** – one which includes other patterns – can readily be incompatible. For instance, a general *Redundancy* pattern will include a generic *Selection* pattern, yet their specialized children will not always be compatible. For instance, *Homogenous Duplex Redundancy* (HDR) – two copies of the same component carried by individual channels; and *Majority Voters* – the simplest and most common of voters, where at least $(N + 1)/2$ variant results must agree. Despite having compatible ancestors, these two patterns should not be used together.

It is possible to counteract this complexity by setting some ground rules to simplify this situation.

Rule #1: Incremental specialization

When specializing a composite pattern, the specialization does not need to apply to both the Complex Pattern attributes and the Included Pattern attributes simultaneously. In fact, this should not be done simultaneously. As depicted in Figure 14, a step-wise specialization of the Complex Pattern attributes ($SMP_i \sim SMP_j$), depicted on the left-hand side, or the use of a more specialized Included Pattern ($SMP_m \sim SMP_n$) without changing the content of the Complex Pattern, as seen on the right-hand side, can guarantee a consistent specialization. The lower part of the diagram depicts the undesirable concurrent specialization of both the Included pattern and Complex pattern containing it, which can lead to inconsistencies.

Rule #2: Incremental consistency checks via divergence nodes

It still remains the design pattern developer's role to ascertain the consistency of composite patterns before using them. This is made easier by the capturing of

Specialization and **Inclusion** relation information at higher abstraction levels. If a point is reached when the Complex Pattern attributes (SMP_i) necessitate the inclusion of a specific refinement of the Included Pattern (SMP_m), this specific Composite Pattern (SMP_n) should be saved explicitly, creating a new base type. This point now serves as a divergence node from which further incremental specialization can occur, without fearing inconsistency.

The rules can be explained through the **Redundancy** Design Pattern example. **Redundancy** can be specialized from a high abstraction level to a very specific case while keeping the Selection Pattern in an abstract form, or vice-versa. Once a specialized form of redundancy is found that requires a specific specialized form of Selection, we can define this pairing to be a new base type, from which further incremental specialization can occur.

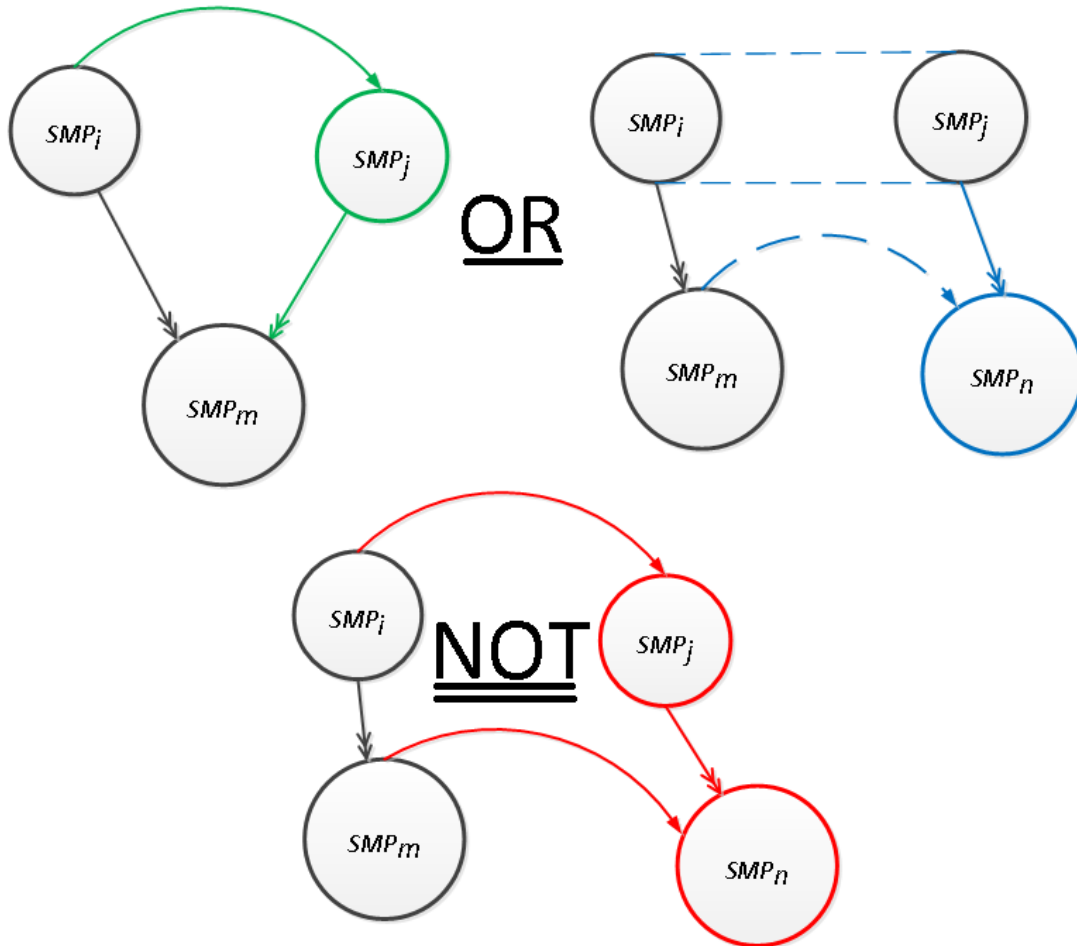


Figure 14: Simple representation of combined design pattern relations

Given the definitions of **Inclusion** (Equation 8) and **Specialization** (Equation 12) provided earlier, it is possible to formulate the combined relationship seen in Figure 14 as follows:

$$\begin{aligned}
& \forall SMP_i, SMP_j, SMP_m, SMP_n: \\
& SMP_i \rightarrow SMP_m \Rightarrow \\
& SMP_i \sim SMP_j \Leftrightarrow \\
& \left(\left(\left(SMP_j \rightarrow SMP_m \right) \vee \right. \right. \\
& \left. \left. \left(\left(\left(decomp(SMP_i) = decomp(SMP_j) \right) \wedge (SMP_j \rightarrow SMP_n) \wedge (SMP_m \sim SMP_n) \right) \right) \wedge \right. \right. \\
& \left. \left. \neg \left(\left(\left(decomp(SMP_i) \neq decomp(SMP_j) \right) \wedge (SMP_j \rightarrow SMP_n) \wedge (SMP_m \sim SMP_n) \right) \right) \right) \right)
\end{aligned}$$

Equation 14: General Definition of Combined Pattern relation

Given these definitions of the Safety Mechanism Pattern, we can now proceed to the instantiation of the approach into model-based development environments.

4.8 Instantiation Guideline and Artifact Mapping

Model-based development, in one form or another, is gaining ground in the automobile industry, as discussed in section 4.6.2. Different models, generating multiple artifacts, are available and in use. Whether by design or, just as likely inadvertently, these artifacts capture many of the *SMP* attributes we have defined in the previous sections. Using our approach, we aim to guide the reuse of SMPs. This serves three purposes:

- In an application context, it gives guidance for the correct capturing of all the attributes necessary for reuse.
- For the Integrated Development Environment (IDE) user context, it allows the evaluation of whether all the necessary information is captured.
- For the IDE developer context, it allows the improvement of the tools to cover any missing information.

In order to evaluate our exemplary development environments, we must first identify which attributes are captured in which development artifact categories. This mapping is investigated in this section.

We proceed to restructure the attributes of the Safety Mechanism Pattern to facilitate the subsequent mapping onto the development artifact types which capture them and highlight the elements necessary for the correct and complete documentation of Safety Mechanism Pattern for reuse. The artifact types required to as introduced in section 4.6.1 are:

- Requirements (*REQ*): Requirements artifacts capture the purpose, function and rationale of the Safety Mechanism and provide a means to validate the Mechanism's functionality.

- **Specifications (*SPEC*):** Specification artifacts drive the implementation of the Safety Mechanism while also capturing platform or technology specific information as well as any related design decisions. They are used to verify the Implementation.
- **Implementation (*IMPL*):** Depending on the development environment used, Implementation artifacts can capture information that goes beyond their immediate purpose, for example, boundary conditions or platform instantiation information.
- **Argumentation (*ARG*):** Next to Safety Cases, Argumentation artifacts also include tests, analyses and their results. They provide an overarching structure that binds the elements of a safety mechanism together.

The artifacts from these categories can then be collected for each individual Safety Mechanism into a Library element or container (*SMLC*) that captures all the relevant information and provides the mechanism allowing the reuse of the pattern.

As a reminder, our aim is to capture all necessary information for pattern reuse, not only such information as can be captured in the modeling artifacts. Hence, the role of the Safety Mechanism Library Container (*SMLC*) is not limited to merely housing the artifacts from the other categories. More importantly, any attribute information – which is relevant to the function of the Safety Mechanism, but cannot be captured explicitly in the models provided by the development environment – will have to be documented in the classical textual form within the library container element. Minimizing this contingent will be a prime directive of our approach and is a direct measure of how adequate the models provided by the used development environment are.

We proceed to map the *SMP* design attributes to the artifact categories that do or could potentially cover them, listed in order of importance:

- **Name (*N*):** The name of a pattern is captured primarily in the implementation artifacts, but also in the corresponding argumentation package. (*IMPL*, *ARG*)
- **Rationale (*R*):** The rationale or intent is best captured in the requirements artifacts but can be supported by argumentation elements if they can contain a structured safety case description. (*REQ*, *ARG*)
- **Motivation (*M*):** A structured safety case description is most suitable to capture how a design pattern will solve a problem and hence the motivation to use it. This can be supported by requirements artifacts or even supplanted by them, if no structured safety case view is provided by the development environment and its models. (*ARG*, *REQ*)
- **Applicability (*A*):** A description of the situation to which the Safety Mechanism should be applied, as well as counterexamples, should be covered by

specification artifacts, but can also be supported by implementation artifacts if the models provide for it. (*SPEC, IMPL*)

- *Structure (S)*: The representation of the elements of a pattern and their relationship (preferably in graphical form) is captured into the implementation description, as well as the specification, depending on the models used. (*SPEC, IMPL*)
- *Participants (P)*: Specification and implementation artifacts support the container element in capturing all the objects involved in the function of the Safety Mechanism. (*SPEC, IMPL, SMLC*)
- *Collaborations (CL)*: How the pattern's various elements collaborate to achieve the stated goal is captured best by a structured safety case, with potential support provided by requirements and specification artifacts capturing the intended behavior. (*ARG, REQ, SPEC*)
- *Consequences (CN)*: Information pertaining to the instantiation and deployment of the pattern, as well as to the impact it may have on the system, is captured in specification and implementation artifacts with additional support and structure provided by argumentation elements. (*SPEC, IMPL, ARG, SMLC*)
- *Implementation (IM)*: Descriptions of the implementation itself, but also its possible failure modes or dysfunctional behavior as well as any constraints or possible pitfalls, is captured by implementation and specification artifacts, supported by argumentation. (*IMPL, SPEC, ARG*)
- *Implications (IC)*: The impact of applying the pattern on non-functional requirements can be captured in system and behavior specification models, with additional support provided by argumentation. (*SPEC, ARG*)
- *Usage Classification (UC)*: The kind of fault/error handling (detection, avoidance, mitigation) the pattern provides should be part of the specification. (*SPEC*)
- *Example (E)*: Providing an understandable representation of a known use of the Safety Mechanism is an important aspect, which can be presented nicely by a structured safety case, if the development environment has such a provision. Alternatively, it can be aided by the specification and implementation artifacts, supported by argumentation, if the models support it. (*ARG, SPEC, IMPL*)
- *Related Patterns*: Information pertaining to the related patterns can be captured in the requirement, specification or implementation artifacts, but will ultimately more likely have to be uniformly handled by the mechanism container. (*SMLC*)

In which artifact, in what form and to what degree an attribute is captured, depends entirely on the development environment used. That is, if the development environment and the models one uses can express an attribute, then it is also possible to capture it in the description of a reusable Safety Mechanism Pattern (*SMP*). The structure and

implementation of the Safety Mechanism Library Container holding all this information together and facilitating reuse, is also entirely dependent on the development environment. The richer and more expressive a development environment and its models, the more attributes one can capture and the better one can capture them. That, however, is not without a caveat; if the models used in an environment were developed independently and used without a holistic Systems Engineering analysis, the chances increase that too much information or even conflicting information be captured in competing artifacts. For instance, specification information captured in a Requirements Engineering tool, e.g. DOORS, may conflict with design specification information captured in a design tool, e.g. UML/SysML using Enterprise Architect. Despite the fact that having independent sources of information – allowing for double-checking – is viewed positively in safety engineering, it is only beneficial if the overlapping information is consistent and comparable.

To more precisely describe the mapping of attributes to artifact categories, we need to introduce some new operators and definitions. First we assume that the artifact categories (*REQ*, *SPEC*, *IMPL*, *ARG*, *SMLC*), like the *SMP* attributes defined in section 4.7.1, can each be understood to be a tuple, containing the artifact category's name or identifier (*ID*) – a string – and the information content of the encapsulated artifacts (*CONTENT*), which may vary depending on form of expression, degree of detail, level of abstraction, completeness of information and so on. For simplicity we will again consider the content information to be represented as a set. Analogous to the attribute operator *attr*, we introduce the operator *ATTR*, which returns the set of development artifacts which comprehensively and consistently describe a Safety Mechanism Pattern (*SMP*). We similarly introduce the artifact operators *CONTENT* and *DECOMP*, analogous to their attribute operator counterparts *content* and *decomp*, respectively.

Revisiting the operations of Safety Mechanism Patterns provided in the previous section, and restructuring to express the assignments we just performed yields a mapping of pattern attributes to the development artifacts which should contain them. For simplicity, Equation 15 gives the mapping for an atomic pattern. By applying the operations defined in section 4.7.2, it is possible to extend the mapping to composite patterns.

To reap the most benefits from the approach, two rules should be followed when applying this definition to a development platform:

- Attributes should be captured in one artifact category, and if possible, in only one artifact. This is a measure of the clarity of the development environment with respect to capturing Safety Mechanism Pattern attributes.
- The residual information content, captured in the *SMLC* category, should be minimized, and if possible, completely eliminated. This is a measure of the completeness of the development environment with respect to capturing Safety Mechanism Pattern attributes.

$$\forall SMP: attr(SMP) = (N, R, M, A, S, P, CL, CN, IM, IC, UC, E, RP) \Leftrightarrow$$

$$(ATTR(SMP) = (REQ, SPEC, IMPL, ARG, SMLC) \wedge \psi \wedge \varphi)$$

where:

$$\psi \Leftrightarrow (REQ = (R_{REQ}, M_{REQ}, CL_{REQ})) \wedge$$

$$(SPEC = (A_{SPEC}, S_{SPEC}, P_{SPEC}, CL_{SPEC}, CN_{SPEC}, IM_{SPEC}, IC_{SPEC}, UC_{SPEC}, E_{SPEC})) \wedge$$

$$(IMPL = (N_{IMPL}, A_{IMPL}, S_{IMPL}, P_{IMPL}, CN_{IMPL}, IM_{IMPL}, E_{IMPL})) \wedge$$

$$(ARG = (N_{ARG}, R_{ARG}, M_{ARG}, CL_{ARG}, CN_{ARG}, IM_{ARG}, IC_{ARG}, E_{ARG})) \wedge$$

$$\left(\begin{array}{l} SMLC = (N_{SMLC}, R_{SMLC}, M_{SMLC}, \\ A_{SMLC}, S_{SMLC}, P_{SMLC}, CL_{SMLC}, CN_{SMLC}, IM_{SMLC}, IC_{SMLC}, UC_{SMLC}, E_{SMLC}, RP_{SMLC}) \end{array} \right)$$

and,

for all $X \in \{N, R, M, A, S, P, CL, CN, IM, IC, UC, E\}$,

$$\varphi \Leftrightarrow \left(\begin{array}{l} content(\Pi_X(decomp(SMP))) = \\ CONTENT(\Pi_{X_{REQ}}(DECOMP(SMP))) \cup \\ CONTENT(\Pi_{X_{SPEC}}(DECOMP(SMP))) \cup \\ CONTENT(\Pi_{X_{IMPL}}(DECOMP(SMP))) \cup \\ CONTENT(\Pi_{X_{ARG}}(DECOMP(SMP))) \cup \\ CONTENT(\Pi_{X_{SMLC}}(DECOMP(SMP))) \end{array} \right)$$

Equation 15: Mapping Development Artifacts to the corresponding Pattern Attributes

The formalization developed in this section will guide the instantiations in the next two chapters.

The next section shows how the approach can be evaluated after instantiation in a development environment.

4.9 Evaluating the Development Environments

The evaluation of development environments solely along objective criteria is not possible in our context, as the captured information content for each attribute differs depending on the tools used, how the tools are integrated into the development environment, and how the users perceive the tool.

4.9.1 Evaluation Scale for Instantiation Case Studies

We will use a scale of 0 to 1 in quarter-point steps to evaluate if, and how well, an attribute is captured in the development environment, according to the differentiation provided in Table 6.

Evaluation	Description
0.00	The attribute cannot be captured in the development environment.
0.25	The development environment has no contingency to capture the attribute, but it can be partially covered by abusing the tool, or is inadvertently captured while documenting another attribute, e.g., capturing motivation, rationale and applicability in a single Requirements Engineering tool entry.
0.50	The development environment expressly captures the attribute, but using an inadequate mechanism, e.g., a text entry for describing complex structure or component relations. Or the attribute is not captured, despite the development environment having the capability to capture it.
0.75	The development environment has a dedicated mechanism for capturing the attribute, but it misses some relevant information, or is not well integrated with other attribute descriptions, or has other (to be documented) problems affecting the quality of the attribute documentation.
1.00	The development environment captures the attribute well. All necessary information is documented.

Table 6: Evaluation Scale for Pattern Library Evaluators, reproduced from [184]

We will apply this scale to our case study evaluations in the next three chapters.

4.9.2 Threats to the validity of the evaluations in the next two chapters

In this section, we reflect on the validity of the evaluations conducted in the PS (Prescriptive Study) phase of our research, using the established framework of threats to validity: conclusion, internal, construct, and external validity. We emphasize that the evaluations presented are of a limited academic nature, aimed at assessing the feasibility and utility of the approach in its initial instantiation, rather than producing generalizable results.

4.9.2.1 Conclusion Validity

Conclusion validity concerns whether the observed relationship — namely, the improvement in task support by modeling environments — is statistically and methodologically sound. In our study, evaluations were conducted using a consistent,

clearly defined scoring scheme based on our catalogue of 13 attributes. Our evaluation scale may introduce subjectivity; however, efforts were made to mitigate this through:

- Clearly defined evaluation criteria.
- Collaborative scoring by a team of researchers to reduce individual bias.
- Inclusion of independent tool users (e.g., research students) in the evaluations.

These steps help ensure that the conclusions drawn regarding improvements in tool support are grounded in a consistent and collaboratively verified evaluation process.

4.9.2.2 Internal Validity

Internal validity relates to whether the improvements observed can be attributed to the targeted enhancements introduced, rather than to other confounding factors. Several measures were taken to strengthen internal validity:

- The same scoring scheme and evaluation process were used before and after the enhancements.
- Evaluations were conducted by the same team (with additional independent users) to maintain consistency.
- The enhancements themselves were derived directly from shortcomings identified in the first evaluation round, strengthening the causal link between intervention and outcome.

As the evaluations were performed in controlled research settings with minimal external interference, we consider the threat to internal validity to be low.

4.9.2.3 Construct Validity

Construct validity addresses whether the evaluations truly measure what they intend to – namely, the capability of the evaluated modeling environments to support the safety mechanism pattern capturing and reuse. Potential concerns include the subjectivity of the scoring and the assumption of equal weight across the 13 attributes. We acknowledge these issues and addressed them explicitly:

- **Subjectivity of the Evaluation Scale:** While the scale is inherently subjective, our aim was not to perform a definitive comparative evaluation of the tools, but rather to assess the instantiation of our approach and its localized impact.

- **Parity of Attribute Weights:** Equal weighting of attributes may not perfectly reflect their real-world importance. However, since all attributes are necessary, and improvements were observed across all of them — not concentrated in any specific subset — we believe this did not meaningfully affect the interpretation of results. Further refinement of weighting (e.g., via expert surveys or impact analyses) was outside the scope of this phase but may be relevant in future applications.

4.9.2.4 External Validity

External validity concerns the generalizability of the findings. In this PS phase, we make no claims of generalizability of the immediate applied practical benefits beyond the specific settings in which the evaluations were conducted. The purpose of this phase was to demonstrate that localized improvements in reuse could be achieved through a targeted instantiation of our approach. This served as a motivation for the subsequent DS-II phase, covered in chapter 7, where a longitudinal, real-world case study and a systematic survey will address broader questions of generalizability and impact at scale.

4.10 Targeted Advantages and Benefits of our Approach

Contrary to previous work, our approach provides an explicit model-based representation of technical system solutions within the context of their usage and the engineering problems they solve, which can be instantiated into any model-based development environment, offering several advantages [195]:

- 1) **Improved problem characterization:** It enables a more precise and contextual understanding of the solution space than typical textual design pattern templates.
- 2) **Natural model-based representation:** It supports direct embedding of reuse-critical information into the models used in practice.
- 3) **Increased clarity and rationale:** It provides practitioners with structured guidance on selecting, tailoring, and applying reusable solutions.
- 4) **Integrated argumentation and architecture:** It aligns safety or dependability arguments with architectural structures, improving traceability and understandability.
- 5) **Support for co-evolution:** It facilitates the synchronized development of safety-related and systems engineering artifacts.

When instantiated into a real development environment, our approach provides both immediate and long-term practical benefits:

- **Guided capturing of reuse-critical solutions:** Practitioners no longer need to approach the reuse problem ad-hoc. By pre-defining what information is needed, our method ensures that all necessary contextual and structural data is captured, regardless of the tooling in use.

Claim 1: The approach enables systematic capturing of reuse-critical information for technical solutions, independently of any specific tool-suite.

- **Assessment and improvement of existing tools:** Defining the required information upfront provides a basis for evaluating whether a tool or framework supports all relevant attributes. This assessment highlights gaps and prioritizes tool extension.

Claim 2: The approach enables systematic evaluation of how well current tools capture reuse-critical attributes.

Claim 3: The approach enables structured and prioritized extension of tools to improve reuse capabilities.

- **Tool and framework selection:** The approach serves as a benchmark for identifying more suitable modeling frameworks and tool-suites. An empirical observation drawn from our case studies supports the idea that more detail-rich, domain-adequate modeling environments enable more complete reuse.

Claim 4: The approach supports informed selection of better-suited tools or modeling frameworks to address specific reuse challenges.

These claims are substantiated across the next three chapters:

- In **Chapter 5**, the approach is instantiated into a domain-independent research tool. We demonstrate a structured reuse library capturing multiple safety mechanisms, and evaluate capturing effectiveness before and after tool enhancements (with a measurable improvement from ~ 42% to 69%).
- In **Chapter 6**, we instantiate the approach into a domain-specific automotive safety modeling framework. The increased domain adequacy enables immediate, high-quality solution capturing for reuse, reaching 89% attribute coverage out-of-the-box, without customization.
- In **Chapter 7**, we generalize the method and demonstrate its application in a large industrial case study over several years. This includes organizational as well as technical enhancements, with empirically demonstrable impact on engineering effectiveness and reuse maturity.

Together, these validations confirm that our approach is not only technically sound but practically transformative, integrating academic methodology with real-world engineering needs.

4.11 Summary

The structured reuse of complex system solutions—exemplified here through safety mechanisms—can be significantly improved by encapsulating all relevant information into a consistent, model-based package. When this information is embedded directly into engineering artifacts and supported by adequate tooling, it enables more effective and robust reuse. To achieve this, automation, tool integration, and domain-adequate modeling support are critical.

In this chapter, we introduced the MModel-based Design PAttern Library (MODPAL) approach and positioned it in the context of existing research. We began by defining the research methodology and outlining the practical challenges faced in current reuse practices. We then presented the conceptual basis of our approach, developed its artifact reference structure, and detailed the associated attribute definitions. Guidelines were provided to support instantiation into real-world tool environments, and an evaluation framework was introduced to assess reuse maturity and effectiveness.

We also articulated the targeted advantages and practical benefits of our approach, identifying both immediate and long-term impacts on engineering practice, tool enhancement, and organizational decision-making.

The following two chapters validate this approach through instantiation into two distinct development environments: AutoFOCUS3, a domain-independent research platform, and SAFE, a domain-specific modeling framework for automotive safety. In Chapter 7, the approach is generalized and applied to a longitudinal industrial case study focused on reuse of vehicle system solutions within a large automotive organization.

5 Instantiation in AutoFOCUS3 – a Research CASE Tool

This chapter presents the first evaluation of the approach, carried out within the domain-independent research CASE tool AutoFOCUS3²⁹ (AF3). The work presented here is based on research activities leading to specifications carried out within the integrated development environment for the ITEA2 SAFE³⁰ / EUROSTARS Safe-E project, with proof-of-concept implementations in AF3. The results were well received by the reviewing OEM advisory board and industry experts of the SAFE project and published in publicly available deliverables and conference proceedings, as presented in section 1.7.

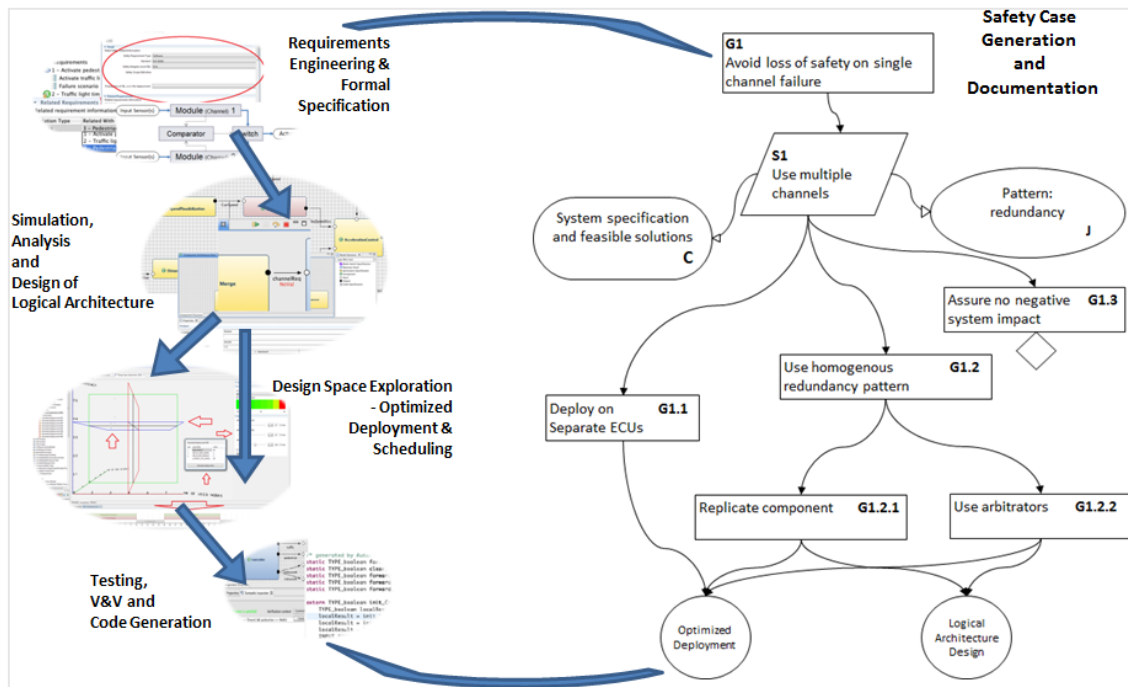


Figure 15: Illustrative overview of proof-of-concept implementation for Pattern Library Approach in AF3 [21]

The evaluation leverages preexisting capabilities in AF3 – such as logical architecture description, model-based requirements engineering, model checking, optimized deployments, and many others – to provide the artifacts for the safety mechanism libraries, as illustratively shown in Figure 15.

To evaluate the instantiation, we first introduce AF3, followed by a description of the integration of the approach into the tool-suite, and concluded with an explanation of the usage workflow.

²⁹ <http://af3.fortiss.org>

³⁰ <http://www.safe-project.eu>

The implementation results presented in this chapter were previously published by the author in [196] and [197] and later in more detail in a Master’s thesis supervised by the author [198].

5.1 Introduction to AUTOFOCUS3

AutoFOCUS3 (AF3) is a research computer-aided software engineering (CASE) tool, which allows modeling and validating concurrent, reactive, distributed, timed systems on the basis of a formal semantics [199]. Most importantly, AF3 provides a graphical user interface that facilitates the modeling of embedded systems in different layers of abstraction while supporting different views on the system model, including the requirements-, component- and platform- view essential for the description of complex systems. This support for views enables applying paradigms such as “separation-of-concerns” – different views for different stake-holders/concerns – as well as “divide-and-conquer” – different abstraction views allowing the decomposition of engineering problems to facilitate solution – and is in line with the concepts introduced in section 2.5.1. This introductory section is based on publicly available descriptions of the AF3 suite, originating on the AF3 website and in papers published by AF3 development team members.

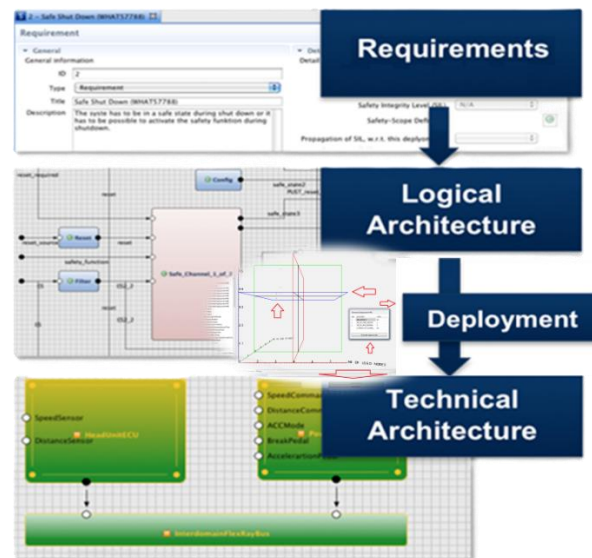


Figure 16: Modeling views in AF3, adapted from [200]

As explained in [201], AUTOFOCUS 3 uses a message-based, discrete-time communication scheme as its core semantic model. In AF3, systems are modeled using networks of components communicating via messages. Message exchange is synchronized with respect to a global, discrete time-base. This computational model supports a high degree of modularity. The discrete time base abstracts from implementation details such as detailed timing or communication mechanisms because the usage of timing information below that chosen granularity of observable discrete

clock ticks is avoided. The message-based time-synchronous communication model caters for both periodic and sporadic communication behavior.

Figure 16 shows the different views supporting seamless model-based development in AF3. An AUTOFOCUS 3 system model is divided into several models providing different levels of abstraction, while supporting different views on the system model [201]:

- The **Requirements** Specification and Analysis View provides for specification, documentation, and analysis of the requirements of a system.
- The **Logical Architecture** View of a system is defined by means of components communicating via message passing through directed, typed channels, using a clearly defined model of computation. Components can directly implement behavior or consist of other components that do so.
- The **Technical Architecture** View describes a hardware topology that is composed of hardware units, e.g. CPUs of a multi-core board, hardware ports (sensors or actuators), busses and a shared-memory component.

Furthermore, AF3 supports so-called integrated or weaving models as introduced in section 4.6.2, which hold information from different levels of abstraction and thereby connect them. The **Deployment** view is one example of such an integrated model, which maps elements from the **Logical Architecture** to elements of the **Technical Architecture**, providing traceability between modeling artifacts.

5.2 Capturing Safety Mechanism Patterns in AF3

Given its flexibility, expressiveness and support for multiple well-integrated modeling views, AF3 is well-suited to support patterns reuse, and consequently there were attempts – predating the work presented here – at integrating support for pattern reuse into AF3. These earlier works were smaller in scope and had different goals. Namely, they preliminary targeted the investigation of the practicality and usefulness of such a capability. As with its counterparts in practical settings, the most useful of these attempts focused on the obvious elements to be reused, in this case the logical architecture of redundancy mechanisms, and were not interested in any other elements. This in essence mirrors the fundamental problem our approach is trying to solve, namely the focus on the central reusable artifact – which does not tell the whole story or allow for a correct reuse in safety-critical context – to the detriment of all else.

This section will give a quick overview of the previous attempt and its drawbacks, as well as the limitations it highlighted in AF3 in the context of Safety Mechanism Pattern reuse. This is followed by a preliminary evaluation of AF3’s usefulness for capturing SMP, as defined in the previous chapter. This preliminary evaluation guided the subsequent activities to introduce support for safety mechanism reuse in the AF3 suite, as seen next.

5.2.1 Previous Pattern Support in AF3

Earlier attempts at reusing architectural patterns in AF3 existed, such as [202], in which the authors demonstrated how Homogenous Duplex Redundancy (HDR) and Triple Modular Redundancy (TMR) could be integrated as patterns into AF3. The approach works by allowing the user to right-click anywhere in the Logical Architecture view of an open pre-existing AF3 model, and select one of the two patterns implemented exemplarily to use. This then copied empty logical components – essentially a skeleton conforming to the pattern – into the open AF3 model. The user could then copy his actual components into the skeleton provided by the feature.

Before this work, AF3 had no pattern capability. The approach was successful in demonstrating that design patterns can be supported in AF3 and that they are useful. Practitioners who experimented with the feature gave positive feedback and voiced their interest in having the functionality expanded.

The feature was however mainly limited by the capabilities of AF3 itself at the time and hence that is the primary source of the major drawbacks:

- The approach is limited to copying logical component architectural snippets into existing models, which then have to be filled by the user with their own functional components.
- Updates to the existing patterns could only be carried out (at code level) by the developer.
- Normal users of AF3 could not expand the feature with other patterns.
- The approach was limited to logical components. All other artifacts necessary to support the reuse of the design pattern and capture the necessary attributes could not be included into the reusable feature.

AF3 had no support for safety case argumentation. More importantly, it also had no support for a reusable pattern library. The patterns featured here had to be manually hard-coded into the tool, rendering this approach unusable to a normal tool user.

The lack of ability to integrate other artifacts generated in AF3 to support the correct reuse of design patterns in a safety-critical context was also a particularly relevant drawback. Indeed, the patterns implemented exemplarily here would have especially benefitted from this support, because they are both variations of the N-Homogenous Redundancy design pattern. Limiting their consideration to logical components greatly reduces the usefulness of the captured pattern, as such descriptions are almost pointless, unless one can maintain a link to a deployment strategy that ensures the redundant components are deployed on different hardware resources.

The analysis of this work shaped our future activities regarding the introduction of support for reusable pattern libraries in AF3, as will be seen in the following sections. But first, we will evaluate how AF3 initially documented Safety Mechanism Patterns with its feature set at the time.

5.2.2 First evaluation of AF3 for capturing Safety Mechanism Patterns

It is important to note that the previous experimental pattern implementations – such as the one described in the previous subsection – were carried out on a development branch and were not included into the standard functional repertoire of AF3 due to their limited usability. Hence, AF3 officially had no support for patterns whatsoever. Nevertheless, we will assume the previous implementation was functional and, on that basis, evaluate how AF3 could have handled the documentation of Safety Mechanism Patterns. This is done by analyzing which of the attributes – defined in section 4.5.1 – can be captured into the tool and to which extent, rated according to the evaluation scheme introduced in section 4.9.

Table 7: Evaluation of SMP coverage in AF3 before implementation of our approach, reproduced from [184]

Attribute	Symbol	Evaluation Remarks	Score
Name	N	Captured in the feature.	1.00
Rationale	R	Could be captured by AF3, but was not part of the feature.	0.50
Motivation	M	Could be captured by AF3 along with other requirements, but was not part of the feature.	0.25
Applicability	A	Partially captured in the reusable Logical Architecture but needed further details.	0.50
Structure	S	Captured for the most part in the reusable Logical Architecture.	0.75
Participants	P	Implicitly captured for the most part in the structure of the reusable Logical Architecture, but would have benefitted from increased elucidation.	0.50
Collaborations	CL	Implicitly partially captured in the structure of the reusable Logical Architecture, but would have benefitted greatly from argumentation descriptions.	0.25
Consequences	CN	The usage was limited to copying Logical Architectures with predefined instantiation rules, while all other attributes were missing.	0.25
Implementation	IM	The components of the Logical Architecture captured most of the implementation information, except the deployment rules.	0.75
Implications	IC	This part went completely missing, some of the impact on execution time and resources were nevertheless inadvertently captured by AF3's semantics.	0.25

Usage Classification	UC	This part is not handled in AF3	0.00	
Example	E	The graphical interface of AF3 makes the captured Logical Architecture fairly easy to understand, so we can consider it to be an example. Some additional description would have facilitated the understandability of the pattern even more.	0.50	
Related Patterns	RP	Not handled in AF3.	0.00	
			Total Tally (out of 13)	5.50
			Percentage	42.3%

It is important to note that the evaluation is only concerned with establishing a baseline for AF3 capabilities before the implementation of our approach and should not be viewed as an evaluation of earlier results, which had different goals anyhow.

5.2.3 Adapting AF3 to the pattern library approach

Based on the previous evaluation, AF3 was analyzed for reuse enhancement, leading to the conclusion that the AF3 suite already had good potential to capture Safety Mechanism Patterns and support their reuse. To achieve this potential AF3 had to be extended to include two missing features. First, support for assurance argumentation and safety cases – as discussed in detail in section 2.4 – was added. We then added a pattern library mechanism allowing the bundling of development artifacts and argumentation elements into a holistic approach for the correct reuse of SMP. This section will discuss the introduction of these two features to AF3.

5.2.3.1 Addition of safety case support

As detailed in section 2.4.1, recent years have seen a marked shift in the regulatory approach to ensuring system safety – away from a mere demonstration of compliance with prescriptive safety codes and standards – and towards shifting the responsibility back onto the developers and operators of the system, who should construct and present well-reasoned arguments demonstrating that their systems achieve acceptable levels of safety.

A safety case is only as good as the information it captures, as discussed in section 2.4.4. The integration of the product functional safety case into the tool suite containing the actual development artifacts would greatly increase the argumentation strength. As AF3 already provided seamless model-based development capabilities, extending it to support argumentation and safety cases was the logical next step.

The first AF3 extensions introduced support modular GSN-based safety case argumentation [200], in which our team made the following main contributions:

- The introduction of a model-based development approach integrating views for the construction of a system and the argumentation about its functional safety.
- The provision of modular construction and argumentation principles within these views, reflecting current requirements from the safety domain.
- The documentation and application of design decision, used simultaneously for the construction of a system as well as the argumentation about its functional safety.

5.2.3.2 Addition of Safety Mechanism Pattern-library

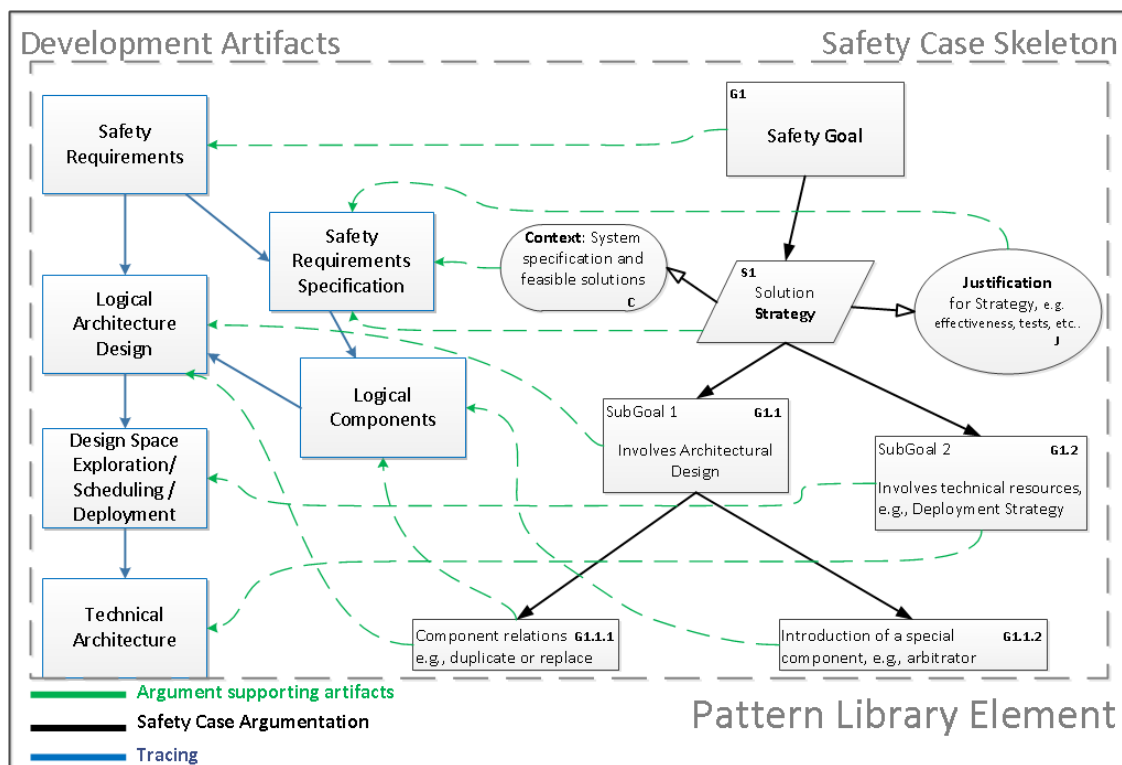


Figure 17: Illustrative overview of candidate pattern library elements in AF3, reproduced from [21]

To begin the implementation, we had to identify which AF3 artifacts would be parts of the pattern library element. It is important to note that we did not wish to convert AF3 into a pattern documentation and reuse tool suite, but rather aimed at leveraging the many preexisting features of AF3 for seamless model-based development to showcase the approach and serve as a proof-of-concept. Its domain independence, along with its ease of use and expansible feature set, is one of AF3's strengths. With the addition of an integrated Safety Case view, the potential for a high coverage of Safety Mechanism Pattern attributes was guaranteed.

An analysis of the AF3 feature set led us to identify potential artifact types and categories, as well as their links, which could become elements of a pattern library. Expanding on the generic structure given in Figure 11, we replaced the development artifact categories on the left-hand side with artifacts from AF3, demonstrating what a pattern library in AF3 could look like, shown in Figure 17. This defined our pre-selection of mandatory development and argumentation artifacts, necessary for a correct description of a pattern and thus shaped our activities for the implementation of the approach, as explained in detail in the following section.

5.3 Implementation of our approach in AF3 and resulting Guidelines

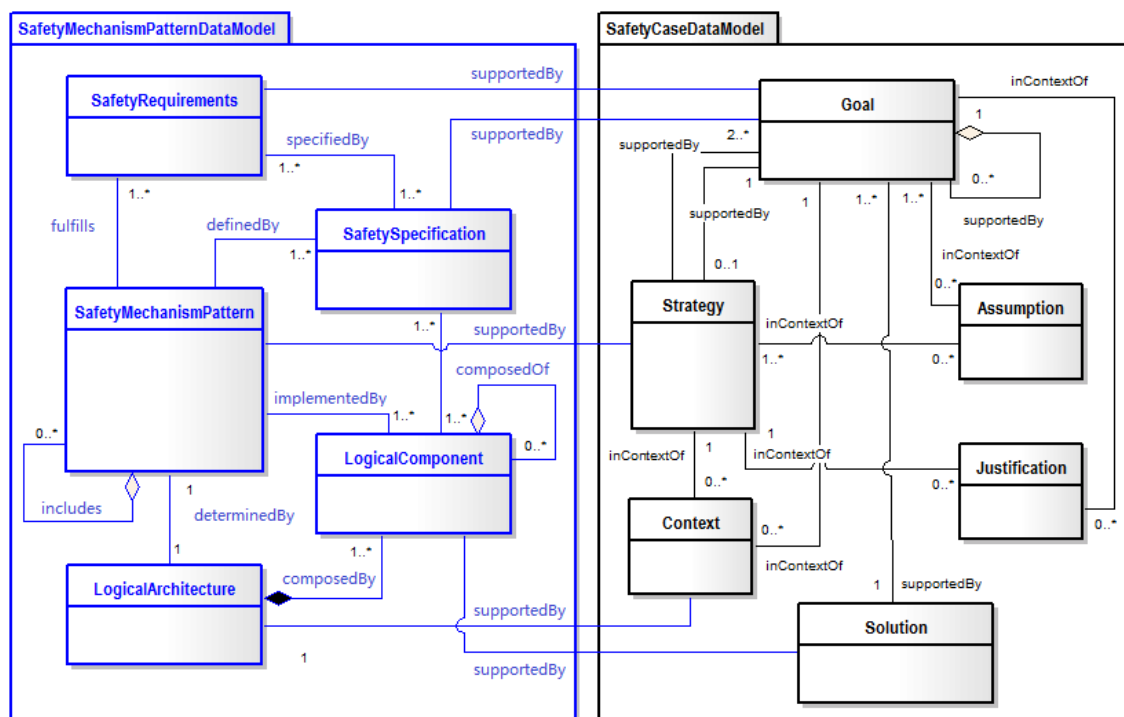


Figure 18: AF3 Pattern Library Element Data Model, reproduced from [196]

Having identified candidate AF3 artifacts, as discussed in the previous section, we proceed in this section to focus on the structure and usage of AF3 elements forming a library of reusable patterns, including the data models, library mechanism, and workflow required to support seamless integration into the AF3 tool suite.

We designed a pattern library mechanism, which structures the development artifacts and weaves them together with argumentation elements to support pattern capturing and reuse. Figure 18 shows the structure model of the pattern library mechanism element in AF3, comprised of development artifacts on the left-hand side, and the supporting safety case on the right-hand side, needed to represent a Safety Mechanism. As discussed in section 5.2.2, the safety case itself was recently added by the AF3 team. The structure model of the Safety Mechanism Pattern Library and its weaving of development and

argumentation artifacts to support pattern capturing and reuse is the author's contribution, shown in blue in the figure. The next sections discuss the implementation in detail.

5.3.1 Structure Model Elements

Figure 19 provides a closer look at the left-hand side of Figure 18, detailing the development artifacts of the pattern library. The elements of the data model seen in the figure are:

- **SafetyMechanismPattern**

This is a container object collecting information necessary for reuse, which is listed as a Template of attributes and adapted to the context of safety mechanism patterns, as defined in the previous two chapters. This element also provides direct traceability and allows for a quick propagation and adaptation of changes across related patterns, increasing confidence in reuse as well as the maintainability of the library.

- **SafetyRequirement**

This element details the requirement the pattern fulfils, as well as any safety related attributes, e.g., Safety Integrity Level.

- **SafetySpecification**

A concrete description of the safety mechanism's functional behavior and its interaction with the environment, and serves as a basis for the implementation in a logical component. It can be formal, e.g., in AF3 the behavior of safety mechanisms can be captured in state automata.

- **LogicalComponent**

This element captures the implementation or concrete representation of a Safety Mechanism at software level, whose definition includes transfer behavior, e.g., code specification or a state automaton. It can be a composition of logical components which collectively determine the Safety Mechanism's behavior.

- **LogicalArchitecture**

This element defines the context of the safety mechanism pattern, by describing the boundaries / interface of the *LogicalComponent* element implementing the mechanism as well as giving a description of any other components involved.

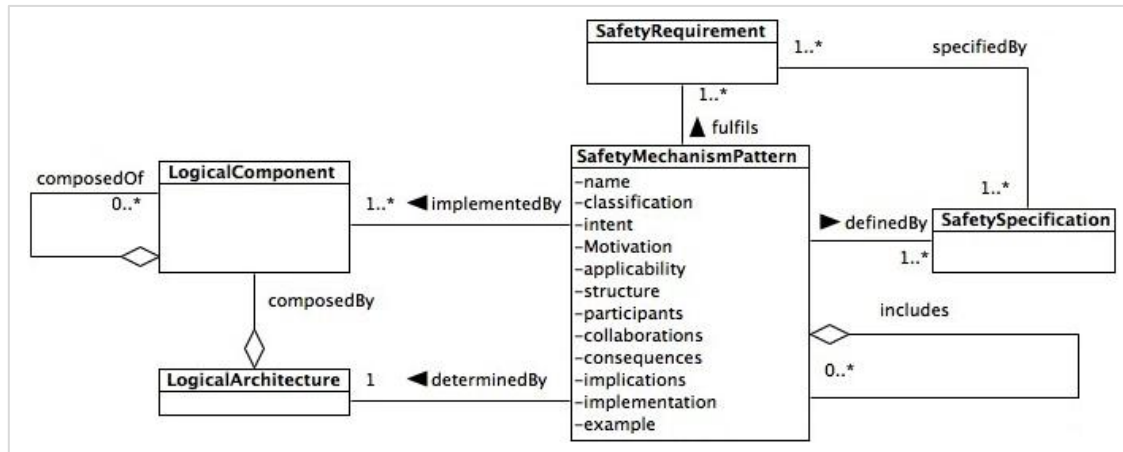


Figure 19: Pattern Library Element: Structure Model for Development Artifacts reproduced from [196]

5.3.2 Structure Model Relations

The relationships between the model's entities are as follows:

- fulfils

The use of the pattern fulfils the stated safety requirement(s).

- specifiedBy

This relation shows that a safety requirement is specified or brought to a more concrete representation by one or more specifications of the implementation, providing the “how” to the “what”. This relation is also bidirectional, because it could be that a solution specification specifies one or more safety requirements.

- definedBy

A Safety Mechanism's behavior is defined by its specification. This relation can also give a description of the architectural context of the Safety Mechanism.

- determinedBy

The Safety Mechanism Pattern is determined by a unique structure or logical architecture.

- implementedBy

The Safety Mechanism is implemented at software level by a logical component.

- composedOf

A logical component can be composed of other logical components or simply direct behavior descriptions, e.g., a code specification or a state automaton.

- composedBy

This relationship shows that a logical architecture or context of a Safety Mechanism is composed by at least two logical components, which interact with the Safety Mechanism's logical component. In its simplest form, one is an input and the other is an output component.

- includes

This relationship indicates that a Safety Mechanism Pattern can include other patterns within its structure. This entails inheritance of the respective safety requirements, safety specification, logical component and the logical architecture of the related or reused Safety Mechanism Pattern. The form and structure of inclusion has to be defined in a case-by-case manner.

5.3.3 Mapping development artifacts to the Safety Case model

Having defined the development artifacts part of the structure model, a quick explanation of their mapping to the corresponding elements in the SafetyCaseModel seen on the right-hand side of Figure 18 is given next.

- SafetyRequirement – supportedby – Goal

The entry point to any reuse problem is a goal or requirement that has to be fulfilled, which was solved before.

- SafetyMechanismPattern – supportedby – Strategy

The information captured in the Safety Mechanism Pattern and the information it contains lays out a plan for solving the problem, i.e. a strategy, and helps explain the relation between the elements of the pattern, most importantly their refinement and decomposition.

- SafetySpecification – supportedby – Goal

A safety specification is mapped to the (sub-)goal entity in the Safety Case data model, as it describes a refinement and decomposition of the higher-level goals towards implementing a solution.

- LogicalComponent – supportedby – Solution

The logical component is the solution for the stated goal.

- LogicalArchitecture – supportedby – Context

The logical architecture provides context to the safety pattern and the logical component and its implementation, without which the pattern is either meaningless or cannot be described concisely.

5.3.4 Guidelines for implementation and usage workflow

Our approach and the implementation support in AF3 envision two roles. A Safety Mechanism Pattern *Developer* defines new patterns and documents them for later reuse by a pattern *User*. We defined a workflow of activities to go with each role, which also affects how the AF3 artifacts are handled and instantiated, as explained in detail in the following subsections.

5.3.4.1 Pattern Developer Role

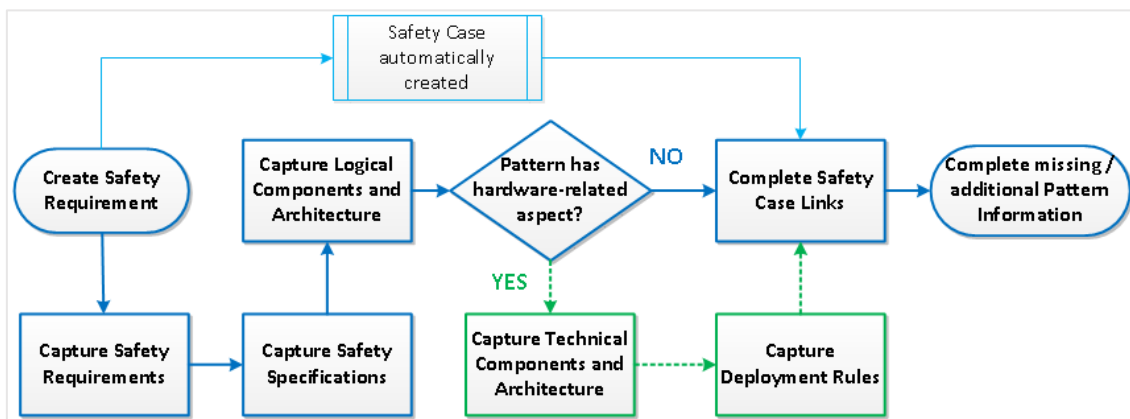


Figure 20: Workflow steps for Safety Mechanism Pattern Developer Role in AF3, reproduced from [203]

Figure 20 shows the steps a developer has to undertake to document a new Safety Mechanism Pattern in AF3, some of which are optional. To support the *Developer* role a comprehensive analysis of the numerous Safety Mechanisms provided in section 3.3 (and seen in Figure 8) was carried out, which identified three archetypes of structures for capturing Safety Mechanisms in AF3. To save a new Safety Mechanism, the *Developer* has to decide which archetype best fits their need. The *Developer* is aided in this task by a wizard, shown in Figure 21, which also assures that library elements cannot be created without the mandatory information.

The individual steps for the *Developer* role are:

1- Create Safety Requirement

By designating a requirement as safety-relevant, a corresponding Safety Case is automatically created.

2- Capture Safety Requirement

The Safety Requirement now has to be filled in with all relevant information. This is done using MIRA – the Model-Based Integrated Requirements Specification and Analysis

Tool in AF3, which also allows to check requirements for consistency and completeness, as well as defining general maturity levels for requirements [204].

3- Capture Safety Specifications

Next, the *Developer* can specify the behavior of the Safety Mechanism formally, e.g., using tables or state automata. This allows for a formal verification of the implemented Safety Mechanism, as well as the automated generation of test-cases, which can then be used to check the system architecture specification against the requirements in a continuous manner, as seen in [205].

4- Capture Logical Components and Architecture

The *Developer* can now begin with the actual implementation of the Safety Mechanism Pattern, by defining logical components and their relations in the **Logical Architecture** view in AF3. Referring to the description of AF3 provided in section 5.1, this includes many aspects – such as typed directed ports – allowing for a detailed and precise documentation of the pattern implementation.

If the pattern is implemented entirely in software and has no hardware aspects, the Developer can skip to Step 7, otherwise we proceed to the next step.

5- Capture Technical Components and Architecture

Next, any hardware related implementation aspects are captured using the **Technical Architecture** view in AF3. This allows the addition of hardware resources, such as computation nodes and communication busses, as well as specifying the properties of these nodes, such as power consumption, memory capacity.

6- Capture Deployment Rules

AF3 allows a precise definition of deployment rules to govern the allocation of logical components to computational resources (nodes). Furthermore, it is possible to use the Design Space Exploration³¹ capability of AF3 to generate deployments and schedules optimized to fulfill multiple criteria, such as worst case execution time, memory constraints, bus loads or power consumption, but also safety constraints, such as ASIL-decomposition and random hardware failure probabilities, as we had demonstrated in [206].

³¹Defined as a process of systematically finding a solution from a set of possible designs, with respect to a set of given constraints.

HDR Pattern Data

Overview

Fill out the following additional data regarding the safety mechanism pattern. Fields marked with * must be filled out in order to proceed with pattern documentation process.

Also known as: Homogeneous Redundancy Pattern, Standby-Spare Pattern, Dynamic Redundancy Pattern, Two-Channel Redundancy Pattern

Classification:* Failure Detection & Failure Handling

Applicability:* This pattern is used to increase the safety and reliability of the system by duplicating the same channel/component to deal with the random faults. The component to be duplicated is the active one, meanwhile the duplicated one remains in standby modus, but it does not mean that it is not computing or processing data. This pattern can be applied to any level of system design.

Consequences:* This pattern is not appropriate for systematic failures, because the implementation of both components/channels is identical. If a failure is caught by the pattern the switch component will switch to one of the faulty channels, and it will lead to a complete system failure. The other negative aspect of this pattern is that information could be lost when a switch between main and redundant channel is carried out, the data in question is related to the already processed input.

Implications:

- Reliability: According to the nature of the fault, it can be classified into two types: random which is handled by this pattern and systematic which cannot be handled here. For each module, the failure rate is equal to the sum of random and systematic failure rate: Formula.
- Safety: The homogeneous duplex pattern includes two design techniques: test by redundant hardware and fault detection and diagnosis (comparator and acceptance test). According to the hardware and software requirements in the standard IEC 61508-2, 3, the recommendations for

Sample: See component architecture

Known uses: The Submarine Command System (SMCS), Intelligent-Dual Sequential Ignition (IDS)

Safety Mechanism Pattern Data

Properties New Safety Mechanism Pattern Project

New Safety Mechanism Pattern process steps:

Step 1: Add Safety Requirements ✓

Step 2: Add Logical Component ✓

Step 3: Add Technical Component ✓

Step 4: Add Deployment ✓

Step 5: Add Safety Case ✓

Step 6: Add Pattern Data ✓ **Save SMP**

Steps completed!

Now press "save SMP" to add the safety mechanism pattern to the library.

Figure 21: AF3 Safety Mechanism Pattern Developer Wizard Interface, reproduced from [196]

7- Complete Safety Case Links

In order to make use of the integrated argumentation capability in AF3, the *Developer* has to build the safety case – arguing the use and functionality of the Safety Mechanism Pattern he is developing – and link all its elements to the corresponding development artifacts in the other AF3 views (**Requirements and Specification – Logical Architecture – Technical Architecture – Deployment**). More information about the integration of Safety Cases in AF3 is provided in [200].

8- Complete missing / additional Pattern Information

Finally, the wizard guides the Developer to provide additional pattern information, such as the pattern classification. As will be seen in the evaluation in section 5.4, although the approach brought significant improvements to AF3's capability of capturing Safety Mechanism Patterns for reuse, some of the required pattern attributes are still nevertheless not satisfactorily captured in AF3's artifacts – especially *implications* and *consequences*. We appended the library mechanism with the capability to add some short textual information to enhance the documentation qualities of these attributes. This is meant as a

stop-gap until AF3 capabilities improve enough to allow capturing the information solely in modeling artifacts. These shortcomings are factored into the subsequent evaluation.

Throughout this process, the *Developer* is guided by the wizard, whose interface can be seen in Figure 21, which makes sure that all mandatory information, necessary for the correct documentation of the Safety Mechanism Pattern attributes, is provided into the library element. The new Safety Mechanism Pattern can now be saved as an element of the pattern library, where it is accessible to other AF3 pattern *Users*.

5.3.4.2 Pattern User Role

For the *User* role, we must differentiate between using the library element in an already existing project, which we will call Augmentation, or in a Greenfield one – that is, a new project that lacks any constraints imposed by prior work. The pattern library deployment wizard offers both options to the *User*, who must select the suitable one based on their needs and use-case.

Greenfield mode

In the Greenfield mode, the *User* is not bound to a pre-existing project or design, but needs a redundant design for their function, application, or system. They wish to start with a comprehensive structure model that includes all the necessary artifacts and then fill in details and add other model elements to it later. The *User* select the pattern, which is then copied with all its relevant artifacts and their relations into the AF3 project. The *User* can now proceed to add information and modeling artifacts as he pleases.

Augmentation mode

In this mode, the *User* is allowed to apply the pattern to pre-existing AF3 modeling elements. For instance, selecting a component then choosing to apply a redundancy pattern to it. This entails more than the mere copying of components and artifacts from the pattern library. The analysis of the structure of the Safety Mechanisms, as well as an analysis of the usage roles and modes, led to the identification of three categories for the instantiation and deployment of the artifacts captured in the library element. Taking the usage scenario into account, these categories define how each artifact in the library element will be added into the development environment. These categories are:

- **Copy:** Artifacts are copied into the environment as is. *SafetyRequirements*, and *SafetySpecification* elements, along with all the *SafetyCaseModel* artifacts, are treated in this manner, as they are clearly defined static elements that do need to be changed upon usage.
- **Instance:** artifacts are to be instantiated upon usage. This may include the input of some parameters by the users. *LogicalComponent* entities are always used in

this manner. *LogicalArchitecture* elements are treated in this manner only if the usage mode is Greenfield.

- **Replace:** Artifacts are replaced by ones existing in the model. This mode applies to *LogicalArchitecture* elements and is obviously only usable when deploying the library element into a preexisting model. E.g., logical components stored in the library element to describe the interaction of the central safety mechanism component with its environment are replaced by existing components from the model, to which the library pattern / safety mechanism is to be applied.

The deployment of the library elements is guided by a wizard, whose function it is, among others, to ensure that library elements are deployed seamlessly. To that end, the wizard prompts the user at each step to link / map the components of the inserted library to their counterparts in the existing model. E.g., a comparator deployment prompts the user to select the two components whose outputs are to be compared, as well as the original requirement, which prompted the deployment of the comparator safety mechanism from the library in the first place. The development process still has to be monitored and approved at key decision points by a human expert. The Augmentation mode is still a work in progress, but it has been fully specified for implementation.

5.3.5 Extended Structure Model

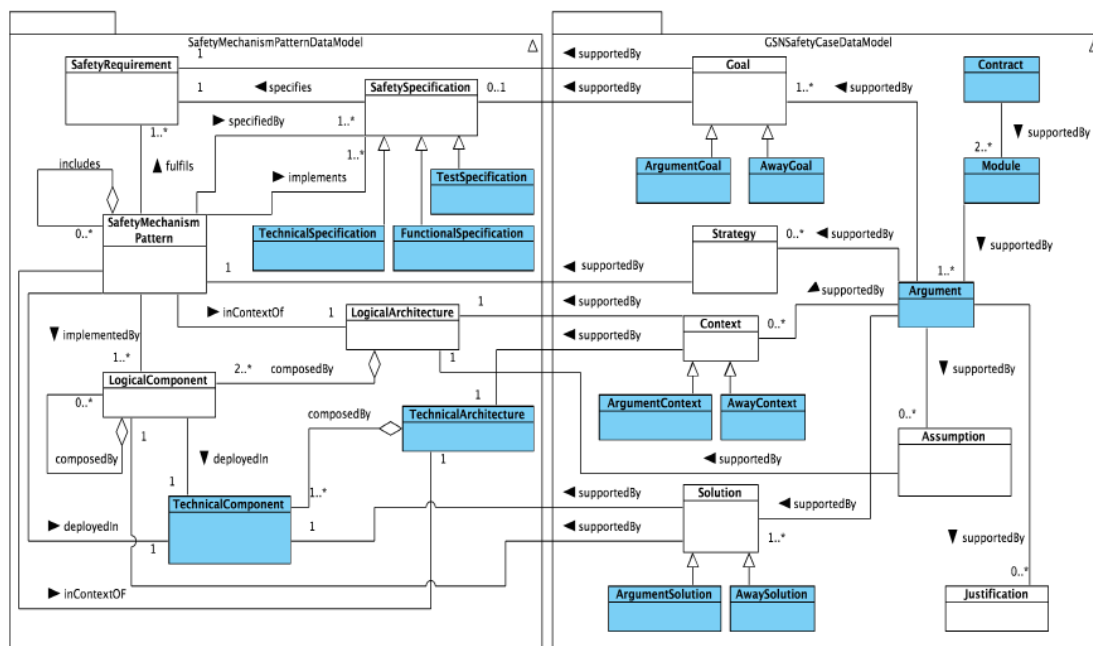


Figure 22: AF3 Pattern Library Extended Data Model, adapted from [196]

The structure model introduced in section 5.3.1 represents the absolute minimum set of artifacts necessary to represent a Safety Mechanism Pattern in AF3. The library structure is, however, incomplete for the *redundancy* example without the *TechnicalArchitecture* view, providing hardware resources, and without capturing the deployment rules, which

determine how the software components are assigned to the hardware resources. This is a new piece of information and gives rise to the need to extend the structure model with optional elements beyond the basic ones previously shown in Figure 18.

To encapsulate more information, such as the deployment rule, or facilitate a refinement of the specification to support the categories provided by ISO26262 in its functional and technical safety concepts, we extended the structure model as shown in Figure 22, with the new elements *TechnicalComponent* and *TechnicalArchitecture*.

The extended model shown here is only meant as an example. Other elements containing information pertaining to the suitability of the safety mechanism, such as tests or analyses, could also be introduced as needed. Also shown is an expanded safety case modeling view, which supports extended GSN elements allowing the construction and reuse of modular safety cases. These implementations are not the focus of this thesis. More details are provided in [198], and Appendix A2.

5.3.6 Implementation examples in AF3

After having in essence analyzed all the Safety Mechanisms discussed in section 3.3 (and seen in Figure 8), we chose to exemplarily implement several of them in AF3, such as: *1oo2Voting*; *Comparison*; *RangeCheck*; and *HomogeneousDuplexRedundancy (HDR)*.

For brevity, this section will be limited to demonstrating *RangeCheck*, a fairly simple pattern, and the *HDR* example, which is the most complex of them, representing a composite pattern, which includes both the *Comparison* and *RangeCheck* patterns. Additional examples are provided in more details in Appendix A3, with the AF3 implementations being covered in detail in [198].

5.3.6.1 RangeCheck

A range checker is a Safety Mechanism designed to detect invalid or corrupted data within a given context, by testing whether the input value is within a pre-specified range. This is a very useful plausibilization and data integrity check. For instance, if the controller of an adaptive cruise control (ACC) system receives false data – due to function or communication errors – signaling that the gap to the car in front has a negative value and applies this value unaltered, this could lead to an unnecessary emergency braking maneuver that could endanger the driver's life and that of other road users.

Range checks can also be used with comparators that offer a tolerance margin, where the comparison turns into a range check of the target value including its tolerance interval.

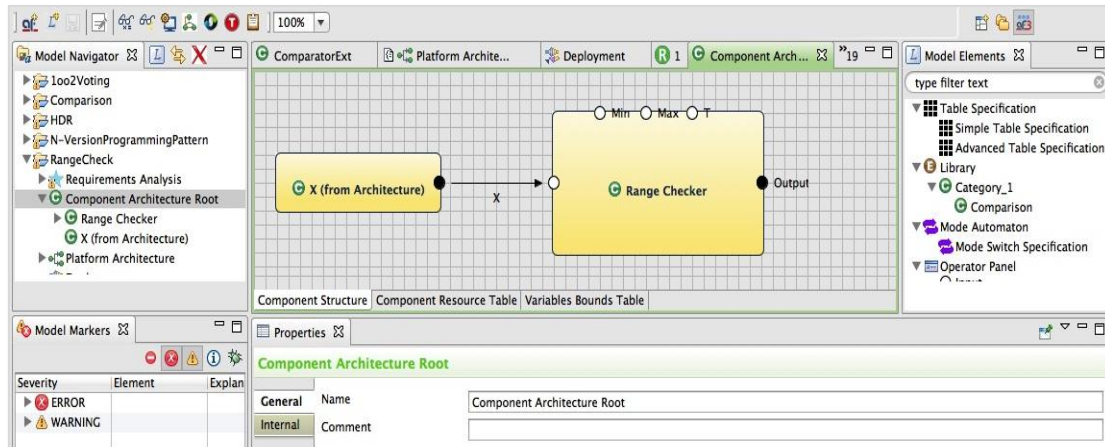


Figure 23: AF3 implementation example – *RangeCheck*: *LogicalArchitecture* [198]

The structure of the *RangeCheck* pattern can be seen in Figure 23, which captures its interaction with the environment using the *LogicalArchitecture* view. The corresponding safety case is shown in Figure 24. Its elements are linked to the supporting development artifacts using the library mechanism.

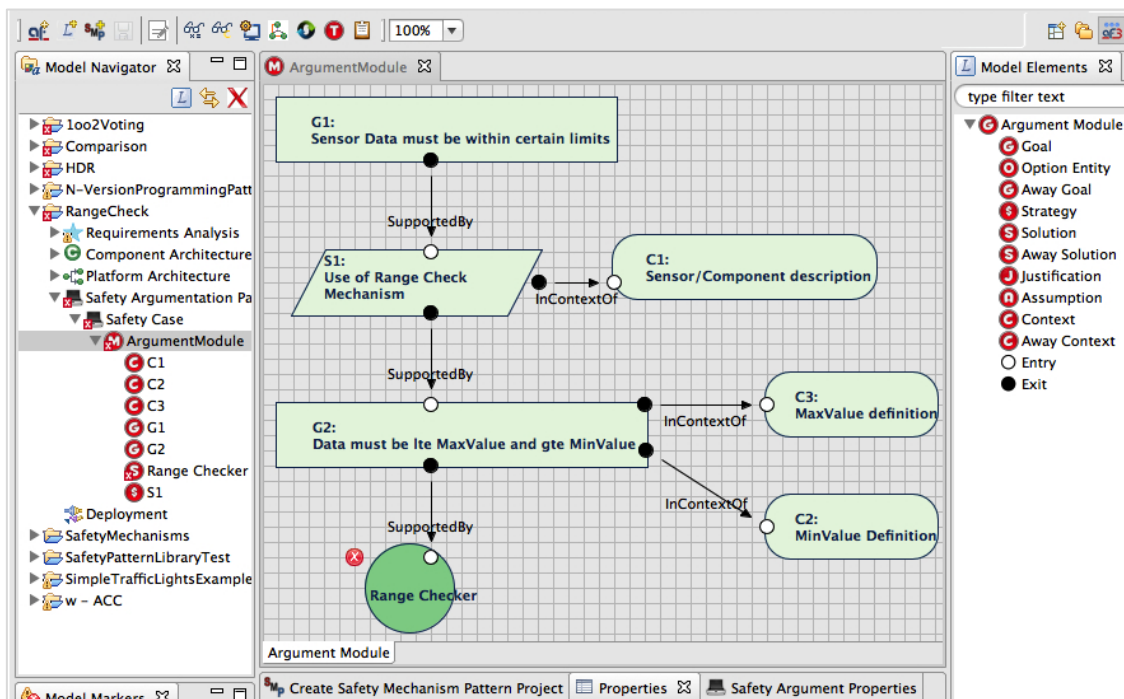


Figure 24: AF3 implementation example – *RangeCheck*: *SafetyCase* [198]

Note how the safety case elements especially support capturing the *Participants* and *Collaboration* attributes, by capturing contextual information such as component description as well as the definition of minimum and maximum values for the range checker, and integrating it into the general argumentation.

5.3.6.2 HomogeneousDuplexRedundancy HDR

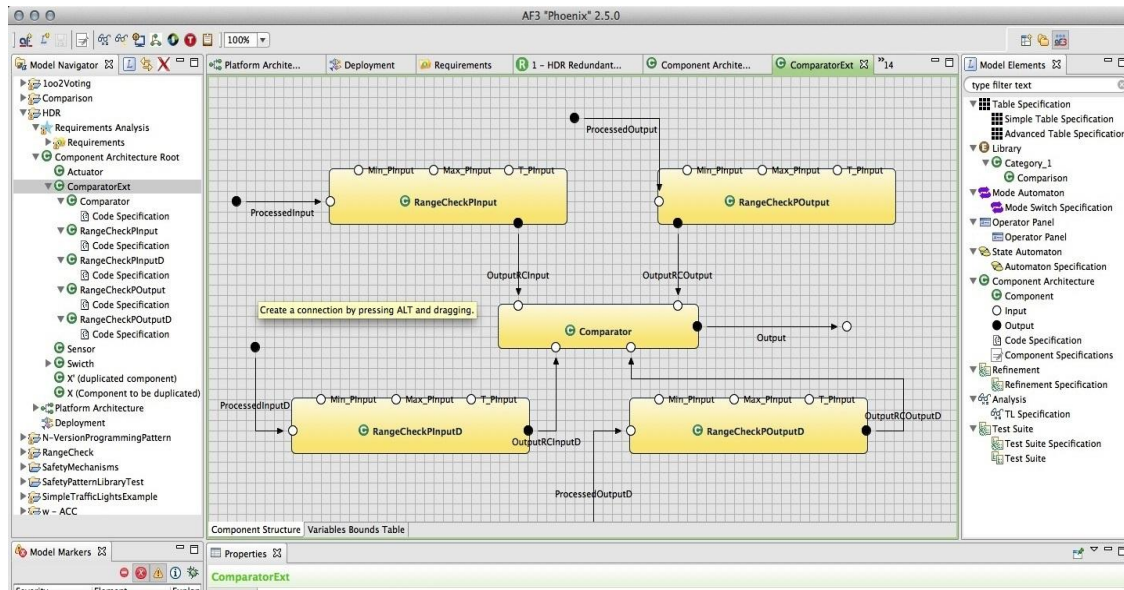


Figure 25: AF3 implementation example – HDR: LogicalComponent, reproduced from [196]

HDR, whose schematic was previously introduced in section 1.1, is a hardware-based Safety Mechanism Pattern designed to counteract random hardware failures – relying on the notion that it is highly unlikely for two identical components to suffer a *random* fault simultaneously. *HDR* is especially useful in the context of embedded safety-critical systems with no fail-safe state, where fault identification is nevertheless possible. It is used to increase a system’s reliability – and subsequently safety – by duplicating the critical component.

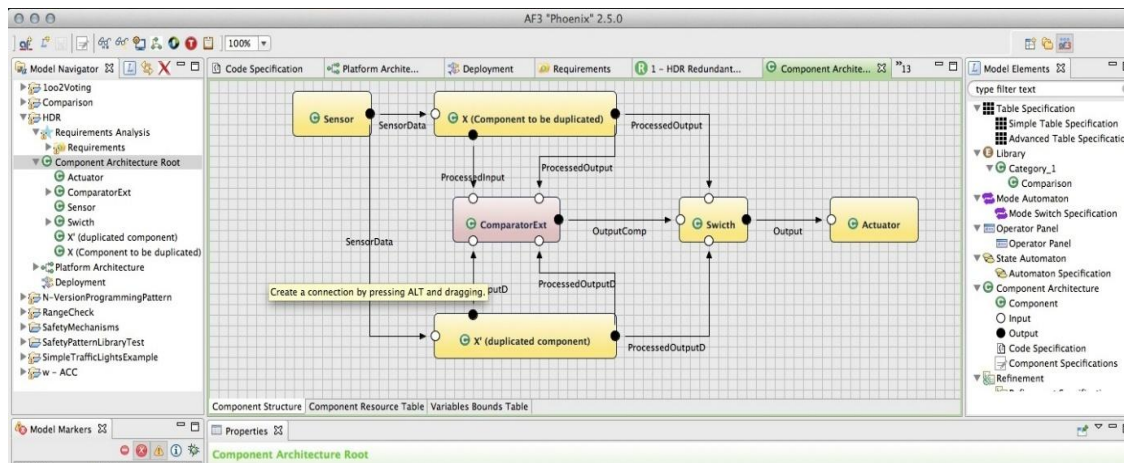


Figure 26: AF3 implementation example – HDR: LogicalArchitecture, reproduced from [196]

The inner workings of the *HDR* pattern can be seen in Figure 25, which captures how the logical component implements its functionality, including its interaction with the included *RangeCheck* pattern logical component.

The interaction of the *HDR* Safety Mechanism with its environment is captured in the *LogicalArchitecture* element shown in Figure 26, with the corresponding *Deployment* view containing the *TechnicalArchitecture* platform view shown in Figure 27.

The ability to save the deployment rules in the pattern is very important, as it allows the correct allocation of logical components to hardware resources, in this case assuring the correct functionality of the Safety Mechanism Pattern.

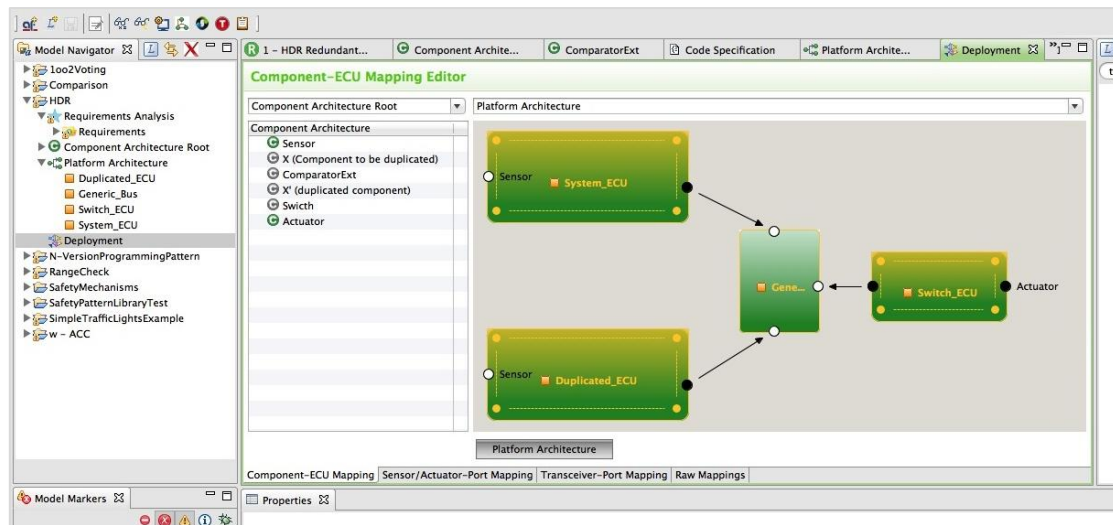


Figure 27: AF3 implementation example – HDR: Deployment to *TechnicalArchitecture*, reproduced from [196]

The supporting safety case is shown exploded in Figure 28. Using the relations described in the GSN³² standard [58] and implemented in AF3, it is possible to perform completeness and consistency checks on the safety case, such as “solutions must stem from goals and no other elements” or “no open goals remain”. Using the extended GSN elements – supporting modular reusable safety cases – shown in Figure 22, the *HDR* safety case also incorporates the safety cases of *RangeCheck* and *Comparison*, at the appropriate location.

Not shown but also implemented with AF3, are the corresponding requirements of any of the central *LogicalComponent* elements, e.g., the *Comparator*, as well as the formal specification of mechanism behavior, allowing automatic test suite generation, as discussed in section 5.1.

³² More details provided in Appendix A1.

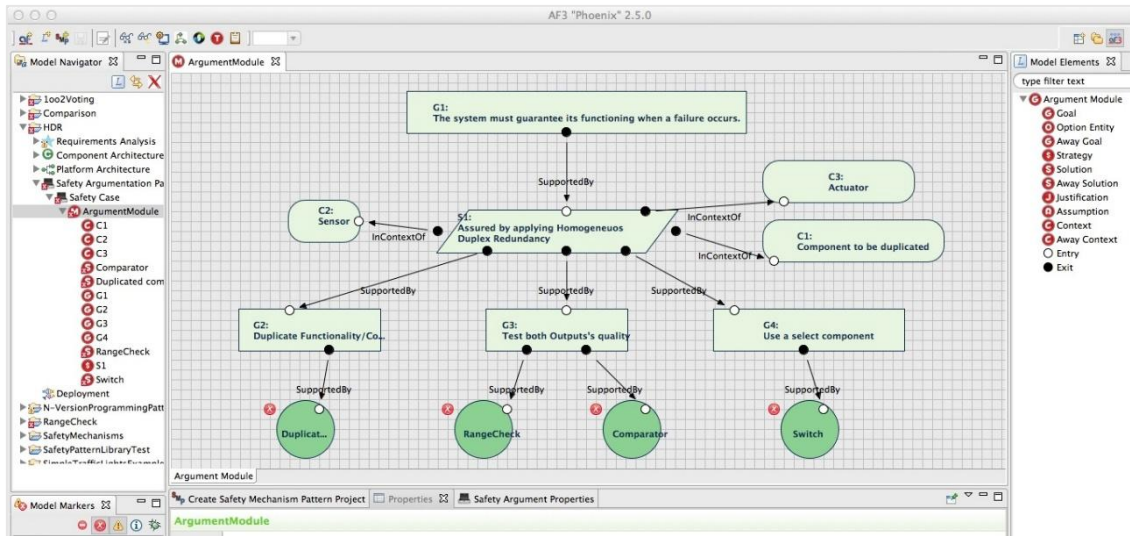


Figure 28: AF3 implementation example – HDR: SafetyCase, reproduced from [196]

5.4 Evaluation of Safety Mechanism Pattern Coverage in AF3

In conformance with our approach’s stated target of avoiding the typically purely verbose nature of capturing pattern attributes, our use of a model-based tool suite means that we are able to integrate this information and capture most of it into the elements of the structure model. There still are, however, some limitations:

- AF3 does not support a dysfunctional component behavior view – describing the possible malfunctioning behavior of components – and as such, it is not possible to capture the failure modes of components explicitly. In turn, this makes it difficult to precisely describe the *Rationale* and *Motivation*, as well as *Applicability*, of a Safety Mechanism Pattern, for which failure modes are a key motivator and essential piece of information.
- AF3 does not support failure propagation analyses. This decreases the quality of capturing *Applicability*, *Participants*, and *Collaborations*, because these attributes are all best explained through the improvement in failure handling provided by the Safety Mechanism Pattern. This improvement cannot be captured explicitly in a dedicated view.
- Our pattern library implementation currently only supports pattern Inclusion relations, but not Specialization. Pattern Inclusion is only partially automated, as we still have to manually define how the correlating artifacts in each pattern interact. The link between the patterns guarantees a traceability and maintainability of the complex (composite) pattern.

Nevertheless, we were confident that the quality of attribute capturing and pattern reuse in AF3 had greatly improved with the newly added safety case support and our dedicated Safety Mechanism Pattern Library to AF3. To test this, we once more performed the evaluation previously carried out in section 5.2.2, with the following results:

Table 8: Evaluation of SMP coverage in AF3 after implementation of our approach, reproduced from [184]

Attribute	Symbol	Evaluation Remarks	Score
Name	N	Captured in the name of the Safety Mechanism Pattern itself.	1.00
Rationale	R	This attribute is captured in the <i>SafetyRequirement</i> entity and supported by the Safety Case Elements. Somewhat limited by missing dysfunctional view and generic requirements artifacts in AF3.	0.75
Motivation	M	This attribute provides the rationale behind using the pattern and is captured in the <i>SafetyRequirement</i> entity, with any additional, implementation-specific information being captured in the <i>SafetySpecification</i> entity, and supported by the Safety Case elements. Also somewhat limited by missing dysfunctional view in AF3.	0.75
Applicability	A	Describes the architectural context of the safety mechanism as well as instantiation information and is captured by the <i>LogicalArchitecture</i> entity and augmented by the workflow described in section 5.3.4. Somewhat limited by missing dysfunctional view and support for failure propagation analysis in AF3.	0.75
Structure	S	This attribute is captured by the entities <i>LogicalComponent</i> and <i>LogicalArchitecture</i> and their relation and supported by data model of the pattern library approach.	0.75
Participants	P	Captured at an interaction level by the <i>LogicalArchitecture</i> entity and at a structural level by the structure of the Safety Mechanism Pattern library.	0.75
Collaborations	CL	This attribute is captured by the structure of the safety mechanism library and the safety case supporting it. Somewhat limited by missing dysfunctional view and failure propagation analyses in AF3.	0.75

Consequences	CN	The parts of this attribute describing which components in the logical architecture or inside the Safety Mechanism's component have to be instantiated or further developed upon usage are captured in the structure model as well as in the workflow described in section 5.3.4. Limited by missing dysfunctional view and failure propagation analyses in AF3.	0.50
Implementation	IM	Is for the most part captured in the <i>SafetySpecification</i> entity as well as the entities <i>LogicalComponent</i> and <i>LogicalArchitecture</i> and their relation. Somewhat limited by missing dysfunctional view and failure propagation analyses in AF3.	0.75
Implications	IC	Is still not supported very well. Out of the constituent aspects reliability, modifiability, costs and execution time, only the latter two could be improved due to the added ability to capture deployment relations and <i>TechnicalArchitecture</i> models in the pattern library. Reliability analysis requires tool support for dysfunctional description and failure propagation.	0.50
Usage Classification	UC	States to which category of Safety Mechanisms (Fault Avoidance, Error Detection and Error Containment/Handling – see section 3.3) the pattern belongs, and is captured as a textual attribute in the library element container, but would benefit from a more general classification scheme which also incorporates failure propagation analysis and allows the automated inheritance of practical functional and dysfunctional behavior aspects.	0.50
Example	E	The graphical interface of AF3 makes the captured Logical Architecture fairly easy to understand, so we can consider it to be an example. Some additional description would have facilitated the understandability of the pattern even more. We are considering	0.75

		adding some visual aids to the pattern library wizard to help the user understand how the pattern's elements collaborate to carry out the functionality.	
Related Patterns	RP	This attribute is captured in the data model for inclusion relations by the recursive aggregation relation of the <i>SafetyMechanismPattern</i> entity. The implementation is incomplete and ongoing – limiting the propagation of pattern attribute relations to a semi-automatic support.	0.50
			Total Tally (out of 13)
			9.00
			Percentage
			69.2%

The results show a significant improvement in AF3's ability to capture Safety Mechanism Patterns for reuse. We interpret this as a validation of the approach's premise based on predefining the information targeted for capture in the model-based development environment independently of the environment itself.

The shortcomings exposed by the evaluation reflect limitations due to the domain-independent nature of AF3, with many of the missing features being very specific to tools specifically developed for safety-critical embedded systems application development. Some of the ongoing implementation works may yet improve the results by a small margin.

5.5 Summary

This chapter presented the first instantiation of the proposed reuse approach within the AutoFOCUS3 research CASE tool. The goal was to assess feasibility and implementation effort in a domain-independent modeling environment and to evaluate the impact of the approach on the completeness and adequacy of solution capture. An initial baseline evaluation of the tool's native reuse capability was followed by a structured enhancement of its meta-model and workflow using the MODPAL framework, resulting in a tool-supported pattern library implementation for safety mechanisms.

The implemented data model, usage workflows, and role definitions were described in detail, and a concrete implementation example was provided to illustrate practical application (with further examples available in Appendix B). The results showed a marked improvement—over 25% increase in attribute capture coverage—confirming the feasibility and effectiveness of the approach within a generic tool setting.

While the evaluation necessarily involved a degree of subjectivity, the focus remained on comparative impact rather than absolute metrics. As such, the results clearly demonstrate

the added value of the structured approach and its ability to guide meaningful tool enhancement for reuse.

The next chapter builds on these findings by applying the approach in a domain-specific context, evaluating whether domain adequacy and modeling richness further influence the effectiveness and ease of reuse framework instantiation.

6 Instantiation in SAFE – an Automotive Safety Modeling Framework

This chapter presents the second evaluation of the approach, done within the automotive domain-specific SAFE Technology Platform. The work presented here is based on research activities carried out by the author leading to specifications within the integrated development environment for the ITEA2 SAFE³³ / EUROSTARS Safe-E project, with proof-of-concept implementations in AF3. The initial results were well received by the reviewing OEM advisory board and industry experts of the SAFE project and published in a publicly available deliverable [197], as presented in section 1.7. The evaluator builds upon initial concepts developed by the SAFE/Safe-E project partners in [130] and [207].

The evaluation leverages the capabilities of the SAFE meta-model, which, by reusing well-established constructs – such as EAST-ADL and AUTOSAR – and extending them with safety specific information, allows a very precise description of automotive systems in the context of safety-critical development. It was our assumption that such a highly context-specific and content-rich platform would allow an even better capturing of Safety Mechanisms than that possible in a domain-independent environment such as AF3.

To evaluate the instantiation, we first introduce the SAFE project and its meta-model and technology platform, followed by a description of the work done to extend the SAFE meta-model with safety case documentation capability. This is followed by an overview of the existing safety mechanism description, and concluded with a mapping of our approach onto the SAFE platform.

6.1 The SAFE Project

Increasing complexity due to new and emerging technologies as well as the imminent introduction of the new automotive-specific functional safety standard ISO26262 were the main triggers for the SAFE/Safe-E project. Safety is a system aspect, which must be considered during early design phases, first at system level, and then recursively at the individual component levels, to guarantee correct development. The hitherto standard practice of developing components independently – often by independent suppliers, followed by “big-bang” integration by the automobile manufacturers had already reached its limits towards handling the upcoming complex and interdependent systems, such as new propulsion technologies and increasing levels of automation, and also reinforced the divide between system development and safety engineering. The project aimed at analyzing the emerging standard³⁴, and developing concepts for the ISO26262-compliant

³³ <http://www.safe-project.eu>

³⁴ Originally targeting a 2009/2010 start, the project finally began in July 2011, a few months before the ISO26262 was introduced in November 2011.

development of Safe Automotive software architecturEs, the project's acronym, while reusing pre-existing established architecture descriptions such as AUTOSAR, to facilitate integration and acceptance into the automobile industry. To cover such an ambitious spectrum of topics, the project consortium consisted of 18 partners covering the automotive supply chain – automobile manufacturers, Tier 1 suppliers, silicon suppliers and tool developers, as well as academia – working together over the course of three and a half years. Upon its successful conclusion at the end of 2014, the project was bestowed with an award of excellence for its efforts in business impact and standardization by the European research framework ITEA³⁵.

As discussed in section 2.3.1, the ISO26262 is an automotive-specific adaptation of the generic basic standard (IEC 61508) for functional safety requirements in electrical and electronic (E/E) systems. This adaptation can be applied to all operations during the safety lifecycle of systems, their electrical and electronic components and software elements include safety-related functions.

6.1.1 SAFE Meta-model Development Methodology

The increase in system complexity translates to complex software architectures that require multiple views and abstraction layers to describe, and necessitate complex analyses – as mandated by safety standards – involving a large and not seldom disparate amount of information, as discussed in more detail in chapter 2.

In order to carry out these analyses in a model-based approach, it is necessary to capture the required information in the meta-model. Seeing as the analyses, and thus the information to be captured, range from abstract architecture description, through requirements refinement and on to hardware metrics and SW/HW components, it becomes necessary for the meta-model to be able to capture all this information [208].

The necessary information was collected by analyzing the ISO 26262 safety standard, among other sources, and captured and refined in the form of requirements for the SAFE Meta-Model. The requirements were then further refined and concentrated according to the project scope. The refined requirements served as a basis for the specification and implementation of the SAFE meta-model and Tool Platform, upon which the second instantiation of our approach is constructed.

6.1.2 Reused frameworks

In order not to reinvent the wheel, existing modeling frameworks and architecture description languages (ADL) were analyzed for suitability to the required purposes. The next section gives a more detailed overview of EAST-ADL and AUTOSAR, the two frameworks upon which the SAFE meta-model is built.

³⁵ <https://itea4.org/project/safe.html>

6.1.2.1 East-ADL

Chief among these, the Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL), which is an ADL based on concepts from UML, SysML and AADL, but adapted for automotive needs and compliance with AUTOSAR. EAST-ADL is optimized for top-down description in the automotive domain and is the subject of numerous – past as well as ongoing – expansion and refinement research projects. It was found to cover many (but not all) of the aspects required, especially those pertaining to software architecture description. Most importantly, EAST-ADL offers multiple abstraction levels and dedicated system views, as seen in Figure 29, to handle various concerns and suit the needs of different stakeholders. The concept of abstraction layers and system views was introduced in section 2.5.1.

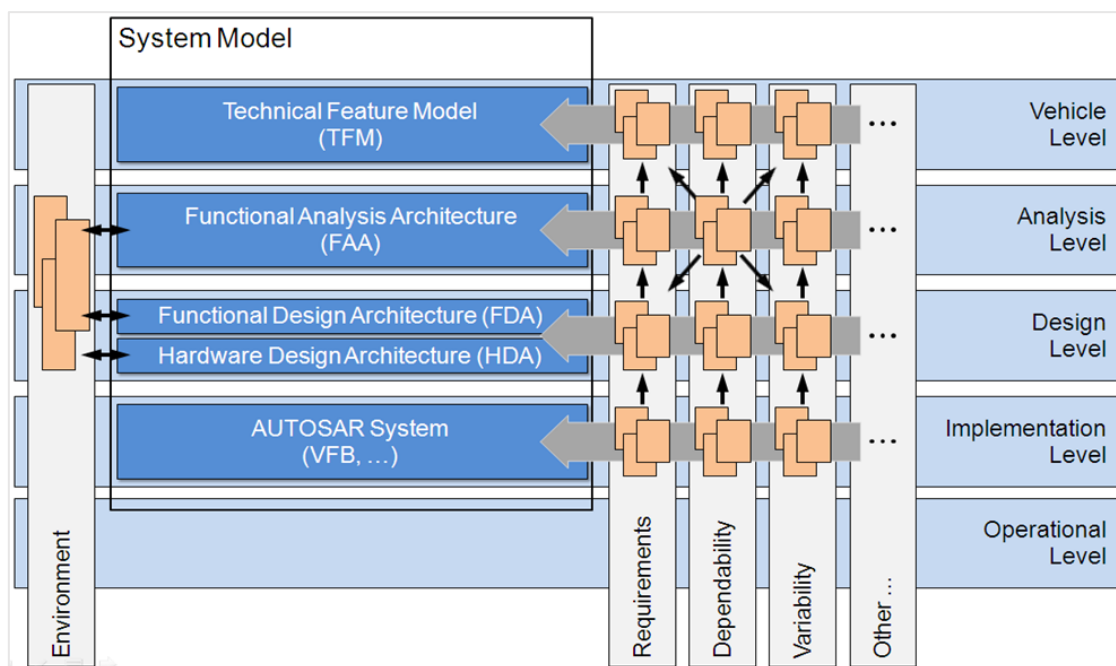


Figure 29: EAST-ADL Abstraction layers and System views model as applied to SAFE-MF [209]

More recently – and well after the research works discussed here were finished – EAST-ADL2 was introduced, as the second major iteration of the EAST-ADL architecture description language, specifically designed to support the model-based development of automotive embedded systems across all abstraction levels. Building on the original EAST-ADL, version 2 refines and extends the language’s capability to represent the complete automotive system lifecycle, from high-level vehicle features and functional requirements to software, hardware, and implementation-level details. EAST-ADL2 emphasizes integration with AUTOSAR by aligning its lower abstraction levels with AUTOSAR concepts, while maintaining higher-level constructs for requirements engineering, functional modeling, variability management, and safety analysis. It enables

traceability across abstraction layers, supports compliance with standards like ISO 26262, and fosters reuse and consistency within large-scale, safety-critical automotive system development projects.

EAST-ADL thus serves as a basis, from which the SAFE Meta-Model is introduced as an extension package. This only got us halfway, as EAST-ADL remains a fairly abstract high-level description, while all analyses and information pertaining to the development of safety-critical product ultimately have to be assigned to components; in our case in AUTOSAR, described in more detail in the next section.

6.1.2.2 AUTOSAR

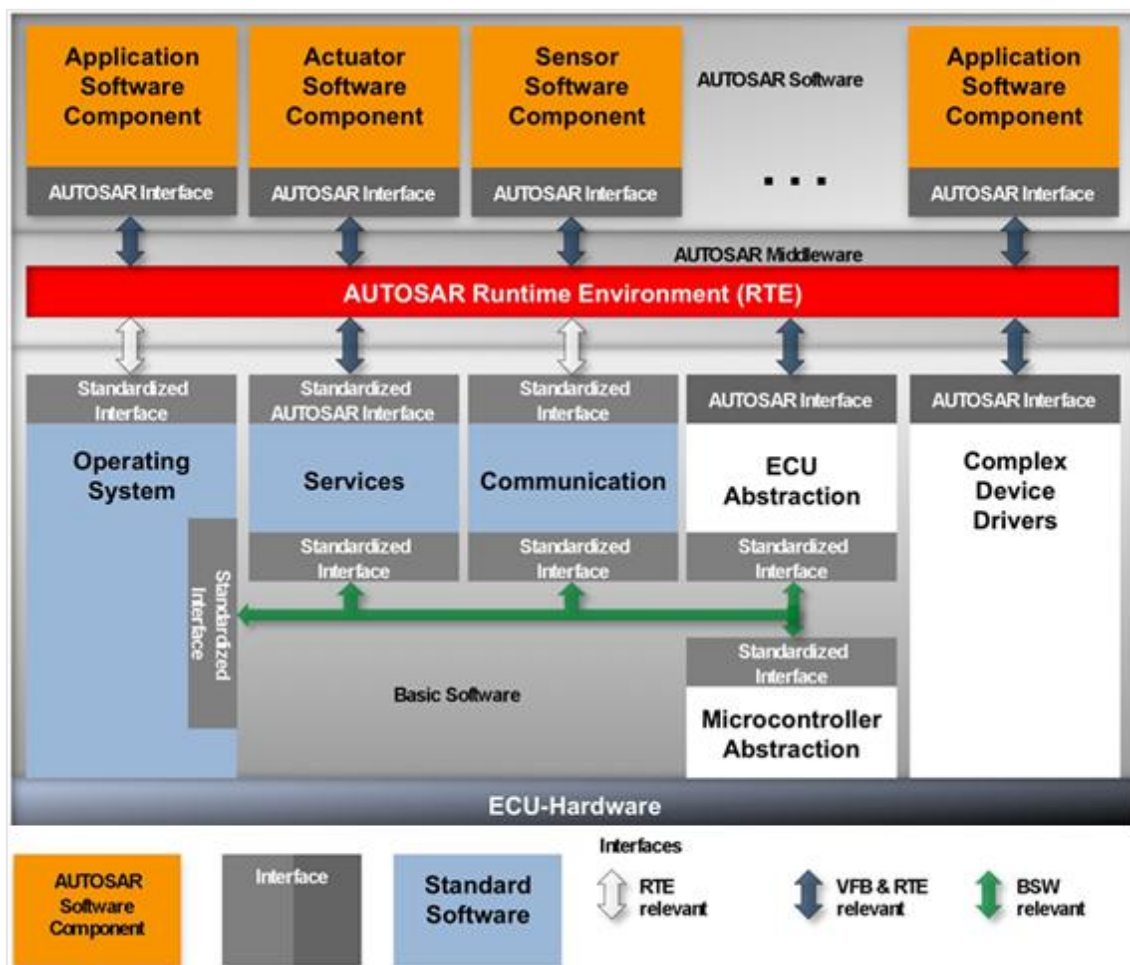


Figure 30: AUTOSAR System Architecture in 2015 as applied to SAFE-MF [210]

The AUTomotive Open System Architecture (AUTOSAR) is an open and standardized automotive software architecture framework, jointly developed by vehicle manufacturers, suppliers and tool developers, to facilitate and standardize software communication, transfer and maintainability across hardware platforms [210]. AUTOSAR provides a layered architecture that separates application software from hardware through well-

defined interfaces, enabling portability, reusability, and interoperability across suppliers and vehicle manufacturers (OEMs), as seen in Figure 30.

Newer versions of AUTOSAR offer both *classic* platforms – the *original*, which was the only version available and considered at the time the SAFE-MF was being developed, optimized for real-time embedded systems in traditional ECUs – and *adaptive* platforms – geared towards so-called automotive high-performance computing³⁶ for connected and autonomous vehicles. It defines standard software modules, configuration mechanisms, and a methodology for system software development, and is tightly aligned with industry safety and quality standards such as ISO 26262 and ASPICE.

6.1.3 SAFE Meta-model Structure

Figure 31 shows a simplified overview of the SAFE meta-model as an extensions-package of EAST-ADL, compliant to and implemented AUTOSAR components.

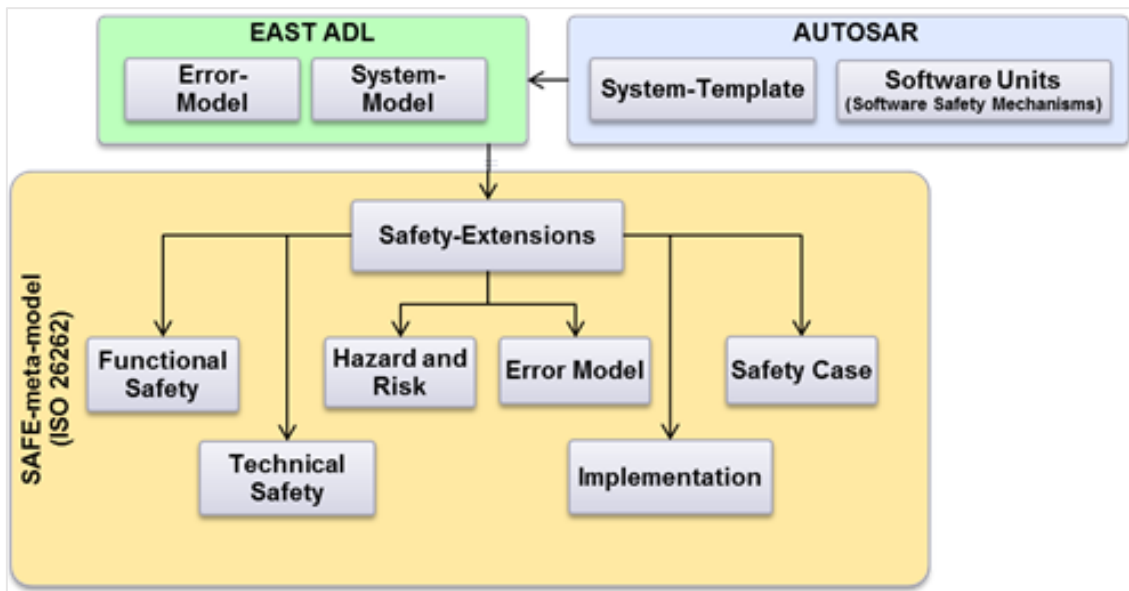


Figure 31: SAFE Meta-model Extension [208]

Each of the packages in the safety extensions – whose class diagram is shown in the next figure – is captured in an exhaustive ISO26262-compliant artifact data model, with complex relations guaranteeing consistency among the individual extensions, as well as between the extensions and the linked EAST-ADL and AUTOSAR elements. This complex structure of the SAFE meta-model underpins the extent of its expressiveness, with a very high coverage of safety information required by the ISO26262. As explained in [208], the salient point when considering the construction of the SAFE meta-model is that EAST-ADL does not map directly onto AUTOSAR. There are large gaps in many areas, much (sometimes conflicting) overlap in some areas, and some areas that simply

³⁶ no relation to *actual* high-performance computing used in supercomputers and compute clusters to solve advanced computation problems.

do not directly match. Mapping EAST-ADL to AUTOSAR is thus best done within a specific context; in this case the context is functional safety.

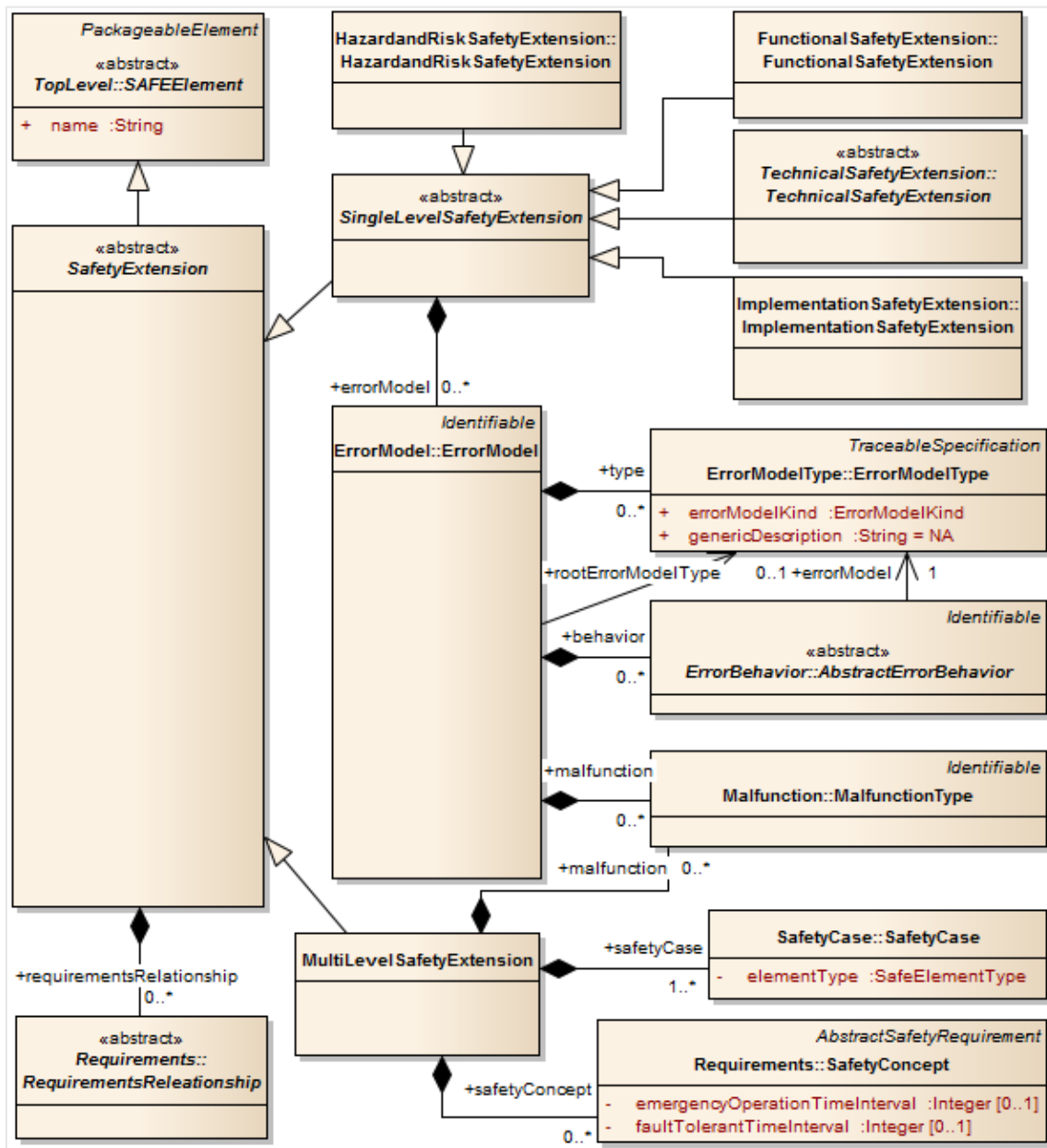


Figure 32: SAFE Meta-model: Safety Extensions Class Diagram [197]

As such this gives rise to three kinds of artifacts in the SAFE meta-model:

- 1:1 mapping onto existing EAST-ADL and AUTOSAR artifacts; the existing artifacts perfectly suit the identified needs.
- Extensions of pre-existing artifacts; the existing artifacts provide a solid basis but some necessary information is missing.
- New Artifacts; this necessary information was not available or adequately covered in the existing modeling languages.

The last two types of artifacts present the majority of the work done in the SAFE project, targeting the integration of safety information into the central meta-model and tooling framework. We will be exploring two such extensions in the following subsections, which are most relevant to Safety Mechanism capturing, starting with an extension carried out by the author, Safety Case modeling.

6.2 Safety Case Modelling in SAFE

The ISO26262:2011 was one of the first safety standards to explicitly demand the existence of a “safety case”, used to link safety goals to the solutions fulfilling them and to demonstrate compliance with the standard. As explained in detail in section 2.4, safety cases do not merely provide links from safety requirements to the artifacts fulfilling them, but go beyond to providing a structure for the argument chain, which is often much more complex than simple direct tracing between elements. Even simple safety cases may involve linking many artifacts of different types into an argument chain to show how the safety goal is fulfilled. However, this first version of the standard lacked detailed guidance on constructing and evaluating these safety arguments, as discussed in [211].

As discussed in detail in section 2.4.2, given the varied and complex nature of the artifacts a safety case for a complex system may have, introducing a dedicated view for safety cases into the project’s model-based approach was a natural step, as it would facilitate linking all the model-based artifacts provided by the SAFE meta-model.

We first started by analyzing safety case expression capabilities within EAST-ADL, which we found unsatisfactory, and went on to extend the SAFE meta-model with safety case elements based on GSN, as will be explained briefly in the next sections. More details are provided in [197].

6.2.1 Safety Cases in EAST-ADL

EAST-ADL introduces different levels of abstraction within a hierarchical modeling concept to facilitate handling the complexity of systems under development, as previously seen in Figure 29. Besides the different abstraction levels, EAST-ADL includes several dedicated packages, such as the variability package, the timing package, and the dependability package which was of special interest for this work task. An overview of the dependability package is given in Figure 33, featuring various sub-packages targeting dependability concerns in the automotive domain. These include the basic artifacts needed for expressing safety activities, such as hazards, hazardous events and safety goals.

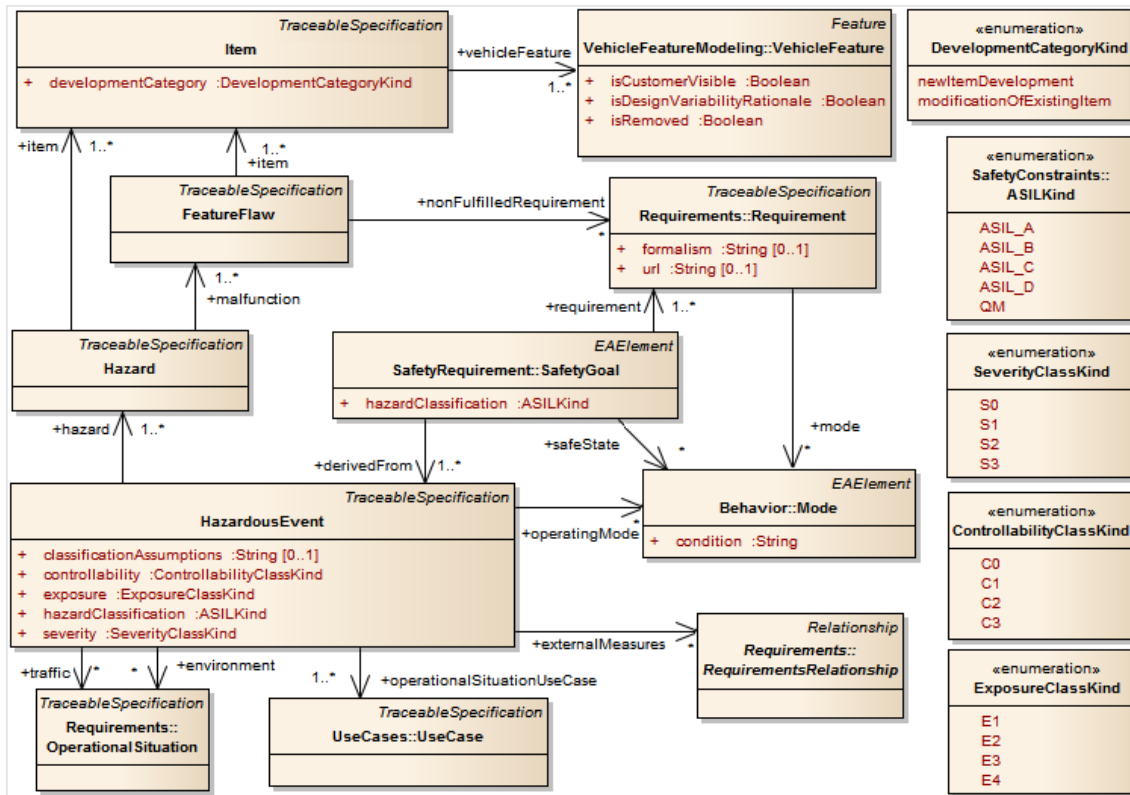


Figure 33: EAST-ADL Dependability Package [209]

Furthermore, EAST-ADL (referenced here in version 2.1.11 from 2013, as used in the SAFE project) includes Safety Case description capability in the SafetyCase Sub-Package, shown in Figure 34. It was our objective to reuse as much as possible of the already existing content provided in EAST-ADL.

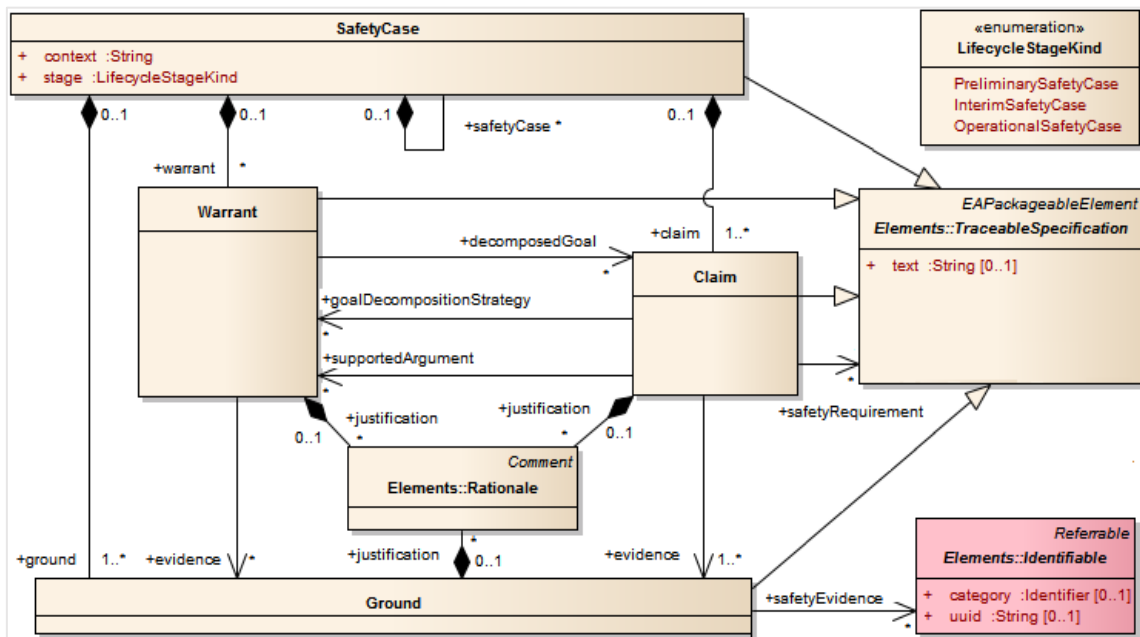


Figure 34: EAST-ADL Safety Case [209]

The main elements of the EAST-ADL safety case are [209]:

- **Claim**
A *Claim* represents a statement, the truth of which needs to be confirmed and which has associations to the strategy for goal decomposition and to supported arguments. It also holds associations to the evidences for the *SafetyCase*.
- **Ground**
A *Claim* is based on *Grounds* (evidences) - specific facts about a precise situation that clarify and make good the *Claim*. *Grounds* represent statements that explain how the *SafetyCase Ground* clarifies and make good the *Claim*. Grounds have associations to the entities that are the evidences in the *SafetyCase*.
- **Warrant**
Warrant represents argumentation of the facts to the *Claim* in general ways. The *Warrant* entity has associations with the decomposed goals and with the evidences for the *SafetyCase*.
- The overall objective of an argument is to lead the evidence to the claim and hence arguments are actions of inferring a conclusion from premised propositions. An argument is considered valid if the conclusion can be logically derived from its premises. An argument is considered sound if it is valid and all premises are true.

The elements and their relations are in turn based on the Toulmin Model of Argumentation defined in Stephen Toulmin's 1958 work "Uses of Argumentation" [212], the main concept of which is shown graphically in Figure 35. The semantics – shortly described – are: *Data* qualified by a *Warrant* (Argument) is deemed to satisfactorily support a *Claim*, unless a *Rebuttal* can be found.

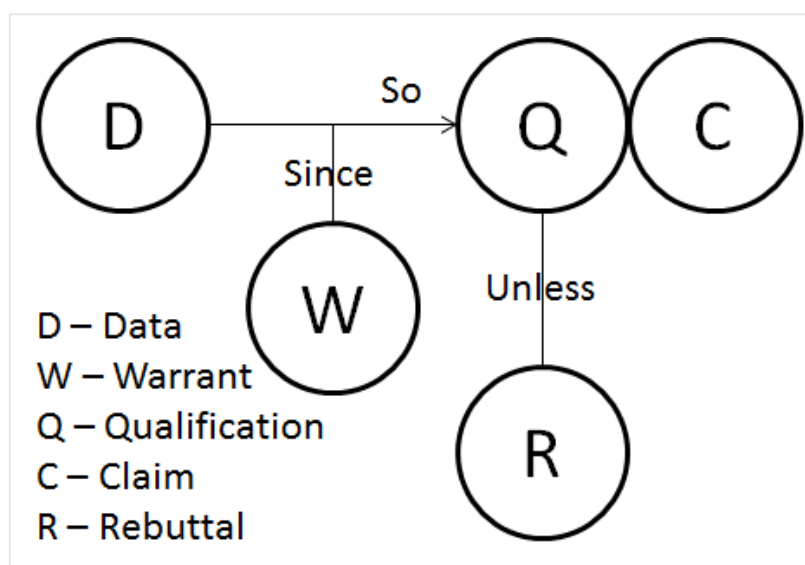


Figure 35: Toulmin Argumentation Concept Elements [57]

Expressing context, which is a central concept of safety argumentation, and differentiating assumptions from strategies or justifications, is difficult using this scheme. Furthermore, Toulmin argumentation – as used here – follows a forward-facing affirmative line of argumentation, i.e., the claim is valid unless refuted, which may be useful when speaking of innocence in a court of law, but not conducive to good engineering in the safety critical application domain. Safety engineers are indeed encouraged to do the opposite, consider their systems as unsafe until proven otherwise.

We thus chose to discard the EAST-ADL safety case description and replace it with GSN. This decision specifically provides us with the necessary argumentation elements, such as context, assumption and justification elements, and also because of GSN's neutral argumentation semantics. Replacing the existing safety case extension in EAST-ADL with GSN has already been suggested in several research projects, such as the proposed safety case extensions for EAST-ADL in the ATESS2 project³⁷.

6.2.2 Safety Cases in the SAFE Meta-model

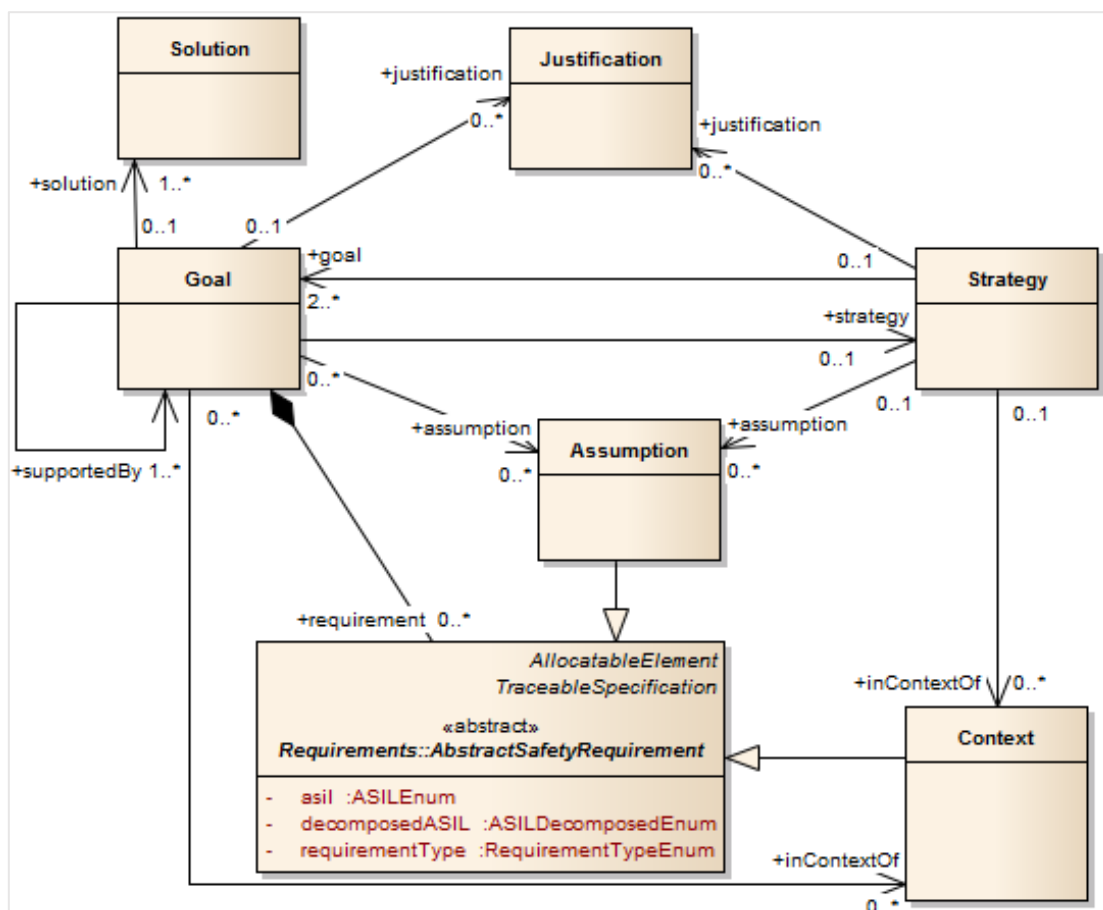


Figure 36: Safety Case class extension to SAFE meta-model [197]

³⁷ Advancing Traffic Efficiency and Safety through Software Technology.
<https://cordis.europa.eu/project/id/224442>

Using GSN as a reference, we extended the SAFE meta-model with a safety case package, whose class proposed internal class diagram can be seen in Figure 36. The Safety Case class is centered on the Safety Goal, which can be decomposed directly into 2 or more goals or indirectly via the use of a strategy. Each goal shall resolve to at least one solution. These rules, defined by the GSN standard [58], can be used in a model-based approach to test safety cases for completeness and consistency. Contexts and justifications are presented and assumptions can be included as well, with the safety goal forming the center of the safety case structure. The GSN elements and relations shown in the figure were previously introduced in section 2.4.3, with more details provided in Appendix A1.

Most importantly, with the introduction of safety case support to the SAFE meta-model, the linking of development artifacts to safety case elements can be automated. For instance, the *Solution* element, which represents any information that supports or proves that the *Goal* it is connected to is in fact achieved, can contain information of many types, as discussed in detail in section 2.4.2. Figure 37 presents how the *Solution* element can be specialized to hold a wide array of information supporting the claim, simply via linking it to general existing EAST-ADL classes that could supply this information.

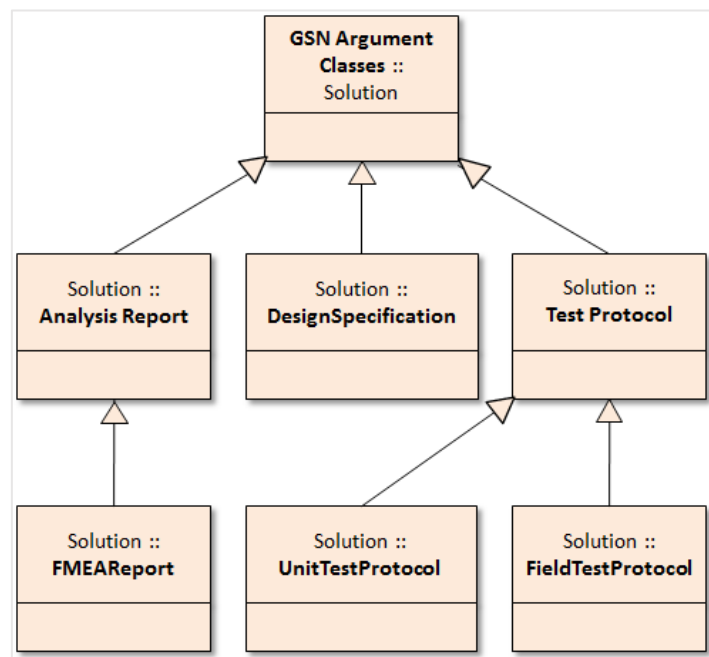


Figure 37: Potential automation links form EAST-ADL elements to Safety Case Solution [197]

SAFE meta-model artifacts which could potentially provide supporting information to safety cases – and as such were prime candidates for automatic linking to the corresponding safety case elements – were extensively analyzed. The analysis identified many artifacts, highlighting the expressiveness of the SAFE meta-model and the large potential benefits from automation via tool-support. The results are too large to include in this thesis and can be found in section 8 of [197].

Traceability is guaranteed by the SAFE Meta-model satisfy mechanism linking hazard description and subsequent analysis to the corresponding top level safety goal and onto the concept to which the tactic and the requirements it fulfills belong.

On the other hand, the requirements are then refined through the multiple abstraction layers supported in the SAFE Meta-model (through a mapping of system level views from EAST-ADL to ISO26262, for more details see the SAFE Meta-model description [132]) to the specification of Software Safety Requirements (SSR), which holds the tactic fine description and carries the same name. This then links to implementation descriptions, configurations and actual instances. As specified in [130], and demonstrated in [207], using this structured description and the implementation specific information it captures, it is possible to automatically generate code for the captured Tactics, with the generated code complying with the specified target component architecture, for instance AUTOSAR.

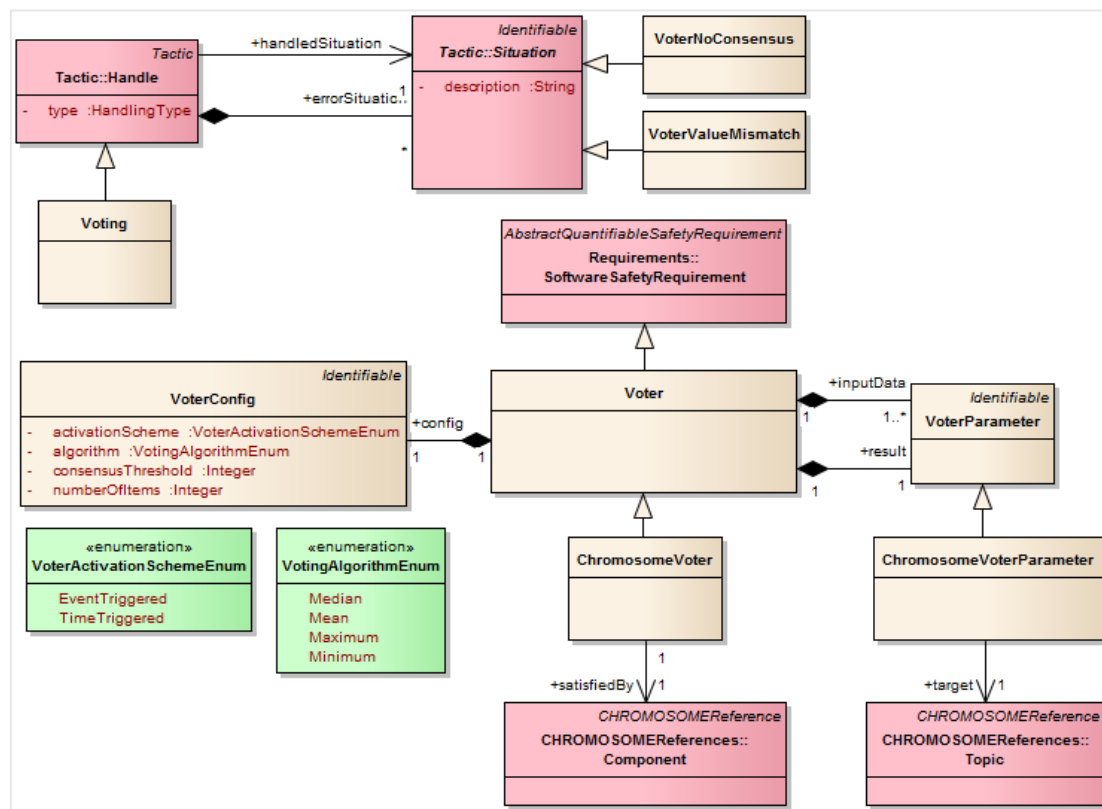


Figure 39: Voter Tactic in SAFE Meta-model [130]

Figure 39 shows the class description of a *Voter* Tactic from the SAFE Meta-model. The lower part of the figure shows the Tactic descriptor and the implementation specification artifacts for both AUTOSAR components, and CHROMOSOME, an open-source research middleware [213]. The instantiation of the Tactic classification mechanism introduced in Figure 38, is seen in the upper half of Figure 39, as well as in the right hand side of Figure 40, which presents a simpler Tactic, *HealthMonitor*, with CHROMOSOME implementation only shown on the left hand-side of the figure.

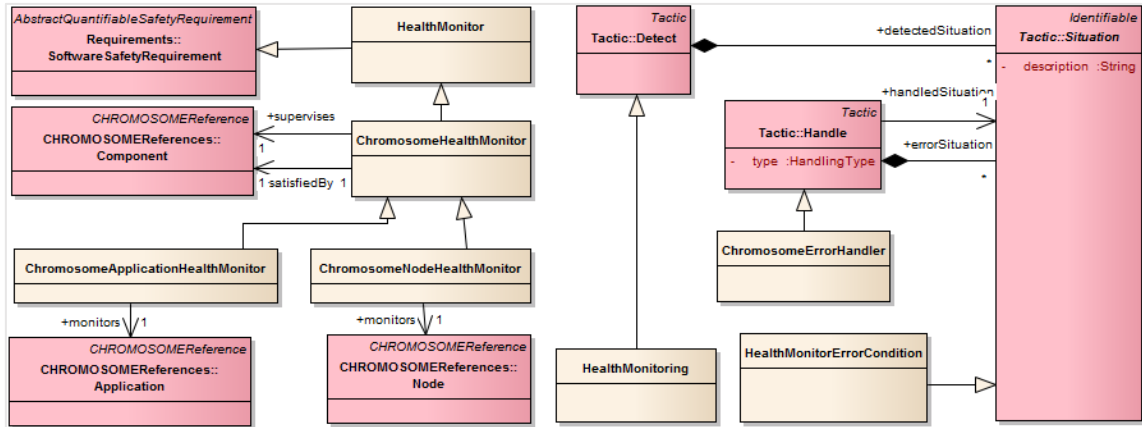


Figure 40: HealthMonitor Tactic in Safe Meta-model [130]

To fully appreciate the expressive power of the SAFE meta-model, we rearranged the previous figure, such that the link between the classification aspects and the requirements specification and implementation description be made explicit, through the inclusion of the SAFE Meta-Model requirements structure representation.

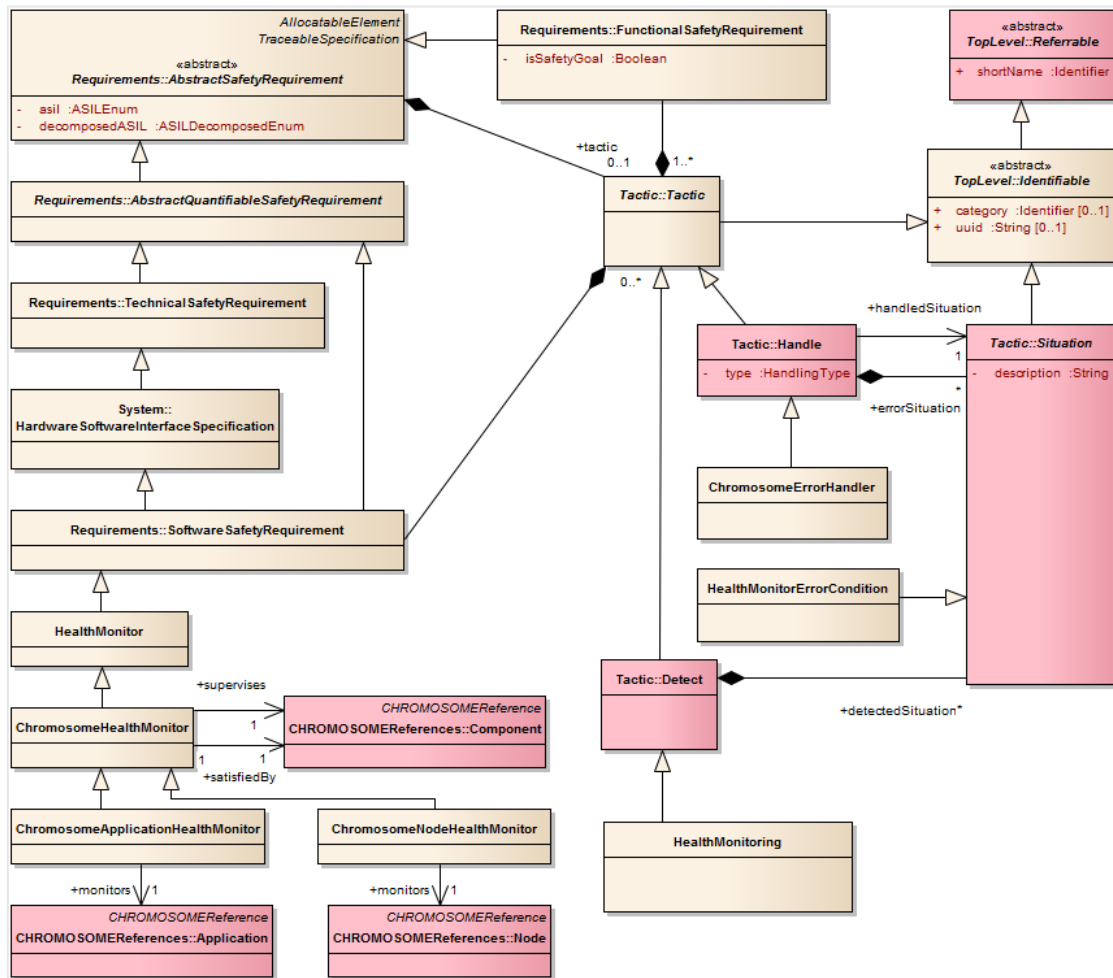


Figure 41: High-level View on HealthMonitor Tactic [197]

It is noteworthy that no direct link exists between the classification branch and the implementation specification branch, which are connected only at the apex through specialization of the requirements artifacts. To prevent model ambiguity through loss of information and assure conformity between the two branches, a further categorization of the functionality of the tactic was added to the tactic classification mechanism. This is then matched to the tactic description Software Safety Requirement (SSR) using OCL³⁸.

6.4 Evaluation of Safety Mechanism Pattern Coverage in the SAFE Platform

To better understand Tactic expression in the SAFE meta-model, we need to regroup the artifacts from Figure 41 into abstract categories to more closely resemble the artifact categorization introduced in Figure 11.

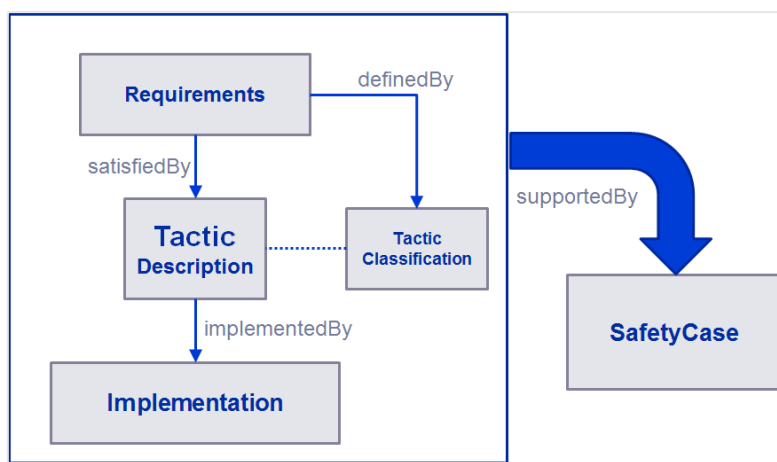


Figure 42: Abstract Artifact Structure for Tactic Library Approach in SAFE Meta-model [197]

This regrouping gives us the following main high-level categories.

- **Requirements**

Requirement capturing in the SAFE meta-model builds on the already extensive requirements expression capability of EAST-ADL, extending it with ISO26262-compliant hazard description, as seen in Figure 4, and refinement and breakdown into functional and technical safety concepts, as seen in the left hand side of Figure 41, among many features. Coupled with support for malfunction behavior and failure description, as well as failure propagation and cut-set analysis, this means that this category is captured quite thoroughly.

- **Tactic Specification** (via Tactic Description and Tactic Classification)

³⁸ Object Constraint Language is a precise text language UML extension used to provide constraint – as well as object query – expressions for a meta-model, that cannot otherwise be expressed by diagrammatic notation.

Both categories comprise means of specifying the Tactic structure and relations, including offering a classification of tactics into categories, and enriching the Tactic description with information necessary for correct implementation.

- **Tactic Implementation**

SAFE Tactics are linked to AUTOSAR or CHROMOSOME implementations, which means that the implementation descriptions are very precise, with well-known instantiation mechanisms, and plenty of information about their applicability and consequences.

Extending these categories to be supported by the safety case extension of the SAFE meta-model introduced in section 6.2, in a structured approach as shown in Figure 42, it would be possible to capture most pattern attributes within the model elements.

Using this structure to map our pattern library approach onto the SAFE meta-model and technology platform, we can evaluate the capturing of Safety Mechanism Patterns in SAFE, as seen in the next table:

Table 9: Evaluation of SMP coverage in the SAFE Modeling Framework, reproduced from [184]

Attribute	Symbol	Evaluation Remarks	Score
Name	N	Captured in the name of the Tactic and the AUTOSAR or CHROMOSOME implementation it links to.	1.00
Intent	I	This attribute is captured using the extensive requirements expression capability of EAST-ADL, extending using SAFE artifacts to cover ISO26262-compliant hazard description, safety concepts, support for malfunction behavior and failure description and propagation.	1.00
Motivation	M	This attribute provides the rationale behind using the pattern and is captured in the requirements and safety concept artifacts of SAFE, and supported by the Safety Case elements. This combination provides for a strong coverage of this attribute.	1.00
Applicability	A	Describes the architectural context of the safety mechanism as well as instantiation information and is captured by the Tactic Description category, which includes not only clear links the category of Safety Mechanisms (Fault Avoidance, Error Detection and Error Containment/Handling – see section 3.3) the	1.00

		pattern belongs to, but to the actual fault models the patterns addresses. Furthermore, the links from requirement to specification to actual implementation artifacts means this attribute is captured very well.	
Structure	S	This attribute is captured by the Tactic Description category, which clearly defines how the pattern relates to other artifacts. The actual implementations are standard AUTOSAR or CHROMOSOME artifacts, so their internal workings are well-known and documented.	1.00
Participants	P	Captured in the Tactic Description Category and supported by the safety concept and safety case artifacts in SAFE.	1.00
Collaborations	CL	This attributed is captured by the structure of the Tactic element and the safety case supporting it. It is further supported by the direct link to the dysfunctional view and failure propagation artifacts. A precise capturing is somewhat limited by the lack of timing annotations in SAFE.	0.75
Consequences	CN	The parts of this attribute describing which components in the logical architecture or inside the Safety Mechanism's component have to be instantiated or further developed upon usage are captured in the Tactic description and the included implementation descriptions, which allow a correct instantiation into multiple target implementation environments, such as AUTOSAR. This attribute is largely dependent on tool support, and there are currently no tool implementations covering the full-range of SAFE specification and capability.	0.75
Implementation	IM	Is captured very precisely by including implementation-related information in the Tactic Description, and then linking to standard, well-defined and documented implementations in target environments, such as AUTOSAR or CHROMOSOME.	1.00

Implications	IC	Is still not supported completely. Out of the constituent aspects reliability, modifiability, costs and execution time, Reliability analysis could be improved in comparison to the AF3 implementation – evaluated in section 5.4 – due to the added ability to capture dysfunctional description and failure propagation. A precise capturing is further limited by the lack of timing annotations in SAFE.	0.75
Usage Classification	UC	Is captured in the Tactic Classification artifact, which not only clearly links the pattern to a category of Safety Mechanisms (Fault Avoidance, Error Detection and Error Containment/Handling – see section 3.3), but to the actual fault models the patterns addresses. The benefit is nevertheless somewhat limited due to the lack of Pattern Relation link to other Tactics at the meta-model level, or indeed of any description of what the pattern categories mean. This denies the capacity for automated inheritance of practical functional and dysfunctional behavior aspects between related patterns in a category.	0.75
Example	E	Despite the great expressive power of the SAFE meta-model and technology platform, there are still no standard tools implementing the entirety of the SAFE scope. Tactics in SAFE have no actual provision for examples. Assuming the implementing tools will have some graphical capability, the potential for capturing explanatory examples will be very high, with the examples being very detailed if needed, due to the expressive power of the underlying meta-model.	0.75
Related Patterns	RP	The Tactic categorization in SAFE is currently used to ensure consistency – via OCL – between the requirements branch and the specification branch of a Tactic, as seen in Figure 41: High-level View on <i>HealthMonitor</i> Tactic [197]. It does not contain any Pattern Relation link to other Tactics at the meta-model level, or indeed any description of what the pattern categories otherwise mean. This denies the capacity for	0.50

A secondary objective was to explore the impact of domain adequacy on reuse performance. The SAFE platform – featuring safety-focused modeling constructs and integrated safety case support, the latter implemented by the author as part of a publicly documented contribution to the corresponding ITEA2 European research project – offered a richer, more semantically aligned environment. The comparative evaluation of the SAFE instantiation against the extended AutoFOCUS3 implementation revealed an additional 20% improvement in attribute capture (rising from 69% to 89%) achieved *out of the box*, without requiring further adaptation or extension work. This underlines the critical role of domain adequacy and content richness in enabling effective reuse.

As with the previous chapter, the evaluation is comparative and not intended to produce absolute tool ratings. It confirms MODPAL’s adaptability and highlights the role of modeling framework expressiveness in enabling systematic reuse. The next chapter builds on these findings, applying the generalized approach in a real-world industrial setting.

7 Instantiation in COREPA³⁹ – a Generalized Industrial Case Study

In this section we will discuss the generalization of the approach and its instantiation in an industrial setting.

This case study establishes the general applicability and feasibility of our approach, along with evaluations of improvements achieved and demonstrations of the guided and prioritized evaluation and enhancement of development environment and tooling capabilities, and is rounded up by a systematic expert survey.

Disclaimer: Much of the detailed information regarding this case study, which continues the research efforts described in earlier phases, is proprietary and continues to be actively used by the author's former employer, a globally leading automotive Tier-1 supplier. Consequently, we have limited the material included in this chapter to information that has been officially released through prior public channels, including conference submissions and industrial presentations. Nonetheless, the results provided clearly illustrate the applicability, utility and positive impact of our approach to a broader problem type: the development of a large-scale component and architecture knowledge base for reusing automotive system solutions in an industrial context.

Statement on Survey Data Collection, Usage, and Anonymization

The survey responses collected for the evaluation and validation of the proposed approach were obtained through a voluntary and pseudonymized survey conducted internally at a globally leading automotive Tier-1 supplier – referred to from this point forward as *the Company* – for the purpose of evaluating and improving an MBSE-enabled architecture knowledge capturing and system solution reuse approach. Prior to survey distribution, all applicable internal guidelines regarding surveys, data protection, and participant privacy rights were carefully followed, and the necessary approvals from the designated internal authorities were obtained. Participants were explicitly informed at the outset that their participation was voluntary, that their responses would be anonymous to both the survey conductors and management, and that their answers would be used strictly for evaluation purposes. No directly personally identifiable information was collected.

The dissemination of findings through academic publication, aimed at verifying and communicating the results, was explicitly planned and approved from the earliest planning stages of the survey. A corresponding paper, referencing the survey and preliminary findings, was submitted to an academic conference prior to the researcher's departure from the Company, evidencing the original intent for public academic dissemination. Although the paper was not accepted for publication, this thesis continues that dissemination effort in line with the original communicated intentions.

The data presented in the analysis chapters are aggregated and anonymized. For full transparency, Appendix A5 includes the raw pseudonymized Likert-scale responses to survey items. No names,

³⁹ COREPA refers to a component library and corresponding architecture knowledge capturing and solution reuse framework in which the approach was initially instantiated and which was *formerly* used in this industrial setting. The trademark associated with COREPA has since lapsed, and the name is referenced here solely for clarity and continuity with prior academic publications by the author.

direct identifiers, timestamps, or project-specific references are included, and participant order has been reshuffled to further minimize any risk of re-identification. The use of this survey data complies with established ethical research standards, including respect for participant confidentiality, transparency of purpose, and academic integrity. No confidential business information beyond the scope of the originally stated research objectives has been disclosed.

7.1 Research methodology

This chapter represents the *Descriptive Study II (DS-II)* phase of DRM, focused on evaluating the developed support in an industrial, real-world context.

The work presented here follows the form of a *longitudinal industrial case study* conducted over approximately eight years at a globally leading Tier 1 automotive supplier. In line with established case study methodologies as discussed in [214] and [215], the study focused on a bounded case: the application and evolution of a model-based framework within a specific organizational context, enabling systematic reuse of architectural knowledge and defining a prioritized roadmap of capability extensions leading to the synthesis of reference vehicle architectures.

Three characteristics define its longitudinal nature. *First*, the research was conducted over an extended period, allowing for the observation of long-term effects and the co-evolution of the framework and its context [215]. *Second*, the study involved multiple iterative phases, including the initial instantiation of the reuse framework, its application via a scoring system to assess development environments, targeted toolchain enhancements based on identified gaps, and repeated re-evaluation cycles. These cycles enabled insights into the framework's refinement over time and its integration into evolving industrial practice [216]. *Third*, the study collected and analyzed data from multiple sources — including artifact analysis, practitioner feedback, and a final expert survey — to understand both process and outcomes in a dynamic real-world setting [214].

In the final phase, a systematic survey was conducted among expert practitioners who had used the approach over several years, capturing qualified insights on usability, effectiveness, and organizational impact, thereby validating the support's real-world relevance and utility. Guidelines derived from best practices in research methodology were collected and applied, as discussed in section 7.3.1, with additional methodology and application rigor topics discussed where needed in the subsequent section. The threats to the validity of the survey are discussed, together with their mitigations, in section 7.3.4.

This multi-phase, time-extended evaluation reflects key principles of longitudinal industrial case study research, emphasizing real-world relevance, iterative improvement, and sustained observation over time, and rounds up our DRM-based research work.

7.2 Instantiation of the Pattern Library approach in Industry

7.2.1 Background: COREPA Component Library

Driven by the incessant rise in the number of integrated control units, communication systems and software components, managing architectural complexity in the automotive industry, let alone mastering it, is becoming an increasingly difficult task. Combining the recent push towards automated and autonomous driving systems with the rich array of functions and components the Company offers, selecting the right set of functions necessary to fulfill the customers' wishes and needs, as well as the corresponding technical components and system design can quickly become a daunting task without adequate methodical and tool support. As a leading automotive systems supplier, the Company's challenges are further increased by the sheer diversity of the designs its numerous customers employ. The situation is exacerbated even further by the fact that these complex functions are not the sole inhabitant of the technical components they are deployed on, but must often harmoniously coexist with other advanced assistance functions and even share resources with them as necessary.

The Company's solution for managing the complexity and number of functions, components and designs in which they can be combined centers around a reusable functional and technical component library, COREPA (COmponent REadiness Preparation by Architecture analysis). In its infancy, COREPA had the aim of capturing existing solutions (both functional and technical) into models, and using these models to perform post hoc analyses of overlaps in both functional and technical contexts, in order to recommend potential synergies and savings. The initial focus of COREPA was on functionalities of one (of three) divisions of the company, which offered Advanced Driver Assistance Systems (ADAS), braking solutions, and passive safety (e.g., airbags) and sensor solutions. A more future facing goal was to analyze anticipated future system function bundles for a frontloading of potential synergetic savings into the development of the relevant functionalities. One example of such a function bundle could be the set of functions and technical components needed to achieve a 5-star rating in the Euro NCAP⁴⁰ standard crash test – and their design(s) into a system architecture, as illustrated in the following figure.

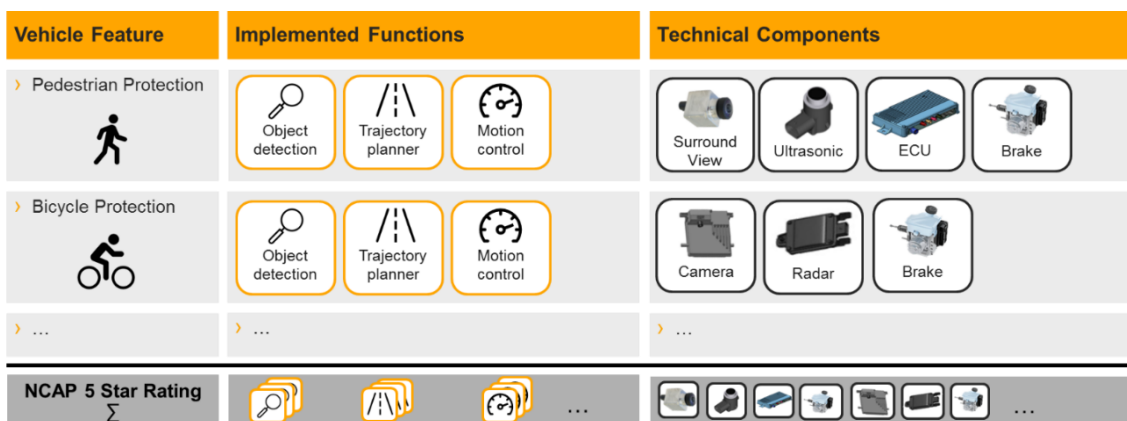


Figure 43: Features, Functions and Components in COREPA⁴¹

⁴⁰ European New Car Assessment Programme – an organization that evaluates vehicle safety performance through crash tests and other assessments

⁴¹ Image by the author and copyright of the Company 2018, released since for multiple public events.

Such a future analysis capability would enable an additional strategic goal: COREPA would play a role in the definition of the technology roadmap for the Company's products, by analyzing synergies in product roadmaps, capturing key design drivers for future integrated systems and identifying cornerstone future technologies. The reuse of available system design solution descriptions – requirements, function descriptions, safety analyses and architecture designs – was a secondary but increasingly important goal. System designs in COREPA are primarily modeled using SysML, with functional components captured in a logical view, and technical components in a physical view.

7.2.2 Approach to Instantiation & Research Foundations

In the longitudinal industrial case study presented in this chapter, the systematic reuse methodology developed in this thesis was applied to an existing modeling component library at a globally leading Tier-1 automotive supplier. The systems being combined in the modeling component library were developed in isolated organizational silos, usually resulting in redundant artifacts and significant inconsistencies, reducing efficiency and hampering combination and effective reuse. Moreover, system architects typically attempted to anticipate systemic interactions and emergent effects individually and without structured methodological support. Such challenges were exacerbated by the growing complexity associated with increased functional integration, automation, and connectivity in contemporary automotive systems. Initially, the library employed informal and ad-hoc methods for combining individually developed systems, performing a post hoc search for redundant components, and attempting to consolidate the domain architectures and map the consolidated functional components onto technical assets. Together, these issues formed a formidable set of contextual challenges.

During this industrial case study, the proposed reuse approach was initially applied in a limited scope, discussed in section 7.2.3.1, strategically targeting an area where the potential for enhanced reuse capability could be readily demonstrated through appropriate tooling support and corresponding modeling extensions. This selective entry point was chosen for two primary reasons. First, the implementation had to remain aligned with existing engineering responsibilities, as the initiative was pursued alongside a full-time engineering role rather than under a dedicated research assignment. Second, a focused, early-stage success was considered crucial for building credibility and securing buy-in from both management and peers. The resulting “quick win” not only demonstrated the feasibility of the approach but also laid the groundwork for broader organizational interest in systematic reuse, enabling deeper and more strategic iterations of the method.

Empowered by this success, the reuse approach evolved beyond its initial application, catalyzing a foundational rethinking of both the reuse library and the company's overarching MBSE strategy. The ensuing discussions, running over the course of many years and in which the author played a central role, contributed significantly to the formalization and harmonization of modeling practices, including the standardization of core Systems Engineering processes and terminology. As these initiatives gained traction, they supported the establishment of a company-wide systems engineering capability development and qualification program — tailored in alignment with INCOSE guidelines and recommendations — which the author co-created and led over multiple years. The influence of this work extended to broader organizational dimensions, fostering critical

discussions on collaboration practices and interdisciplinary integration. While these transformations were initially driven informally by the author through peer-recognized expertise in Systems and Safety Engineering and MBSE, they were later formalized through the author's appointment as Principal Expert for Systems Engineering and Architectures at the Vehicle and Mobility Solution level within the organization.

On the technical reuse capability aspects – after the initial phase, a more systematic needs analysis enabled by the model-based reuse approach introduced in this thesis indicated a clear need for a structured, unified architectural framework to overcome the aforementioned set of contextual challenges, one that is specifically aimed at enabling reuse and emphasizing commonalities. Consequently, the concept of a *reference architecture* was introduced by the author and operationalized within this industrial context. The definition and approach adopted in this thesis integrate aspects from multiple established frameworks in literature—most notably, the conceptualizations provided by Cloutier et al. [168] and Nakagawa et al. [170].

Cloutier et al. define a reference architecture as a generalized architectural framework encompassing industry best practices, established patterns, and embedded architectural knowledge, thereby serving as a reusable, standardized starting point for system design [168]. Adopting this perspective provided significant value, as it aligned closely with the need to systematically capture, formalize, and embed tacit architectural knowledge and proven best practices directly into the modeling library artifacts, transforming them from loosely organized components into structured and reusable building blocks.

The main attributes that Cloutier highlights in defining a reference architecture include [168]:

Abstraction and Generality

A reference architecture is intentionally abstract to remain widely applicable. It defines essential components, structures, and interactions, but avoids overly prescriptive or detailed specifications that would limit applicability or innovation.

Reusability and Guidance

The core intent of reference architectures is to provide guidance for system architects and designers, enabling efficient reuse of proven architectural solutions, thus avoiding repetitive problem-solving.

Standardization and Best Practices

Reference architectures encapsulate industry standards, best practices, and proven methods within a specific application domain, effectively reducing risk and fostering consistency across systems.

Knowledge Repository

They act as repositories of explicit architectural knowledge, capturing both design rationale and experience-based solutions that practitioners can readily access, adapt, and apply to specific implementations.

However, the challenges faced within this particular industrial setting required going beyond Cloutier's general framework. Due to the parallel, siloed development of system solutions within the organization, it became essential to explicitly address both the commonality and variability across systems and domains. In this respect, the perspective of Nakagawa et al. [170] proved particularly relevant. They describe a reference architecture as a set of standardized architectural artifacts *designed not only for reuse but explicitly structured to capture and manage domain-specific commonalities and variabilities*. This definition explicitly addresses challenges arising from domain-specific redundancy and variability management – precisely the issue faced in the case study due to parallel and isolated solution developments.

Thus, the reference architecture developed and instantiated in the case study effectively combined these two perspectives. It provided standardized architectural patterns and explicitly embedded design rationale and architectural knowledge (following Cloutier's framework), and simultaneously emphasized structured management and explicit maximization of system commonalities (aligned with Nakagawa et al.). Consequently, the resulting reference architecture facilitated highly structured, domain-specific reuse while dramatically reducing redundant development artifacts. It also enabled a coordinated anticipation of systemic and emergent behaviors that architects previously struggled to address individually. The architecture provided a common platform to assess and ensure that systemic effects were correctly anticipated and accounted for, crucial given the increasing complexity and interconnectedness of modern automotive systems.

Furthermore, the reference architecture was enhanced through the reuse framework developed in this thesis to provide automated calculations and synthesis of optimized deployment configurations, significantly extending beyond traditional structural definitions of reference architectures traditionally found in literature, as represented by Cloutier and Nakagawa and their colleagues' works, but also, for instance, by Bass, Clements, and Kazman [169], who primarily emphasize standardized structural interfaces and software-level guidance. By integrating automation and optimization capabilities, the implemented reference architecture thus transcended purely structural standardization, offering active support for rigorous architecture synthesis and optimization processes in practice.

In summary, the concept and instantiation of the reference architecture concept presented in this chapter represent a purposeful integration of existing definitions. Cloutier's emphasis on capturing and embedding architectural patterns and knowledge, combined with Nakagawa et al.'s explicit management of domain commonality and variability, provided a powerful framework to systematically address previously siloed developments, reduce redundancy, enhance systemic architectural integrity, and enable effective and automated reuse in complex, safety-critical automotive contexts.

7.2.3 Applying the MODPAL approach to COREPA

7.2.3.1 Initial instantiation

Assessing the completeness or adequacy of the capturing of solutions in COREPA was a difficult task, because there was no “definition of done”, i.e., no standardized structured definition of when

a solution or component was described well enough in the models to allow analysis or reuse. Initial queries to company experts attested between 70–80% adequacy and completeness, albeit admitting that these were “only estimates”.

As part of a push to place the reuse aspect at the forefront of COREPA and increase the overall maturity, we introduced the pattern library method to guide and provide structure for the reuse process. We knew that many immediate workflow issues, e.g., partially automating deployment calculations – with multi-criteria optimization including safety considerations – was possible, as we had demonstrated this previously in [206]. The attribute catalogue used to capture Safety Mechanisms for reuse was not adequate for this new context, and a new one had to be defined. To lower the initial instantiation hurdle, we used the previous attribute catalogue as a starting point, and condensed it during the investigation down to 7 standard attributes – well-known from Design Pattern literature: (*Name, Type, Description, Context, Applicability, Behavior, Implications*).

A scoring scheme similar to the one used in the previous two use-cases – but adapted to this new kind of reuse problem – was defined and an assessment was then carried out, similar to the previous two case studies, to evaluate how well these attributes were covered by the current models. The results were used to guide initial tool extensions to COREPA, including an initial variant management (for different versions of the same functional component, e.g., a *trajectory planner*) and arbitration capability, as well as a large undertaking to leverage modeling to integrate multi-criteria architecture evaluation and deployment optimization into the COREPA capabilities. This capability allowed automatic optimized deployment calculations of logical solutions to technical components, leveraging formal Design Space Exploration (DSE) capabilities, the results of which were published in [217]. The results of the evaluation of COREPA reuse adequacy – before and after the completion of the first wave of tool extensions – are shown in the following table:

Table 10: COREPA Evaluation Results before and after the first phase of applying the approach, reproduced from [184]

Template Attribute	Definition	Evaluation before tool extensions	Evaluation after tool extensions
Name	Meaningful name of the item	1	1
Type	Type of the item	1	1
Description	Short statement describing the item	1	1
Context	Exemplary context in which the item has been applied	0.75	1
Applicability	Specific usage limitations	0.25	0.5
Behavior	Description of how the item behaves as a black box	0	0.25
Implications	Impact on using this item on non-functional requirements (safety, execution time,...)	0.25	1
Total Tally (out of 7)		4.25	5.75
Percentage		60.7%	82%

The result of this preliminary evaluation was pleasing as it showed significant improvement – from 60,7% to 82% capturing quality. Tool improvements and extensions continued.

7.2.3.2 Expanded application of method & subsequent enhancements

Using this early success as a motivator, a more comprehensive attribute catalogue defining multiple “reusable COREPA items” was defined, and the old evaluation was then revisited. The exact attributes in this newer catalogue – along with the exact descriptions of the resulting enhanced COREPA models, as well as those in newer iterations, are privileged intellectual property of the Company in active use and can thus unfortunately not be included in this thesis.

The results of this new evaluation – the first of several to come – were more sobering; the framework adequacy and completeness evaluation in capturing reusable items – before any of the extensions carried out in the previous 2 years – dropped from 60.7% in the previous evaluation to 39.6%. The result of the older tool extensions – considered in the previous table – *combined with newer ones* resulted in an evaluation of 72.9%, more clearly demonstrating **how much work had been done, and how much still needed to be done**.

Subsequent iterations led to a cycle of expanded expectations towards the reuse capability, assessments of the existing reuse capabilities, corresponding prioritized extensions of the tooling and modeling frameworks, and repeated reevaluations. Stakeholder expectations were systematically updated in regular interval every 2-3 years, while intermediate feedback was regularly collected informally and integrated into ongoing tool and framework enhancements. The newer attribute catalogues are used not only as a guide for which aspects of the framework to develop next, but also as a gauge for the return on investment any framework development work or tool extension would bring, by analyzing the projected improvement in the framework evaluation vs. the expected effort/cost.

More importantly, the findings of these iterations extended beyond the direct tooling and reuse model capabilities, and onto the necessary organizational enablement for reuse. These two categories of enhancements are discussed in the following two subsections.

7.2.3.3 Enhancements to the MBSE Approach and Modeling Framework

Iterative applications of our method led to the identification of many key gaps in the existing modeling approaches and precipitated substantial expansions of the existing MBSE approach and modeling framework. Previously, the organization suffered from inconsistent terminology and a fragmented modeling framework, impeding effective reuse. For instance, confusion between technical and physical architecture concepts abounded, as the terms were used interchangeably. Additionally, the term “functional architecture” was often misused to describe logical or solution-specific views, prematurely situating modeling efforts in the solution space and neglecting critical early-stage problem space explorations. Most modeling efforts were limited to a *functional architecture* – in fact a logical view of the system, and a *technical/physical architecture* – exclusively a physical view of the system, usually with part numbers. Relations between these views were managed in an ad hoc fashion, with a high percentage of modeling capabilities left unleveraged.

Addressing these and other findings, a structured modeling framework with clearly defined and consistently applied views was developed by the author, distinguishing clearly among operational, functional, logical, and physical views. In line with its strategic focus on Systems Engineering excellence, the Company pursued a seamless model-driven engineering approach; treating models as first-class citizens where relevant information is generated, not just stored. The concepts of this multi-view modeling framework, seen on the next figure, form the foundation of the Company's central MBSE framework, with implementations driven by expert colleagues, based on SAF⁴² [218], whose viewpoints are shown in the next figure.

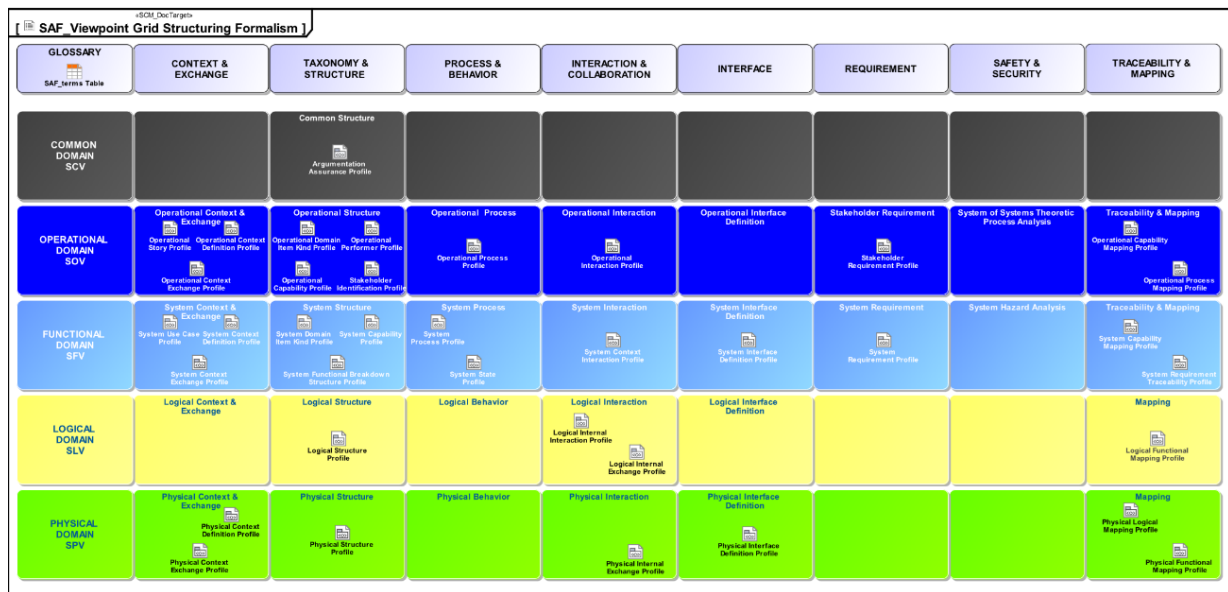


Figure 44: Viewpoints of SAF - see footnote [218]

The resultant framework fully embraces the *separation-of-concerns* and *divide-and-conquer* paradigms, drawing on ISO42010:2022 [80] and ISO15288:2023 [29] to define stakeholder viewpoints as well as hierarchy and abstraction layers, as illustrated in the next figure. The framework facilitated better reuse by ensuring earlier-stage synergies were recognized, such as sharing resources among functionalities that could not operate concurrently (e.g., sensors, software components and compute platform shared between *highway automated driving* and *automated parking* features).

Another enhancement was brought about by introducing a distinct technical view. This technical view allowed abstraction from concrete hardware components, enabling discussions around abstract concepts such as *compute nodes* and *communication channels*. The approach emphasizes frontloading design and analysis capabilities into the modeling framework. The resulting technical metamodel enabled the application of formal methods to support automated allocation relationships, eventually enabling complete synthesis of vehicle architectures from a predefined pool of technical components.

⁴²The System Architecture Framework (SAF) is a domain-independent SysML-based system architecture framework dedicated to support Model Based Systems Engineering (MBSE) of technical systems, which can be used independently or complimentary to Enterprise Architectures. SAF is publicly accessible, and developed by the SAF Working Group of GfSE - the German chapter of INCOSE.

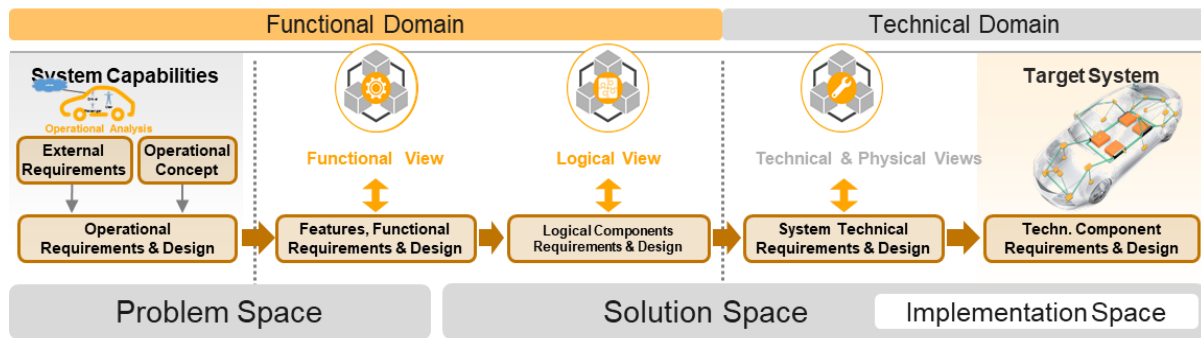


Figure 45: Modeling Framework Views and Perspectives at the Company. Image by the author and copyright of the Company, taken from a public event [219]

These enhancements enabled a significant shift from manual architecture definition, aggregation and consolidation; a time-consuming and error-prone endeavor – toward automated generation and synthesis, greatly improving the efficiency and effectiveness of architecture reuse at scale. Additional details are provided in [220], and expanded on in [221] – both co-authored by us, where more information is also provided about the COREPA viewpoints and modeling artifacts. An explanation of the developed tooling capability, featuring a visualization of the workflow resulting in multi-criteria pareto-optimized architecture deployment options generated by the tool is shown in the next figure.

Automated Architecture Evaluation, Analysis & Synthesis

Correct-by-Design Multi-criteria Design Space Exploration

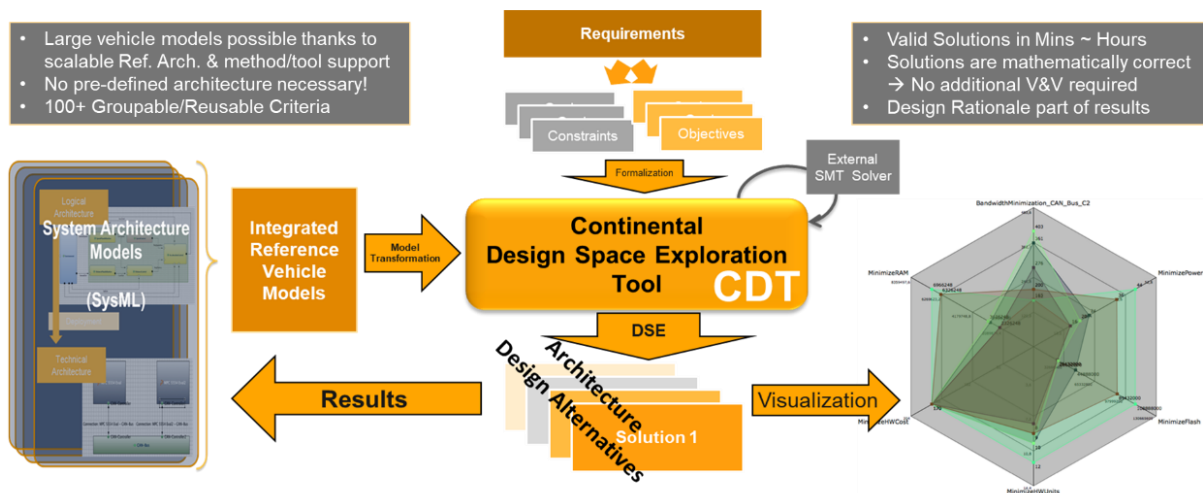


Figure 46: Overview of developed architecture evaluation and synthesis tooling. Image by the author and copyright of the Company, taken from a public event [219]

In 2020, the company decided to expand architecture and solution reuse efforts to encompass entire vehicle architectures and beyond – onto infrastructure and solutions for fleets. During a competitive internal evaluation process involving approaches proposed by different parts of the organization, the developed method achieved clear and unanimous support, with a significant lead over competing approaches, despite the broader politically dynamic context.

As a consequence, the enhanced capabilities enabled by the extended modeling framework were leveraged even further in the successor framework to COREPA, with the author continuing as technical/scientific principal. In the successor project, which was active at the time of our departure from the *Company*, and hence cannot be detailed here, the synergies start with a consolidated *feature catalogue*. We drew primary inspiration for this concept from investigating **Product Line Engineering (PLE)**, as described in ISO/IEC 26580 [222]. Multiple reference vehicles are defined operationally; by target markets, feature loading, and market segment. Complex variant and model management allows the selected feature set to translate to modeling assets in the functional view, where custom adjustments can be made and the consolidated functional architecture generated, and onto the logical view where design assets are consolidated and arbitrated, forming the basis for the synthesis of vehicle compute E/E architectures with automatically calculated deployment/scheduling pairs. As a preparation to integrate these capabilities into the Company's cloud-enabled virtualization and collaborative development framework, the architecture exploration and synthesis tooling suite underwent a successful refactorization and initial migration; from a standalone executable running on powerful local workstations, to a cloud-based service accessible by anyone, leveraging a data-driven globally collaborative mindset.

7.2.3.4 Enhancements to Organizational Systems Engineering Capabilities

Beyond technical enhancements, the broader organizational capabilities in Systems Engineering required significant development to effectively support systematic reuse. The automotive industry generally – with the company being no exception – faced challenges related to inconsistent terminology and a fragmented understanding of Systems Engineering practices (mostly limited to the management of requirements databases and the creation of informal architecture diagrams treated as *suggestions* which were quickly superseded by reality), and varied process maturity across different product lines and business units. This inconsistency significantly hindered collaboration and undermined the potential for reuse.

To address these issues, comprehensive efforts were undertaken to standardize and unify the systems engineering processes across the organization, aligning with ISO 15288 guidelines and including tailored processes for innovation phases to ensure that validated concepts could smoothly transition to productization phases without redundant redevelopment efforts. Concurrently, a unified systems engineering taxonomy and glossary were developed and widely disseminated to foster a common language and understanding among diverse teams.

Furthermore, recognizing the critical need for personnel upskilling, an INCOSE-aligned Systems Engineering Qualification Program was established and delivered company-wide. This initiative significantly raised the collective MBSE and systems engineering competence within the organization. This enhanced capability, along with the results of the systematic reuse framework itself, facilitated a continuous reevaluation and learning process, uncovering critical gaps and fostering more strategic development planning. This approach enhanced the perception and acceptance of systematic reuse among engineers and management.

The impact of these organizational measures was assessed and confirmed through an expert survey evaluating the impact of our reuse approach, conducted after many years of applying the

framework at the company. The results (discussed in detail in the next section) demonstrated clear correlations between individual MBSE competence and positive perceptions of the usability and ease of adoption of the reuse framework. Furthermore, a strong correlation emerged between organizational support for reuse (as measured by an Organizational Support Index, OSI) and a positive outlook on reuse methodologies. These findings underscore the importance of aligning technical enhancements with corresponding strategic organizational capability development – to maximize the impact and sustainability of systematic reuse initiatives.

7.3 Expert Survey – Exploring the approach’s multi-year impact

Leveraging the systematicity of the pattern library approach in this industrial setting was a multi-year endeavor, with first activities started in 2016, and significant work ongoing through 2024. After almost eight years of ongoing research and continuous development and collaboration, we wanted to investigate the usability, utility, and impact of the approach, and validate the claims we had made – and believed to have been achieved by using the approach.

Given the highly specific setting of this industrial use-case, especially the fact that the respondents are a very small group of experts who all know each other as well as the author, we decided an anonymized web-based questionnaire-style survey would be the best form for this validation.

The survey was conducted as part of a systematic effort to assess and enhance reuse activities across multiple levels of the organization. It aimed to evaluate the practical impact of the reuse framework and identify areas for improvement based on expert feedback. In the context of ethical research guidelines, this operational background had the added benefit of minimizing the risk of response bias. Specifically, participants were not made aware that their responses would later contribute to doctoral research, thereby avoiding any influence their awareness of the author’s academic objectives might have had on their evaluations—whether through social desirability, leniency, or excessive criticality.

This sub-chapter demonstrates our approach to the survey design and provides foundations for our choices, showcases the survey itself and its results, and provides a discussion of the threats to the validity of our findings and their mitigations.

Appendix A4 provides screen-copies of the actual survey discussed in this section, while tables with the raw pseudonymized responses, aggregated intermediate tables and additional visual result synthesizations are provided in **Appendix A5**.

7.3.1 Survey Design and Implementation

Survey designs typically involve choosing between open-ended questions—allowing respondents freedom to express their perspectives without constraint—and closed-ended questions that provide predefined answers, facilitating comparability and straightforward quantitative analysis, as recommended by Fowler [223]. Patton [224] recommends open-ended questions because they allow experts to "think aloud," capturing richer qualitative insights without prematurely directing or limiting respondents’ thought processes. However, Kitchenham and Pfleeger [225] note that they pose challenges in analyzing responses consistently, potentially limiting the statistical reliability and comparability of survey results. Given the objective of rigorously comparing

usability, understandability, and impact across multiple expert respondents in a structured industrial context, we concluded that primarily using predefined response options would enhance analytical comparability and consistency.

An additional aspect influencing the decision for a mostly closed *questionnaire* type online survey was the fact that the author is known to all survey participants, and had an ongoing influential work engagement with all of them to varying degrees at the time of the survey. As employing an impartial 3rd party interviewer was not practically feasible, this relation was judged to additionally jeopardize any open-ended interviews.

To balance comparability with depth, we nevertheless incorporated one limited open-ended response (“word/phrase” based) to the question on “*the largest drawback or obstacle towards leveraging the reuse framework*”, explicitly inviting candid expert critique.

How long have you used MBSE (Model-Based Systems Engineering) in the context of function design and system architecture development at vehicle level? * ☐ <3 years ☐ 4-10 years ☐ 11+ years	Suggestions for improvement If you used only one word or phrase to describe to the most pressing obstacle for leveraging the approach to capture and reuse architecture knowledge in the organization and create new solutions, what would it be? * (e.g., missing tool support, it's a bad approach, too complicated, ...)
--	--

Figure 47: Examples of closed-ended (left) and open-ended questions (right) in our survey

As shown in Figure 47, we also provided explicit examples of critical feedback to this question (such as “*It’s a bad approach*”). This reduces response ambiguity and signals clearly that critical, honest feedback is genuinely desired, mitigating potential response biases, as reported by Olson [226]. This mixed approach thus effectively combines comparability and openness, appropriately tailored to our specific research objectives and industrial setting.

Based on the highly specific setting of the intended survey – an expert questionnaire with a limited potential number of respondents, evaluating usability, understandability, and impact of multiple years of using an MBSE-based reuse framework in a highly specialized automotive environment in an industrial setting within one company – it was imperative to collect and consolidate best practices of survey design, and use them as guiding principles to design, execute and evaluate the survey.

Using a combination of literature searches, together with directed expert inquiries to respected colleagues in academia working in empirical research, we decided to consolidate two categories of sources for the selection of our guiding principles:

- *First sources*: highly-cited foundational works (hence acceptably older), laying the groundwork for a large body of later research. These were used to derive basic principles.
- *Specialized sources*: a few highly-selective sources offering specialized insights into our particular context (reuse, industrial survey, small number of participants expected, etc.). These sources helped round up our guiding principles and ensure the quality and validity of our survey and its findings.

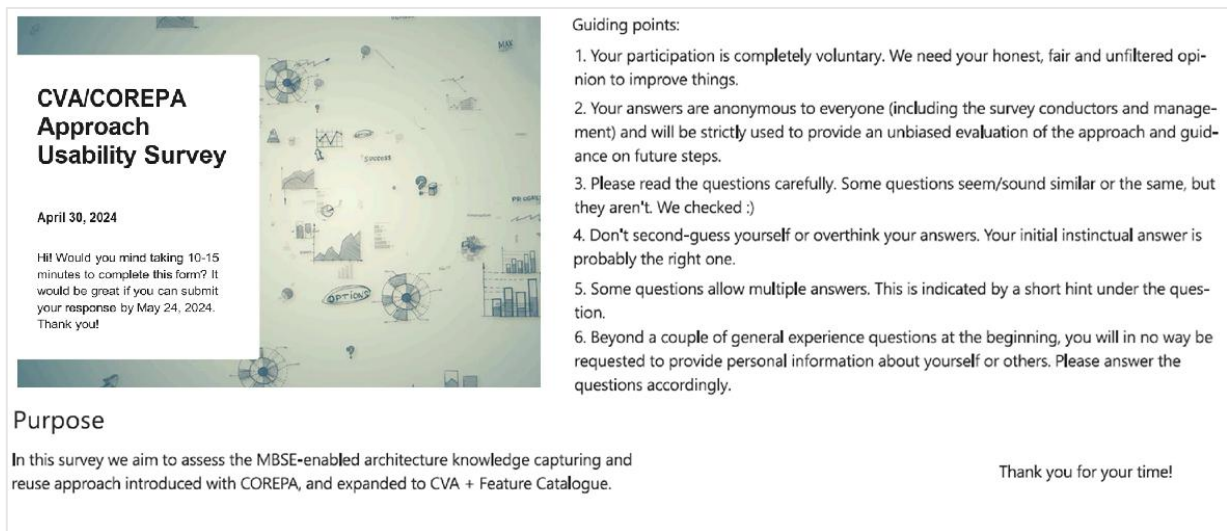
The following sections will enumerate the guiding principles and their implementation, illustrate how the survey was structured and executed, and give an overview of the survey questions.

7.3.1.1 Guiding principles & their integration

Clearly Defined Survey Objectives

First and foremost, Kitchenham and Pfleeger [225] recommend beginning with a clear definition and explicit documentation of the objectives of the survey, specifying what exactly the survey aims to measure or understand. We followed this principle in the design of our survey, ensuring each question directly aligns with the objectives, providing transparency and enhancing construct validity.

We also applied this principle to the survey itself, by stating the objectives of the survey to the participants on the first page of the questionnaire, and providing clear guidelines from the start, as shown on the following figure.



CVA/COREPA Approach Usability Survey

April 30, 2024

Hi! Would you mind taking 10-15 minutes to complete this form? It would be great if you can submit your response by May 24, 2024. Thank you!

Purpose

In this survey we aim to assess the MBSE-enabled architecture knowledge capturing and reuse approach introduced with COREPA, and expanded to CVA + Feature Catalogue.

Guiding points:

1. Your participation is completely voluntary. We need your honest, fair and unfiltered opinion to improve things.
2. Your answers are anonymous to everyone (including the survey conductors and management) and will be strictly used to provide an unbiased evaluation of the approach and guidance on future steps.
3. Please read the questions carefully. Some questions seem/sound similar or the same, but they aren't. We checked :)
4. Don't second-guess yourself or overthink your answers. Your initial instinctual answer is probably the right one.
5. Some questions allow multiple answers. This is indicated by a short hint under the question.
6. Beyond a couple of general experience questions at the beginning, you will in no way be requested to provide personal information about yourself or others. Please answer the questions accordingly.

Thank you for your time!

Figure 48: Opening page of survey with purpose statement and guiding points

Appropriate Sample Selection and Size

Given the fact that we knew our expert population was relatively very small, with less than 30 experts in total having used the method in some form over the years, we aimed to achieve a high response rate as this significantly increases the reliability and generalizability of results in specialized contexts, as well as to explicitly include diverse opinions, including potential critics, to avoid selection bias, as recommended by Wohlin and Runeson [227]. With 15 respondents to our survey (from approximately 25 who were still reachable), we believe we have achieved this objective.

Anonymity, Confidentiality and Ethical Research Considerations

Especially as the author is personally known to the respondents as a leading expert with a prominent position in the organization, we decided to anonymize (as far as possible) both the participation, as well as the responses, of the survey respondents. As seen in Figure 48, we communicated this clearly, along with the fact that no results or responses can be linked to any individual, and that the responses were to be solely used to evaluate and improve the approach

and that individual responses would not be communicated to management. These steps reduce social desirability bias, and increase the validity of the survey results, per Podsakoff et al. [228].

While the survey responses were handled confidentially and not linked back to individuals, full anonymization was not achievable due to the author's familiarity with the respondent pool and the potential for re-identification via indirect demographic or contextual information. As such, the survey data is best characterized as *pseudonymized*, in line with data protection⁴³ and ethical research guidelines, as defined by Israel and Hay [229].

In the interest of reproducibility and transparency, the raw pseudonymized Likert-scale responses collected through the survey are included in Appendix A5. Additional measures have been taken to preserve participant confidentiality, including sanitizing the tables from any timestamps or other digital watermarks which would endanger anonymity, and shuffling the participants to prevent the order of responses from indicating affiliation with respondent groups.

Expert Weighting and Self-assessment Risks

In conjunction with the relatively low number of overall potential respondents, additional risks to the survey results arise from the discrepancies often found in expert surveys, with individual expert responses tending to vary wildly. However, while *individual* expert assessments might differ substantially in *individual* responses, studies have concluded that *combined ratings by multiple experts* remain a valuable tool to identify problematic questions and thereby enhance questionnaire data quality [226].

Furthermore, Chen et al. [230] recognized that as not all experts contribute equally – due to varying expertise, background, and experience – equal weighting of all experts' responses can dilute the validity of findings, especially when expertise varies widely among participants. They proposed integrating subjective and objective metrics to derive a greatly enhanced expert weighting approach, supported by a case study validation.

We have followed these recommendations in our survey design and evaluation of the results. We explicitly requested participants to self-assess (subjective metric) their expertise related to survey topics (e.g., MBSE, reuse practices). We then combined the subjective self-assessments with objective evaluations (e.g., years of experience) to reduce bias and allow meaningful weighting of responses based on expertise. Furthermore, we structured the survey with optional/variation segments depending on respondents' answers, especially to expertise and experience questions. Such an approach significantly mitigates validity concerns related to respondent heterogeneity.

Minimizing Cognitive Load

As recommended by Dillman et al. [231], we kept questions clear, concise, and unambiguous, by using plain language and avoiding double-barreled questions (*asking two things in one question*). We also structured the survey and questions logically, starting from general to specific, to minimize respondents' cognitive effort and reduce dropout rates. Additionally, we segmented the

⁴³ European Union. (2016). *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 (General Data Protection Regulation)*. Official Journal of the European Union, L119, 1–88. Retrieved from <https://eur-lex.europa.eu/eli/reg/2016/679/oj>

survey to add additional or modified questions based on respondents' answers to earlier questions, in order to not confront respondents with questions not relevant to them. Additional information on this structure is provided in the next section.

Balanced Question and Answer Design

As recommended by Olson [226], we included both positively and negatively worded questions to avoid acquiescence (agreement) bias, and clearly differentiated each concept or construct (usability, understandability, impact, etc..) into separate sections, ensuring each item specifically targets one construct at a time, as seen in the original survey screen copy provided in Appendix A4.

As with many exploratory evaluation surveys, some items included were intended to probe participant perceptions in areas where connotations could vary. In particular, interpretive ambiguity was observed for items involving concepts like "disruption," which may carry either positive or negative implications depending on context. Where such ambiguity arose (only one question), affected items were either excluded from quantitative aggregation or presented descriptively, with their interpretive limitations acknowledged. This approach was taken to preserve the integrity and transparency of the analysis.

Additionally, we refrained where possible from "middle-of-the-road" response options to force a decision, while also providing respondents with lower experience with a "Not Applicable" option, and clear instructions to select this response if they do not feel comfortable making an assessment, to avoid tainting the data. Details of the response options and scoring scale will be explained in section 7.3.2.1.

Triangulation of Results

Although explicit triangulation of quantitative analyses – such as our questionnaire data – with qualitative methods (e.g., structured follow-up interviews or focus groups) is generally recommended to enhance validity [224], it was intentionally not pursued in this study for two primary reasons. *Firstly*, the author's role as a known leader in the organization and among participants could inadvertently introduce social desirability bias or other interaction biases in any personal interaction. Face-to-face or even remote follow-ups conducted by the author might have inadvertently influenced participants toward responses they perceive as favorable to the author, thus compromising data authenticity [228]. *Secondly*, practical considerations relating to the participants' workload were paramount. Given that the participants were expert engineers within an industrial setting, the author prioritized minimizing disruption to their primary work responsibilities. It was judged that any incremental benefits derived from explicit triangulation would not sufficiently justify the additional effort and potential inconvenience imposed on these practitioners, aligning with recommendations to carefully balance research rigor with respect for participants' operational constraints [225].

Encouraging Honest Feedback

As shown in the opening notes in Figure 48, we clearly stated the importance of honest and critical feedback, explicitly stating the value of both positive and negative responses, and avoiding language that might bias respondents or subtly encourage particular responses, to minimize biases and improve the validity of the survey results [226].

Pilot Testing and Survey Instrument Refinement

Based on recommendations by Fowler [223], we conducted pilot testing of the questionnaire with three participants representing three archetypes of the target population: a senior expert with 15 years of experience; a former hands-on user now in a managerial role; and a colleague with only limited exposure to the reuse framework.

Each session was conducted online with the author present in the sole capacity of answering any questions or making any clarifications needed by the pilot survey participants. After each session, adjustments were made to the wording, order, and grouping of questions based on participant feedback. This step led to small but significant changes in the wording of some questions, as well as to the structure and order of questions in the survey, significantly improving clarity and response flow, helping to minimize interpretation variance and cognitive load, and hence increasing the validity of the survey and its results [223], [225].

Addressing Validity Threats Explicitly

Last, but definitely not least, it has long become standard recommended practice [231] in all empirical activities in Software and Systems Engineering to explicitly document and address validity threats (internal, external, construct, and conclusion validity) in order to strengthen methodological rigor and support the validity of the findings and claims.

We dedicate section 7.3.4 entirely to this topic, clearly articulating and justifying how the design, execution and evaluation of our survey mitigate potential biases and validity threats.

7.3.1.2 Survey Execution

Based on insights gained during the pilot phase described above, a *two-group phased design* survey rollout strategy was adopted to balance methodological rigor with contextual constraints, as recommended in [225]. We started with a **Core Group** consisting of eight experts actively involved with the reuse approach at the time of the survey – of which five responded. Subsequently, the survey was expanded to an **Extended Group** of 17 additional experts who had previously interacted with the reuse framework to varying degrees, though were not doing so actively at the time of the survey; of these, ten provided valid responses.

This phased approach enabled consistency validation across user profiles (current vs. former participants) and supported sensitivity checks regarding recency of engagement and depth of involvement [225]. Anonymity was strictly upheld throughout, and respondents were explicitly informed about ethical safeguards and the purpose of the survey's [232].

To balance the need for academic transparency with ethical obligations to protect respondent anonymity – particularly given the small, close-knit expert pool – raw response data was shuffled

to prevent identification, and no commentary was made linking individual responses to group membership. Only aggregate results at the group level will be reported, with cross-group comparisons limited to overall metric trends.

7.3.1.3 Structure and layout of Survey

The structure of the expert survey was purposefully designed to ensure both analytical rigor and a smooth respondent experience. Building on the guiding principles from the previous section, the questionnaire follows a modular, conditional logic structure, which dynamically adapted to each respondent's background and experience level. This design reduced respondent fatigue, improved data quality, and enhanced the usability of the instrument for the expert participants, all of whom were actively or historically involved with the reuse methodology evaluated in this study.

Survey Flow and Participant Experience

The survey comprised several thematic segments, as visualized in the flow diagram shown in the next figure. Participants were first presented with a general experience section, followed by questions related to their previous engagement with solution reuse and with the specific reuse methodology introduced and evolved by the author. Conditional logic was applied in this section: if participants had prior reuse experience outside the framework, they were routed through a short additional sequence inquiring about their role (user or responsible engineer) and whether they had performed any evaluations of those reuse efforts. If so, they were asked to specify evaluation criteria from a predefined list with the option to add free-text responses. However, as reported later, no participant had conducted a structured evaluation, despite several reporting they had been “*in charge*” of previous reuse efforts.

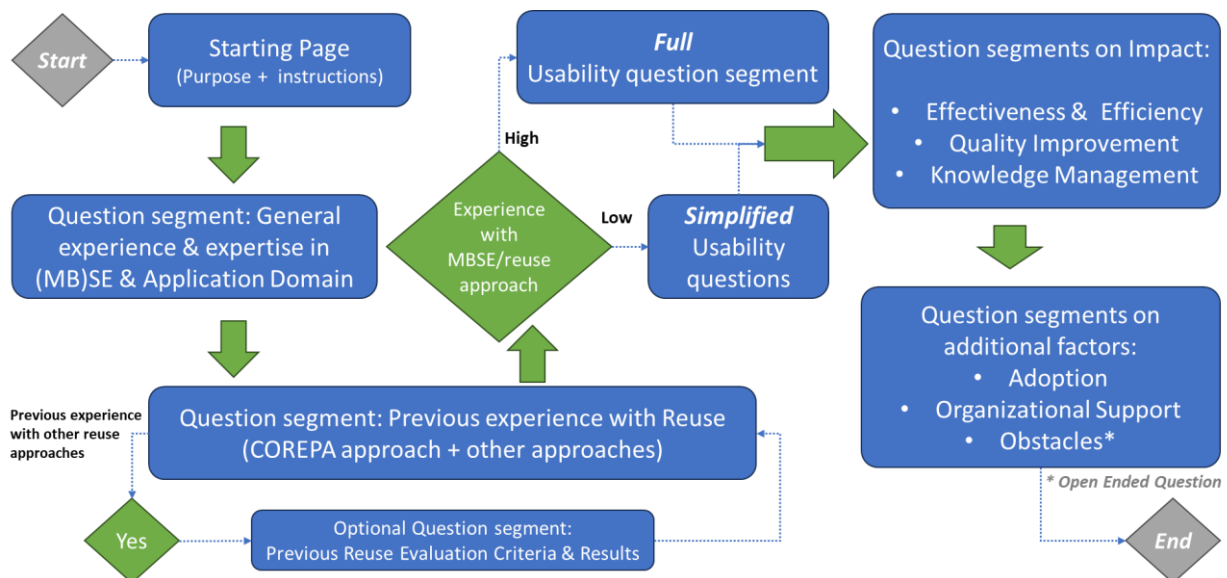


Figure 49: Diagram of Survey Structure and Flow as experienced by participants

Participants were then routed to one of two *usability* tracks based on their self-declared experience with the reuse method and relevant technologies. Those with low experience were directed to a simplified four-question usability segment, whereas more experienced respondents completed a three-question segment with more consolidated, higher-level statements. This approach

minimized interpretive error and response fatigue among less experienced users, thereby enhancing internal validity as stated in [223] and [233]. The remaining sections – on *efficiency and effectiveness*, *knowledge management* and *quality impact*, *adoption potential*, and *organizational culture* and *support* – were presented uniformly across all respondents. All questions included a "Not applicable (N/A)" response option, allowing respondents with lower experience being to abstain from responding, further reducing respondent fatigue and safeguarding data integrity [225].

The final item was an open-ended question asking participants to state – in a word or phrase – what they perceived as the greatest obstacle to fully leveraging the reuse framework. This placement was intentional: placing open-ended questions at the end of structured questionnaires allows respondents to reflect more broadly without risking early response bias or influencing preceding quantitative responses [234]. To signal the value of critical feedback and minimize social desirability bias, example responses—including deliberately negative ones—were provided within the question prompt.

Analytical vs. Experiential Clustering

To maximize both response quality and analytical validity, the survey design deliberately differentiated between experiential clustering for participants and analytical clustering for post-hoc evaluation. While respondents experienced a smooth, cognitively intuitive flow—progressing from general experience to more specific usability, impact and organizational questions—individual questions were analytically re-clustered during evaluation to align with thematic research constructs such as usability, knowledge management impact, and organizational support. Thematically, the questions fall into the following analysis categories:

- Subjective self-assessments,
- Objective experience qualifiers,
- Exploratory reuse questions,
- Usability,
- Efficiency and effectiveness,
- Perceived impact on knowledge *sharing*, quality, and adoption,
- Perceived knowledge management and data access facility,
- Organizational support (forming the *OSI* – discussed in detail in section 7.3.2.7),
- One open-ended question on perceived obstacles, meant to enhance the results of the categories above or open new exploratory topics.

This distinction between *survey-as-experienced* and *survey-as-analyzed* follows well-established principles in survey methodology [224], where structuring the respondent's experience to minimize cognitive load and maintain thematic coherence improves response accuracy and reduces fatigue, as supported by Krosnick [235], and Tourangeau et al. [236]. Additionally, embedding analytically distinct questions within semantically neighboring sections served to

reduce priming effects and response biases, supporting a more neutral and reflective evaluation by participants [228].

Navigation Features and Clarity Mechanisms

The survey was delivered electronically using a professional-grade tool that incorporated several features known to improve data reliability and participant satisfaction. These included section headers and thematic segmentation, progress indicators, and explicit indicators for whether single or multiple responses were allowed for a given question. These elements have been shown to improve engagement and reduce dropout, especially in technical or domain-specific surveys involving professional respondents, per Couper et al. [237].

Survey Familiarity and Shared Vocabulary

Because all participants were drawn from within the same organization and selected based on their direct involvement with the reuse methodology over several years, the need for explanatory glossaries or terminological clarifications within the questionnaire was deemed unnecessary. Participants shared a high level of contextual familiarity with the concepts and tools discussed.

7.3.1.4 Survey questions and possible responses

Following the starting page shown in Figure 48, the survey questions were structured into the following segments from the perspective of the respondents, i.e., the experiential clustering (as shown in Figure 49). For the sections *efficiency and effectiveness*, *quality impact*, *adoption potential*, and *organizational culture*, the experiential and analytical clusterings map 1:1. For the *self-assessment* and *experience* sections, as well as the *knowledge management* section, the analytical cluster each question is mapped to is given in braces after each question. This mapping was not included in the actual survey and hence not visible to the respondents, and solely listed here for clear understanding of how the later evaluations were carried out.

General User Experience

This section covers the participants' general experience, as well as expertise with (Model-Based) Systems & Software Engineering, and the application domain of vehicle systems, vehicle architecture design and optimization.

Table 11: Survey questions regarding general user experience

Question	Possible Responses
How many years have you been working in the field of Systems & Software Engineering? { <i>Objective experience qualifier</i> }	< 4 years
	4-9 years
	10-15 years

	16+ years
What is your level of experience with the analysis, synthesis and optimization of architecture variants at vehicle level? [Up to 2 responses allowed] { <i>Objective experience qualifier</i> }	< 5 years
	5+ years
	I have been responsible for the architecture design of vehicle systems (and their deployment optimization) which have been successfully implemented and fielded.
What is your level of knowledge of MBSE (Model-Based Systems Engineering) in general? [Up to 2 responses allowed] { <i>Subjective self-assessment</i> }	New to the topic, especially in this context.
	Familiar with a modeling tool (e.g., Rhapsody/PREEVision) and language/method (e.g., SysML or Arcadia/Capella), i.e. can use it myself with little or no assistance to solve my primary use-cases.
	Power-user of a modeling tool (e.g., Rhapsody/PREEVision) / language (e.g., SysML or Arcadia/Capella), i.e. can use most tool features, customize functionalities to my needs, and help others get started with using the tool.
	Familiar with multiple modeling paradigms, tools and frameworks.
	I build/extend tools and develop modeling frameworks.
How long have you used MBSE (Model-Based Systems Engineering) in the context of function design and system architecture development at vehicle level? { <i>Subjective self-assessment</i> }	< 3 years
	4-10 years
	11+ years

Experience with solution reuse in general and the COREPA methodology specifically

Table 12: Survey questions concerning experience with reuse

Question	Possible Responses
Have you previously (before your involvement with COREPA) been involved in any initiatives related to solution reuse or reference architectures (defined as modular	Yes, involving a pragmatic informal approach emphasizing efficiency.
	Yes, involving a systematic framework with enforcement of reuse rules.

building blocks with a guideline to create entire architectures using them) within the organization? { <i>Exploratory reuse question + control for usability flow</i> }	No.
If answer to previous question is YES, proceed, otherwise jump to NO section.	
Were you responsible for defining/introducing/enforcing the reuse or where you a user of the approach? { <i>Exploratory reuse question</i> }	Yes, I was responsible.
	No, I was a user.
If answer to previous question is YES, proceed, otherwise jump to NO section.	
Did you perform any evaluations of the reuse approach used? { <i>Exploratory reuse question</i> }	Yes.
	No.
If answer to previous question is YES, proceed, otherwise jump to NO section.	
Which criteria did you evaluate the reuse approach against? [Please select all that apply] { <i>Exploratory reuse question</i> }	Effectiveness
	Usability
	Knowledge Sharing
	Quality Improvement
	Integration with existing workflow/methodology
	Adoption
	Support and resources
	Other, please specify:.....
“NO” Section	
How long have you used the solution reuse/ reference architecture methodology introduced in CVA/COREPA? { <i>Subjective self-assessment</i> }	Less than 1 year.
	1-3 years.
	3+ years
If answer to previous question is “Less than 1 year”, automatically assume the answer to the next question is “1 reference vehicle” and end question segment, otherwise proceed to next question.	
How many different reference vehicle architectures did you develop using the methodology? { <i>Objective experience qualifier</i> }	1 reference vehicle.
	2-4 reference vehicles.
	4+ reference vehicles.

The remainder of the questionnaire followed the format of statements, to which the respondents had to provide varying degrees of agreement or disagreement. As previously stated, we avoided

“middle-of-the-road” evaluation options to force a trending solution and enhance the quality of the collected data. Additionally, we also provided respondents with lower experience with a “Not Applicable” option, as well as clear instructions to select this response if they do not feel comfortable making an assessment, to minimize their cognitive load and to avoid tainting the data.

As explained in the previous section, respondents who indicated having no prior experience with reuse were directed to the simplified 4-question usability segment, while all others were directed to the consolidated 3-questions version.

For brevity, we have consolidated the statements of each segment into one column, with the other column giving one instance of the possible responses according to a modified Likert-Scale, which were the same for all statements in each segment. For a full continuous version of the questionnaire, please see Appendix A4.

(Simplified) Evaluation of the methodology for usability

Table 13: Simplified usability questions

Statements	Possible Responses
▪ It was easy for me to learn how to use and apply the methodology. ▪ The support, instructions and guidance provided for the methodology are easy to understand. ▪ The documentation provided for the methodology is adequate. ▪ Reusing solutions has caused disruptions in our architecture design process.	Strongly agree
	Agree
	Somewhat agree
	Somewhat disagree
	Disagree
	Strongly disagree

(Consolidated) Evaluation of the methodology for usability

Table 14: Consolidated usability questions

Statements	Possible Responses
▪ It was easy for me to understand and apply the method using the support provided.	Strongly agree
	Agree
	Somewhat agree

▪ The method integrates (or is integrated) well with our development workflow. ▪ The method behaves (can be applied and used to generate results) consistently across different projects and scenarios. (Can be skipped if you have only used it in one project. - please select N/A)	Somewhat disagree
	Disagree
	Strongly disagree
	N/A – cannot make a statement

Efficiency and effectiveness

Table 15: Survey questions on efficiency and effectiveness

Statements	Possible Responses
▪ The method has increased the efficiency of reusing existing knowledge for creating and exploring architecture solutions. ▪ Reusing solutions has saved time in the architecture development process. ▪ The results of applying the method provide insight to my stakeholders and capture automotive solution architecture knowledge within the organization.	Strongly agree
	Agree
	Somewhat agree
	Somewhat disagree
	Disagree
	Strongly disagree
	N/A – cannot make a statement

Knowledge Management

Table 16: Survey questions on knowledge management

Statements	Possible Responses
▪ The method promotes knowledge sharing among team members. <i>{Impact on knowledge sharing}</i> ▪ The method promotes knowledge sharing in the organization. <i>{Impact on knowledge sharing}</i> ▪ It was EASY to collect the information needed to properly apply the methodology and	Strongly agree
	Agree
	Somewhat agree

leverage its strengths. { <i>Knowledge sharing and data access facility</i> }	Somewhat disagree
▪ The team was able to collect ALL of the information needed to properly apply the methodology and leverage its strengths. { <i>Knowledge sharing and data access facility</i> }	Disagree
▪ It was possible to collect ENOUGH information to leverage some strengths of the methodology and create value. { <i>Usability bonus question</i> }	Strongly disagree
▪ Better access to data would vastly improve my ability to leverage the reuse methodology to easily generate architecture solutions. { <i>Knowledge sharing and data access facility</i> }	N/A – cannot make a statement

Quality Improvement

Table 17: Survey questions on the impact of the method on quality improvement

Statements	Possible Responses
- Reusing existing architecture knowledge has led to improvements in the quality of our architecture solutions. - The reuse approach has helped in identifying issues early and increasing confidence in the architecture results.	Strongly agree
	Agree
	Somewhat agree
	Somewhat disagree
	Disagree
	Strongly disagree
	N/A – cannot make a statement

Adoption

Table 18: Survey questions on the impact of the method on adoption potential

Statements	Possible Responses
I would recommend the methodology to other teams in our organizations trying to improve their architecture knowledge capturing and reuse.	Strongly agree
	Agree
	Somewhat agree

It is likely that the method will be adopted by other teams addressing similar issues.	Somewhat disagree
	Disagree
	Strongly disagree
	N/A – cannot make a statement

Organizational Culture and Support

Table 19: Survey questions regarding the perceived organizational support

Statements	Possible Responses
- Our organizational culture encourages collaboration and knowledge sharing. - Leadership supports and promotes initiatives related to solution reuse, concretely by providing resources and timelines allowing investment in reuse infrastructure, deployment and application.	Strongly agree
	Agree
	Somewhat agree
	Somewhat disagree
	Disagree
	Strongly disagree
	N/A – cannot make a statement

Open-Ended Suggestions for Improvement

If you used only one word or phrase to describe the most pressing obstacle for leveraging the approach to capture and reuse architecture knowledge in the organization and create new solutions, what would it be? (e.g., missing tool support, it's a bad approach, too complicated, ...)
--

7.3.2 Survey Scoring and Evaluation Scheme

7.3.2.1 Scoring Scale for Survey Questions

To quantify expert responses systematically, each predefined statement response was assigned a numerical score as follows:

- **Strongly Agree:** 6 points
- **Agree:** 5 points
- **Somewhat Agree:** 4 points

- **Somewhat Disagree:** 3 points
- **Disagree:** 2 points
- **Strongly Disagree:** 1 point
- **Not Applicable (N/A):** Omit question from scoring calculations

This scoring scale is adapted from standard Likert⁴⁴ scale evaluation practices, modified here to exclude a neutral midpoint intentionally to encourage respondents to indicate a clear preference [223], while maintaining the recommended symmetry between individual responses to minimize extremity bias.

7.3.2.2 Calculation of Survey Result Scores – the *Metrics*

To evaluate the survey results comprehensively, we aggregated the scores by summing each respondent's points across all statements within each evaluation segment into the following *Metrics: Usability, Effectiveness and Efficiency, and Impact* – the latter divided across 3 subsegments: *Knowledge Management, Quality Improvement, and Adoption*. We then calculated the mean score for each segment across respondents to enable comparative analysis, with higher mean segment scores indicating stronger agreement and thus more positive evaluations by the respondents.

7.3.2.3 Handling of "Not Applicable" (N/A) Responses

The "N/A" option allows respondents lacking sufficient experience to abstain from providing potentially unreliable assessments, thereby safeguarding data integrity [225]. These responses will be excluded from the scoring computation, and only valid numerical responses will be averaged, ensuring scores accurately represent informed judgments.

7.3.2.4 Weighting of Expert Responses

Chen et al. [230] noted that as not all experts contribute equally – due to varying expertise, background, and experience – equal weighting of all experts' responses can dilute the validity of findings, especially when expertise varies widely among participants. They note that to enhance evaluation reliability, responses can be weighted based on the experts' self-assessed experience captured in the initial survey segments.

Weighted mean scores (WMS) can be calculated as follows:

$$WMS = \frac{\sum (Individual\ Response\ Score_i \times Individual\ Expert\ Weight_i)}{\sum (Individual\ Expert\ Weight_i)}$$

Equation 16: Calculation of Weighted Mean Scores for each survey section result

⁴⁴ Rensis Likert. 1932. "A Technique for the Measurement of Attitudes". Archives of Psychology. 140: 1–55.

7.3.2.5 Calculation of individual Expert Weighting Factors

We first calculated expert weighting factors by combining the scores of self-assessment questions from the "General User Experience" and "Solution Reuse and COREPA methods experience" sections, with the assumption that higher expertise scores imply greater expertise and should carry proportionately more weight in aggregated analyses.

However, as surmised by Cooke and Goosens [238], self-assessments are prone to various biases, including overconfidence, social desirability, and self-enhancement. These biases can lead to discrepancies between perceived and actual expertise, with studies [230], [239] showing that individuals often overestimate their abilities, which can distort survey results if not moderated by objective data, such as years of experience, educational background, or number of publications/patents on a subject matter, which provide tangible evidence of an expert's qualifications.

Indeed, our analysis of the responses to the survey's self-assessment questions by the pilot group revealed a notable tendency among experts to overestimate their experience when confronted with more subjective or ambiguously defined questions (e.g., '*How well do you know topic X?*' or '*How long have you been using our method?*'). This tendency is interpreted not as an indication of intentional overstatement or bad faith, but rather as an unintentional cognitive bias, likely influenced by social desirability and *presentism* – the tendency to overemphasize recent or salient experiences. To address this, we introduced additional, objectively framed questions (e.g., '*How many years of professional experience do you have with topic X?*'). Together with qualified response options (e.g., not just years of experience, but a *significant type of experience* for bonus points) this was designed to reduce interpretive variability and provide a more stable basis for calibrating self-reported expertise.

Consequently, combining subjective self-assessments with objective metrics is essential for deriving calibrated expert weightings in surveys. The Classical Model of structured expert judgment integrates experts' performance on calibration questions to assign weights, thereby enhancing the reliability of aggregated judgments [238]. Additionally, Liu and Hu [240] propose a method that combines subjective weights with objective weights, optimizing the variance to achieve a balanced weighting scheme. Chen et al. [230] also recommend integrating subjective and objective metrics to derive a greatly enhanced expert weighting approach and propose a two-tiered method for deriving calibrated expert weights.

To apply these recommendations, we decided to employ *two individual weighting factors for each expert*. A **Subjective Weighting Factor**, calculated based solely on the responses to subjective survey questions; and a **Calibrated Expert Weighting Factor**, calculated by following this formula:

$$\text{Calibrated Expert Weighting Factor (CEWF)} = (0.6 \times \text{Subjective Weighting Factor}) + (0.4 \times \text{Combined Objective Experience Factor})$$

Equation 17: Calculation of Individual Expert Weights

Where:

- the **Subjective Weighting Factor** is calculated by normalizing the combined mean response scores (each ranging 1 – 5, indicating increasing expertise levels) to 3 subjective survey questions: “*What is your level of knowledge of MBSE (Model-Based Systems Engineering) in general?*”; “*How long have you used MBSE (Model-Based Systems Engineering) in the context of function design and system architecture development at vehicle level?*”; and “*How long have you been using the solution reuse/ reference architecture methodology introduced in COREPA?*”.
- the **Combined Objective Experience Factor** is calculated by normalizing the combined mean response scores (each also ranging 1 – 5, indicating increasing expertise levels) to 3 survey questions of a more objective and quantifiable nature: “*How many years have you been working in the field of Systems & Software Engineering?*”; “*What is your level of experience with the analysis, synthesis and optimization of architecture variants at vehicle level?*”; and “*How many different reference vehicle architectures did you develop using the methodology?*”.
- the coefficients for subjective and objective factors (0.6 and 0.4, respectively) were empirically derived from the variance observed in the self-assessment responses of the three pilot participants, in accordance with [240].

To assess the impact of self-assessment on expert weightings while minimizing potential distortions, **three sets of metrics** will be computed, presented and used in the evaluations: (1) *unweighted baseline metrics* – to deliver an unfiltered raw participant ratings, (2) *metrics adjusted solely using subjective self-assessment weightings* – as controls for perception bias, and (3) *metrics adjusted using the calibrated expert weighting factor* introduced above, integrating both subjective and objective components to increase confidence in the ratings.

7.3.2.6 Scoring scale for self-Assessment relevant questions

The question whose responses were relevant for the self-assessments were given scores ranging 1 – 5; reflecting increasing expertise levels. The questions, responses and corresponding scores are shown in the following table.

Table 20: Corresponding numerical scores for the responses to survey questions relevant for self-assessment

Question	Possible Responses	Score (1-5)
How many years have you been working in the field of Systems & Software Engineering?	< 4 years	1
	4-9 years	2
	10-15 years	3.5
	16+ years	5

What is your level of experience with the analysis, synthesis and optimization of architecture variants at vehicle level? [Select up to 2 responses]	< 5 years	1
	5+ years	3.5
	I have been responsible for the architecture design of vehicle systems (and their deployment optimization) which have been successfully implemented and fielded.	+1.5
What is your level of knowledge of MBSE (Model-Based Systems Engineering) in general? [Select up to 2 responses]	New to the topic, especially in this context.	1
	Familiar with a modeling tool (e.g., Rhapsody/PREEVision) and language/method (e.g., SysML or Arcadia/Capella), i.e. can use it myself with little or no assistance to solve my primary use-cases.	2
	Power-user of a modeling tool (e.g., Rhapsody/PREEVision) / language (e.g., SysML or Arcadia/Capella), i.e. can use most tool features, customize functionalities to my needs, and help others get started with using the tool.	3
	Familiar with multiple modeling paradigms, tools and frameworks.	4
	I build/extend tools and develop modeling frameworks.	+1
How long have you used MBSE (Model-Based Systems Engineering) in the context of function design and system architecture development at vehicle level?	< 3 years	1
	4-10 years	3
	11+ years	5
How long have you been using the solution reuse/ reference architecture methodology introduced in COREPA?	Less than 1 year.	1
	1-3 years.	3
	3+ years	5
How many different reference vehicle architectures did you develop using the methodology?	1 reference vehicle.	1
	2-4 reference vehicles.	3
	4+ reference vehicles.	5

7.3.2.7 Consideration of the impact of environmental factors on survey results

Responses to the "Organizational Culture and Support" segment provide context on environmental factors influencing respondents' perceptions and potential biases. Generally, such responses should be considered supplementary metrics for *qualitative interpretation*, rather than inputs for direct numerical weighting. This guidance is supported by Schein [241], who emphasizes that organizational culture data are best used as interpretive context, and by Denison and Mishra [242], who argue that cultural indices should not be treated as direct statistical inputs but rather as explanatory factors.

Nevertheless, for quantitative analysis integration purposes, an **Organizational Support Index (OSI)** may still be calculated by averaging respondents' ratings. This index can then be used to qualitatively assess alignment or discrepancies between perceived organizational support and the primary evaluation segments, aiding contextual interpretation [243]. In this context, large discrepancies indicate potential organizational barriers or facilitators, informing practical interpretations and recommendations. Additionally, this index can also be used to provide additional visualization of the impact of these environmental factors on the overall survey results.

To facilitate interpretation, the OSI can be evaluated around the midpoint of our Likert scale (3.5) as a reference:

- Scores < **3.5** indicate a perceived **unsupportive** environment.
- Scores > **3.5** indicate a perceived **supportive** environment.

This normalization does not alter expert weights but serves as a *lens to interpret the context* behind expert evaluations. For example, a high score from a respondent in a low-support environment may indicate strong framework performance under challenging conditions, while the opposite could suggest results influenced by a highly supportive context. This approach preserves methodological clarity and aligns with best practices in mixed-methods organizational research, as discussed in [241] and [242].

7.3.2.8 Numerical interpretation of survey results

We will interpret the segment (*usability*, *efficiency/effectiveness*, or *impact*) mean scores and subsequent weighted scores using the following guidelines:

- **>4.5 – 6.0:** This score range for a metric indicates highly positive evaluations for that segment and strong expert endorsement.
- **>3.5 – 4.5:** This score range for a metric indicates moderately positive evaluations for that segment and can be used to identify areas for potential improvement.
- **1.0 – 3.5:** This score range for a metric indicates overall unfavorable evaluations for that segment. Significant revision and improvements would be recommended for these areas.

7.3.3 Results, Observations and Findings

This section presents and analyzes the results of the expert survey conducted at the conclusion of the longitudinal industrial case study. It is structured into five subsections: (1) presentation of the quantitative results; (2) direct interpretation of survey findings; (3) broader implications for industrial practice; (4) robustness and sensitivity checks across groups and participant characteristics; and (5) consistency and outlier analyses to ensure internal data validity.

7.3.3.1 Quantitative Results

Following the survey design, pilot validation, and phased execution, as well as the evaluation and scoring schemes described earlier, this section presents the quantitative results of the expert survey. Both raw and weighted data are systematically reported to ensure transparency and reproducibility.

Detailed tables containing per-expert aggregated numerical scores, along with the raw pseudonymized Likert-scale responses, are provided in **Appendix A5**. In this section, cumulative results are summarized numerically, presented in tabular form, and synthesized through appropriate visualizations (e.g., clustered 3D column charts, heatmaps, and boxplots) to facilitate clear interpretation and subsequent analysis [223], [225].

Aggregated metrics are reported per expert for each evaluation segment: *Usability* and *Efficiency & Effectiveness*, and the *Impact on Knowledge Sharing, Quality Improvement, and Adoption*, as well as the perception of *Organizational Support*. To protect participant anonymity, segment-level results are presented without linking scores to individual survey items.

As introduced in Section 7.3.2.5, three sets of evaluation results are presented: (1) unweighted baseline metrics, (2) metrics adjusted using subjective self-assessment weightings, and (3) metrics adjusted using a calibrated expert weighting factor that integrates both subjective and objective experience measures.

In addition, cumulative results are presented separately for two groups: the **Core Group** (comprising experts actively engaged with the reuse approach at the time of the survey, with five responses from eight contacted) and the **Consolidated Group** (comprised of the *Core Group*, plus the *Extended Group* of experts with prior, but not current, involvement, with 10 responses from 17 contacted).

Finally, supplementary visualizations, together with interpretation references linked to the Organizational Support Index (OSI), are provided in the ensuing subsections to support a comprehensive analysis of the findings.

Cumulative Results

Table 21: Final Results of Survey

	Core Group results			Combined Results		
Normalized Self-Assessment Experience Score	0.69			0.61		
Normalized Objective Experience Score	0.55			0.56		
Calibrated Experience Weighting Factor	0.64			0.59		
Organizational Support Index (OSI) (3.5 = neutral, higher is positive)	3.30			3.40		
	Unweighted score	Self-assessed	Calibrated weighting	Unweighted score	Self-assessed	Calibrated weighting
Usability	4.23	4.43	4.57	4.41	4.51	4.57
Effectiveness and Efficiency	4.67	4.79	4.89	4.71	4.82	4.87
Combined Impact	4.69	4.81	4.89	4.75	4.83	4.90
Impact sub-segments:						
Knowledge Sharing	4.70	4.90	5.02	4.80	4.94	5.00
Quality Improvement	4.88	4.93	4.96	4.86	4.93	5.02
Adoption Endorsement	4.50	4.61	4.68	4.60	4.63	4.67

The table above presents aggregated evaluation metrics for both the Core Group (n = 5) and the Combined Group (n = 15), across three weighting schemes: unweighted scores, scores weighted by self-assessed expertise, and scores further calibrated through combined subjective and objective experience metrics. Segment-level metrics are shown for *Usability*, *Effectiveness and Efficiency*, and the perceived *Impact* of the reuse approach on *Knowledge Sharing*, *Quality Improvement*, and *Adoption*. In addition, normalized experience scores and the *Organizational Support Index* (OSI) are reported to provide contextual insight into respondent expertise and organizational alignment. This structured view enables clear cross-group and cross-metric comparisons. Detailed interpretation follows in the next section 7.3.3.2.

Aggregated Segment Results

Figure 50 provides a detailed visual comparison of *Usability* scores across all 15 experts, segmented into unweighted values, self-assessment weighted scores, and fully calibrated weighted scores. This visualization illustrates the variance in perceived usability of the reuse approach and how expert weighting affects the overall evaluation outcome; both positively (e.g., *Expert 4*), as well as negatively (e.g., *Expert 15*). Each clustered column group represents a single expert's responses, enabling fine-grained comparison across evaluation schemes.

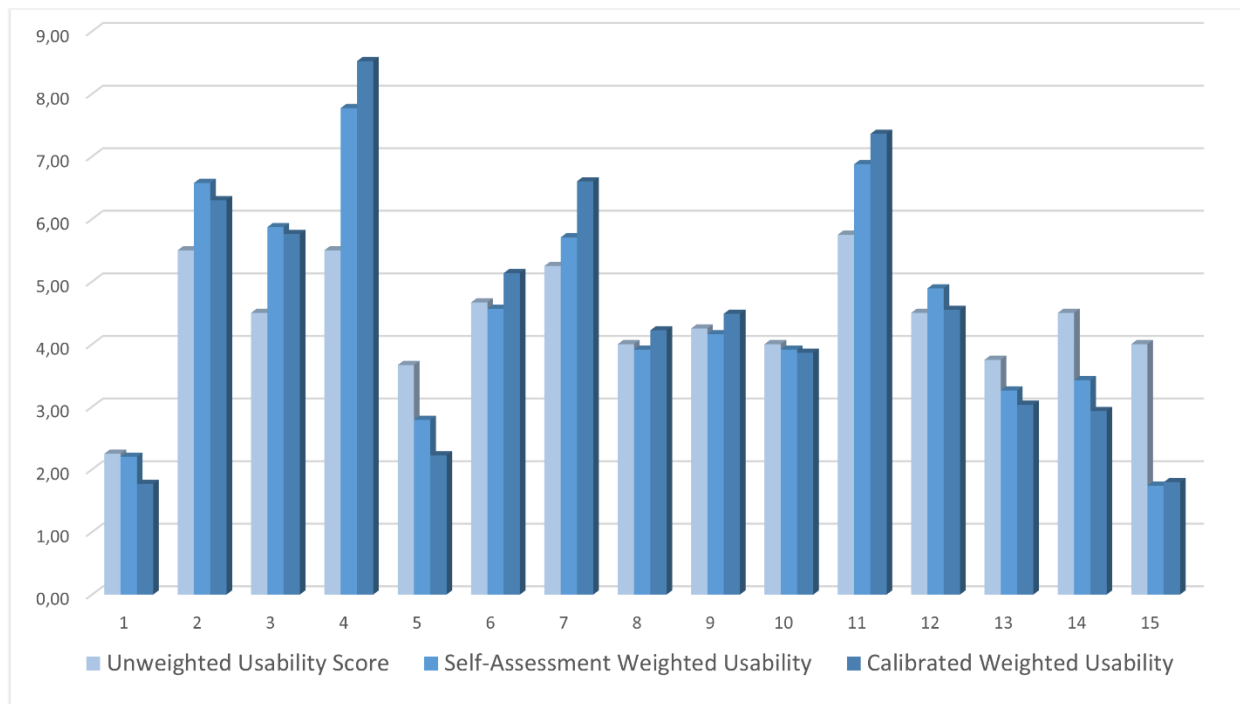


Figure 50: Weighted and unweighted usability evaluations across all 15 experts

For consistency and brevity, only the *Usability* segment aggregated results are shown here; equivalent charts for all other metric segments (*Effectiveness and Efficiency*, *Knowledge Sharing*, *Quality Improvement*, *Adoption*) are provided in **Appendix A5.3**.

To conclude the presentation of raw and weighted results, the following integrative visualization provides a consolidated view of **unweighted** expert-level evaluations across all five core metric dimensions – *Usability*, *Effectiveness & Efficiency*, *Knowledge Sharing*, *Quality Improvement*, and *Adoption*.

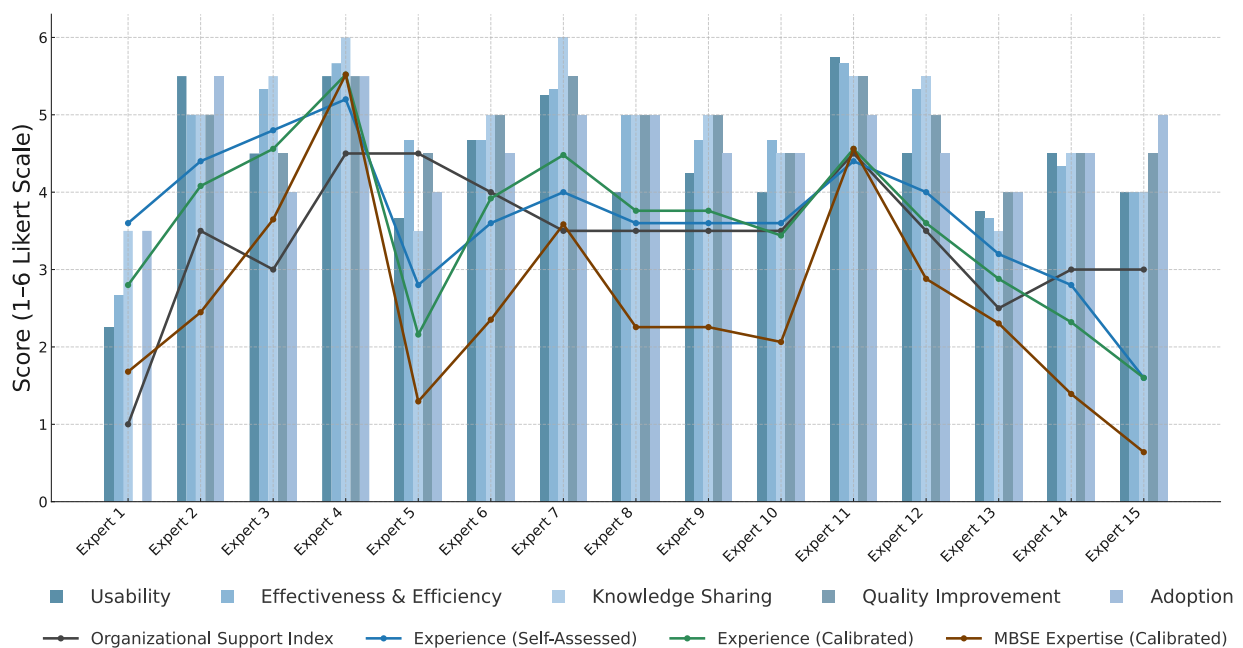


Figure 51: Unweighted expert evaluations overlaid with contextual factors

These segment scores are depicted as clustered bars for each expert. Superimposed line graphs further **contextualize the scores by plotting key influencing factors** captured through the survey: the *Organizational Support Index* (OSI), the *self-assessed and calibrated experience weights*, and the *calibrated MBSE expertise* score. This juxtaposition facilitates an initial visual inspection of potential relationships between individual perceptions and contextual factors. While no interpretive conclusions are drawn here, the figure serves as a foundation for the analytical and correlation-based insights discussed in the following sections.

7.3.3.2 Direct Interpretation of Quantitative Results

Building on the preceding presentation of raw and weighted quantitative results, this section offers a direct interpretation of the survey findings, with visualizations that illustrate overall trends and contextualize individual expert evaluations. It also sets the stage for exploring deeper organizational implications and actionable insights derived from correlation analysis and qualitative expert feedback.

The **survey results present a consistently strong expert evaluation of the reuse methodology across all measured dimensions**, with weighted and unweighted segment scores indicating generally high agreement on usability, effectiveness, impact, and adoption potential. Notably, the only metric exhibiting a very slight deviation from this pattern was usability – specifically in the unweighted scores (4.23) and in the self-assessed weighted scores (4.47) *within the Core Group*. The extended group had a significantly more positive unweighted evaluation, bringing the combined unweighted score to (4.41). Ultimately, this **deviation** is completely **mitigated** when considering the **calibrated weighted results**, which **show identical scores (4.57) across both expert groups**.

Across all other evaluation segments, the methodology received uniformly high scores: unweighted metrics ranged from **4.5 to 4.88**, **self-assessment weighted** scores **between 4.51 and 4.94**, and **calibrated weighted** scores peaked **between 4.57 and 5.02**; out of a **maximum score of 6**. This strongly confirms the overall robustness, utility, and organizational impact of the approach as perceived by experienced experts – a rare level of agreement in industrial surveys involving subjective expert judgment.

The boxplot shown in the next figure provides a distributional overview of responses across key survey segments, revealing both areas of consensus and those with greater divergence among participants. As expected from earlier analyses, the **lowest median scores** and **strongest consistency** occur in relation to the exploratory questions around *ease of access to necessary data*, confirming that poor data availability was a major obstacle across the board. In stark contrast, nearly unanimous agreement was observed for the statement “*Better access to data would vastly improve impact*”, underscoring the practical limitations faced by participants when attempting to apply the reuse methodology. Among the evaluated metrics, **usability and knowledge sharing exhibit the broadest interquartile ranges**, indicating varying user experiences, likely due to differences in MBSE maturity and organizational context. Conversely, **impact on quality improvement shows a highly condensed distribution around high scores**, pointing to strong consensus about the method’s contribution in this area. Similarly, evaluations of *effectiveness and efficiency* and *adoption endorsement* show tight clustering around agreement, reinforcing the

overall positive assessment of the method's value. This visual dispersion supports earlier statistical findings and sets the stage for deeper exploration of the contextual factors influencing metric variability.

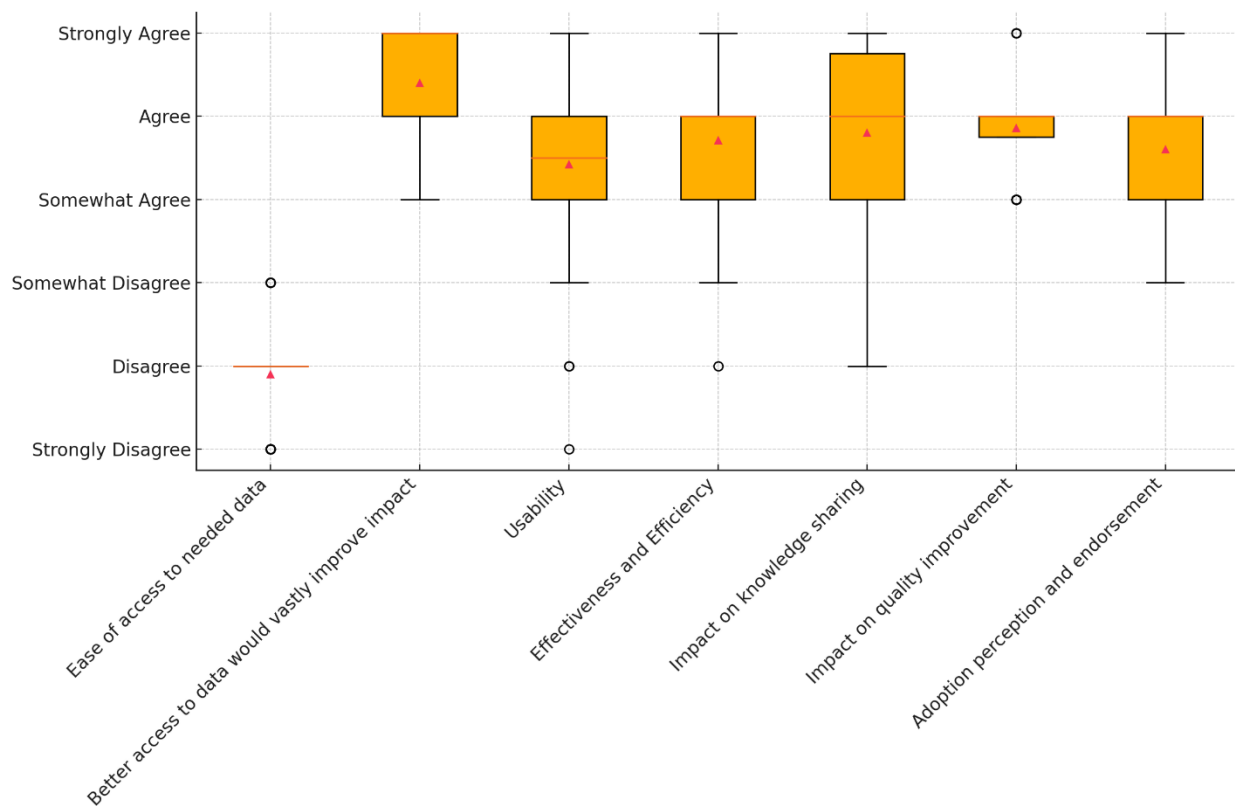


Figure 52: Boxplot of response distributions in key survey segments

Furthermore, the integrative chart (Figure 51), shown at the end of the previous section, juxtaposes each expert's segment evaluations with contextual weighting factors (Organizational Support Index, MBSE expertise, and calibrated experience), highlighting strong internal consistency. Additionally, it provides a clear visual indication of potential relationships between contextual characteristics and survey outcomes. These relationships are investigated more rigorously in the following subsection through correlation analysis.

7.3.3.3 Broader Implications for Industrial Practice and Recommendations

Correlations

The most striking insight from the correlation analysis lies in the strong positive associations between MBSE expertise and every segment score. The heatmaps in Figure 53 and Figure 54 (Core Group and Full Set, respectively) show high correlations between calibrated MBSE expertise and key metrics – particularly usability ($r=0.67$ and $r=0.67$), effectiveness & efficiency ($r=0.57$ and $r=0.63$), and knowledge sharing ($r=0.78$ and $r=0.83$). This reinforces the hypothesis that MBSE capability is a central enabler for solution reuse: those with higher MBSE literacy not only perceive the methodology as more usable but are more likely to report concrete benefits in application.

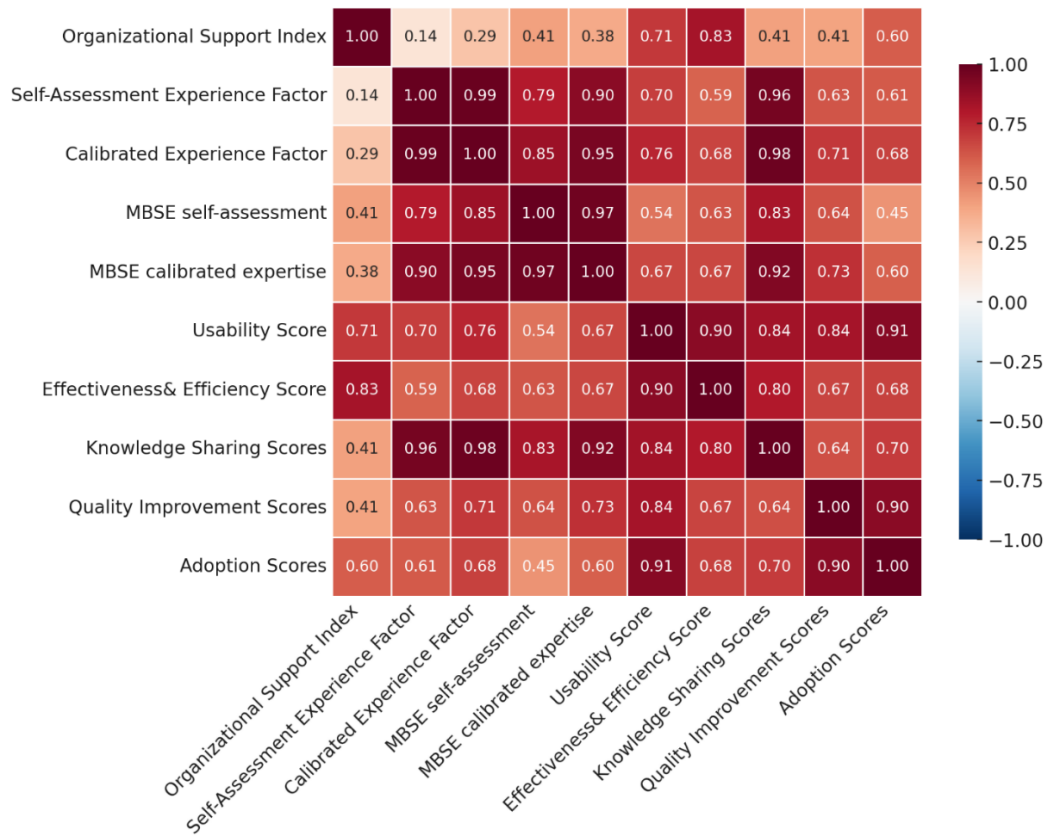


Figure 53: Core Group Correlation Heatmap for Survey Metrics and Key Contextual Factors

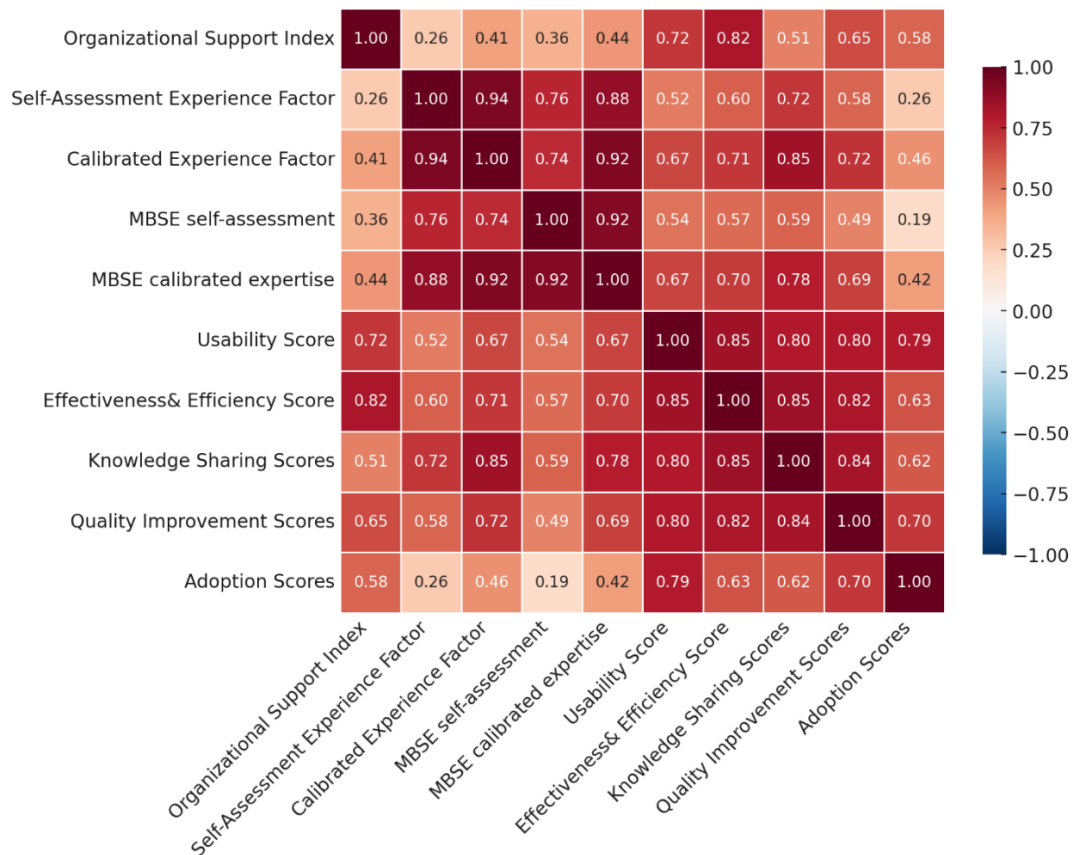


Figure 54: Correlation Heatmap across all 15 Experts for Survey Metrics and Key Contextual Factors

The Organizational Support Index (OSI) – which aggregates perceptions of managerial support and process alignment – also demonstrates meaningful correlations, particularly with perceived usability ($r=0.72$ Core, $r=0.71$ Full Set) and effectiveness ($r=0.82$ and $r=0.83$). This indicates that organizational readiness and encouragement for reuse strongly influence experts' willingness and ability to apply the methodology effectively.

Interestingly, the correlations between reuse evaluations and objective experience (e.g., years of application) were weaker than those tied to MBSE competence, suggesting that **it is not tenure alone, but domain-specific system modeling literacy, that drives successful adoption**. These findings provide robust support for targeted upskilling initiatives focused on MBSE techniques rather than relying solely on experience accumulation.

As the distribution shown in the boxplot earlier (Figure 52) suggests, respondents' views on knowledge-sharing impact were more varied than for other metrics, hinting at organizational and cultural variation in how reuse benefits are realized. This observed variation fits with the respondents' affiliations with different parts of a large organization with primarily decentralized operations, impacting individual experiences and perceptions.

Lack of Evaluation of Prior Reuse Activities: A Revealing Gap

One of the most striking findings from the exploratory section of the survey concerns the **absence of prior formal evaluation of reuse efforts** within the participating experts' past experiences. Of the fifteen experts surveyed, seven reported no prior experience with solution reuse initiatives. Among the remaining eight, only a single participant indicated involvement with a systematic reuse framework; the others referred to *informal, pragmatic efforts* focused primarily on efficiency. Most notably, **none of the respondents – despite four having been directly responsible for the definition or enforcement of reuse approaches – reported having conducted any structured evaluation of these efforts**.

This absence of evaluation suggests a broader issue in industrial systems engineering practice: although reuse is widely acknowledged as an important efficiency lever, **systematic assessment of reuse initiatives appears to be the exception rather than the norm**. This finding aligns with prior literature identifying a general lack of performance metrics and validation practices in early-stage reuse initiatives, especially in domains where legacy systems, fragmented tooling, and organizational silos inhibit holistic oversight, as reported by Frakes & Kang [244] or Li et al. [245]. Furthermore, it underlines the critical need for **integrated measurement frameworks** and **formalized success criteria**, which are often recommended in academic models of reuse maturity, such as that of Wu et al. explained above [185], but rarely implemented in practice.

The fact that none of the experienced reuse participants could recall applying even basic evaluation criteria – such as usability, effectiveness, or integration with workflow – underscores a cultural and methodological blind spot that hampers organizational learning and sustained reuse capability development. It reinforces the broader implication that **without structured feedback and systematic evaluation mechanisms, reuse efforts risk remaining ad hoc and unscalable**, limiting their strategic value. In contrast, the structured evaluation enabled by the reuse framework introduced in this thesis – particularly the expert survey and its quantitative rigor – represents a replicable model for elevating reuse from informal practice to an evidence-driven discipline.

Challenges and Obstacles for Reuse in Practice: A View from the Trenches

The quantitative evaluation is further reinforced by open-ended responses to the final survey question regarding perceived obstacles. Nearly all respondents emphasized one critical challenge: **the lack of access to (re)usable, structured knowledge and architectural data within the organization**. This consistent qualitative theme is mirrored in the quantitative data: as shown in Figure 55, all fifteen experts strongly agreed that better access to relevant internal knowledge would have significantly improved the methodology’s impact – regardless of their experience level or OSI score.

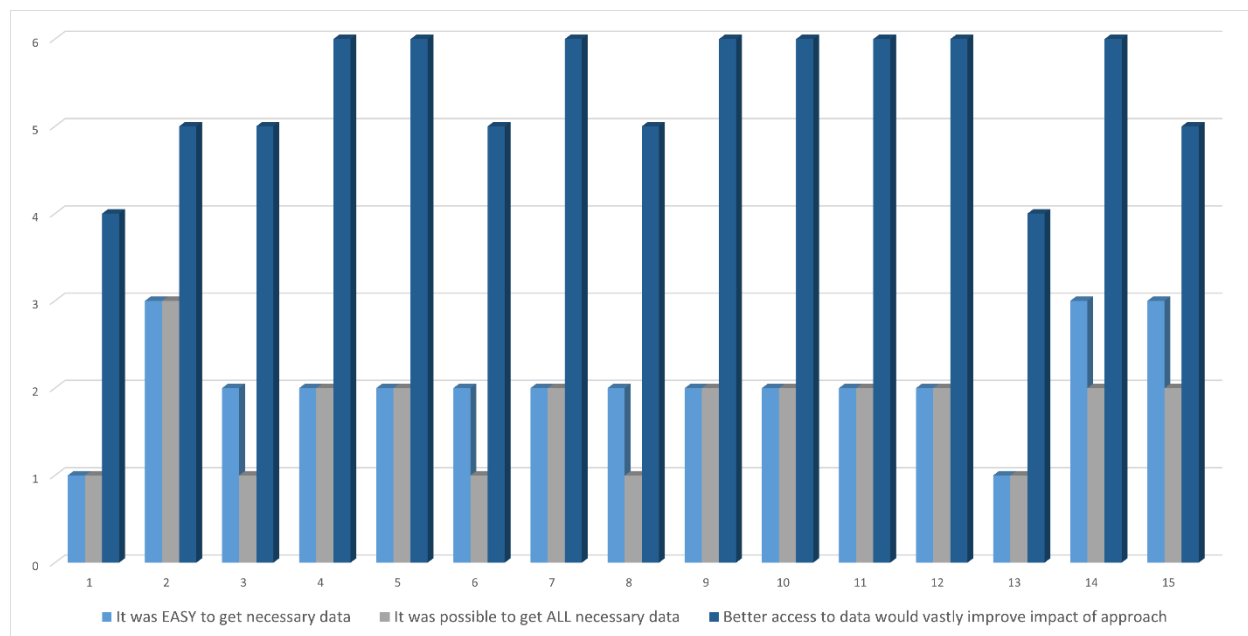


Figure 55: Expert respondents' evaluation of knowledge management and facility of access to necessary data to leverage approach

Notably, nine out of fifteen responses referenced this data access issue explicitly, citing challenges such as “missing architectural knowledge from business units,” “lack of support in gathering needed information,” or simply “data access missing.” A second cluster of comments pointed to insufficient Model-Based Systems Engineering (MBSE) maturity, with remarks including “lack of MBSE adoption,” “missing internal champions,” and “MBSE gap.” Two isolated comments highlighted organizational resistance and time constraints, respectively – yet even these echoed broader themes of cultural and structural friction. A final highly experienced respondent used his response to complement⁴⁵ the method, reinforcing that the primary limiting factor was not the method itself, but the organizational readiness to support and scale it.

This insight exposes a critical structural barrier to reuse: while methodological soundness is essential, it is not sufficient. In large, siloed organizations, achieving meaningful reuse requires not only technical methods but also organizational commitment to architectural transparency, harmonized modeling practices, and structured knowledge management.

⁴⁵ “one order of magnitude more efficient than anything else in the company”

These lessons align closely with and extend the findings of Wu et al. [182], whose academic work on reuse maturity modeling identifies three foundational requirements for successful reuse in MBSE contexts: process formalization, structured reuse integration, and a commitment to continuous improvement. During the longitudinal case study, these recommendations were independently implemented by the author and colleagues through iterative application, tooling enhancements, and targeted organizational interventions – often preceding or exceeding said recommendations.

Moreover, the efforts extended beyond the scope of Wu et al.’s framework, encompassing organizational alignment, upskilling programs, strategic tooling development, and practical governance integration – as discussed in detail in section 7.2.3.4, thus demonstrating a broader and more integrated approach to institutionalizing reuse in a complex industrial environment.

However, we wish to explicitly note Wu et al.’s contributions proposing a phased maturity assessment model for evaluating and enhancing an organization’s capability to support reuse in the context of MBSE, which offers a well-structured and practically relevant framework. Their model, expanded on in a 2021 publication [185], distinguishes multiple levels of reuse maturity (shown in the next table), each defined by clear evaluation criteria and increasing degrees of formalization and integration into engineering processes. The findings and recommendations are largely supported by the author’s own observations and practical experiences, as well as the findings of the survey results, affirming the relevance of their approach.

Table 22: Maturity levels for assessing reuse capabilities in organizations according to Wu et al. [185]

		Axis		
		Identification	Library classification	Reuse
Maturity level	0	No identification of reusable elements	No library classification	No reuse from previous projects
	1	Opportunistic identification of reusable elements from previous projects, without method (uncomplete vision)	Awareness: "I already saw something like this"; Opportunistic oral sharing of reusable elements: "we already done this way", use of paper board...	Opportunistic reuse by copy/paste from previous projects, without method; manual adaptation
	2	Planned identification of reusable elements, without method (uncomplete vision)	Planned sharing of formalized elements (communication, archiving...) identified as reusable (texts, models...)	Planned reuse by copy/paste from previous projects, without method; manual planned adaptation
	3	Defined identification method: classification in function of defined abstraction levels	Defined capitalization method: sharing organized around a sharing structure	Defined reuse method: defined selection of reusable elements and transitions between defined abstraction levels
	4	Quantified measure of defined identification method efficiency (identification time, costs...)	Quantified measure of defined capitalization method efficiency (classification time...)	Quantified measure of defined reuse method efficiency (direct reuse or adaptation time...)
	5	Optimization : continuous improvement of identification method	Optimization : continuous improvement of classification method	Optimization : continuous improvement of reuse method

Combined with the practical aspects of our work, Wu et al.’s maturity assessment model represents an excellent starting point for organizations seeking a structured way to introduce or advance reuse practices within Model-based Systems Engineering environments.

While the survey results paint a largely optimistic picture, the structural challenge of reuse overhead remains a persistent obstacle. **Achieving true reusability in practice demands substantial upfront investment** in abstraction layers, parametrization mechanisms, and architectural scaffolding – investments **which often conflict with project-driven cost and**

schedule pressures. In many cases, reuse goals are compromised when projects are forced to deliver prematurely, resulting in “platforms” that are essentially bespoke solutions with limited generalizability. This phenomenon, rooted in systemic organizational dynamics rather than methodological shortcomings, must be addressed at the strategic management level – a point further explored in the concluding chapter.

Survey Conclusion

Taken together, the empirical results and their alignment with practical interventions implemented over nearly a decade **illustrate not only the viability of the proposed reuse methodology but also its systemic impact when embedded within a strategically aligned organizational context.** The correlations between key contextual factors – such as MBSE competence and perceived usability, or organizational support and adoption outlook – further validate the critical role of capability development and cultural alignment in enabling successful reuse. These insights offer actionable guidance for practitioners seeking to institutionalize reuse within complex engineering environments. To reinforce the credibility of these findings, the following subsections assess the robustness, sensitivity, and consistency of the survey results before turning to an explicit discussion of potential threats to validity.

7.3.3.4 Robustness and Sensitivity Checks

The phased two-group survey design inherently supported robustness and sensitivity checks [225]. Initial analysis of the Core Group established baseline conclusions regarding method usability, effectiveness, and perceived organizational support. Subsequent analysis of the Extended Group – comprising respondents with varying degrees of recent involvement – enabled validation across different levels of familiarity and operational recency [224].

Comparison revealed only minor variations between the groups, with overall segment metrics remaining stable. Subtle shifts in correlations – primarily involving self-assessed competence and perceived usability or adoption – affirm the robustness of the findings. This consistency provides additional confidence in the validity of conclusions drawn from the survey [225].

Visual comparison of correlation matrices derived separately for the Core Group and the Combined Group (see Figure 53 and Figure 54) further supports the stability of relationships among evaluation metrics. Despite differences in group composition, the internal structure of associations – particularly between self-assessed expertise, perceived method usability, and organizational support – remained largely intact. This structural coherence suggests respondents, regardless of subgroup, conceptualized the framework’s value comparably, reinforcing the internal robustness of the findings [225], [224].

7.3.3.5 Consistency and Outlier Checks

Internal consistency of responses was systematically evaluated during data analysis. The clear survey structure and predefined scoring methodology allowed consistency to be readily assessed. Furthermore, the two-group phased design discussed in section 7.3.1.2 inherently enabled a deeper examination of response stability.

One question ("*Reusing solutions has caused disruptions in our architecture design process*"), planned to be qualitatively exploratory, was briefly considered for inclusion in the quantitative analysis, before a deliberate decision was made to exclude it from all analysis – both quantitative and qualitative. The reason for this complete exclusion was *interpretive ambiguity* – particularly across language and culture differences amongst the international participants – regarding whether disruption was seen *positively* (English speakers) or *negatively* (German speakers). This exclusion has negligible impact on the analysis in general, as this question was limited to the simplified Usability question set, which less than half of the respondents were exposed to. Additionally, its inclusion would have quantitatively skewed the results marginally upward. Moreover, its exclusion ensures a balanced spread between the two usability sections, with each now containing four questions: three distinct and one shared.

Only one individual-level inconsistency was identified: a participant reported minimal MBSE familiarity yet simultaneously indicated extensive application of the method. Due to an unintended breach of anonymity – initiated in a fortunate coincidence by the participant voluntarily disclosing their survey completion to the author immediately after submission – the inconsistency was confirmed through direct discussion. The participant acknowledged a mis-click, and with explicit consent, the response was corrected. Beyond this isolated event, no anomalies or outliers were observed, reinforcing dataset reliability, as discussed in [232] and [223].

Consistency was further assessed structurally by comparing the correlation architecture between the Core and Combined groups. Preservation of inter-metric relationships—such as strong links between reuse familiarity and adoption, and stable associations between MBSE competence and perceived effectiveness – suggests a coherent internal response pattern. No inversions or anomalous shifts in correlation directionality were observed [225], [223].

A supporting consistency check comes from the low ratings on the adequacy of the documentation provided for the methodology. These candid responses align with internal awareness of minimal documentation, confirming that respondents were not merely being polite and reinforcing the credibility of the overall feedback.

Minor attenuation of certain relationships (e.g., between reuse familiarity and method usability) was noted in the Extended Group compared to the Core Group. This likely reflects differences in operational immediacy rather than true inconsistency. The phased design's ability to surface such nuances adds analytical depth while affirming overall coherence and reliability [224].

7.3.4 Threats to Validity

7.3.4.1 Conclusion Validity

Threat:

The relatively small sample size (15 respondents from a population of approximately 25 experts) introduces concerns about the statistical reliability of the conclusions, as smaller samples increase the risk of random error influencing results.

Mitigation:

Although the sample is modest, the respondents constitute a majority (62.5% of active users, and 60% of all users) of all individuals exposed to or using the reuse framework in this highly specialized setting. Additionally, all respondents had a minimum of four years of experience in Software and Systems Engineering, and with four respondents (out of 15) having more than 16 years of experience, and three respondents with 10-15 years of experience, and assuming a normal population distribution among the other respondents – the median experience of the respondents exceeds 10 years; supporting their designation as *experts*. Literature emphasizes that achieving **high response rates among limited expert populations** significantly reduces random variability and increases conclusion validity [225]. Furthermore, aggregating expert opinions – especially weighted based on self-assessed expertise and calibrated with objective values – meaningfully enhances the reliability and validity of conclusions, even from smaller samples [230]. Therefore, the conclusion validity threat due to sample size is adequately mitigated by high response rates, high experience of the respondents and appropriate expert weighting.

7.3.4.2 Internal Validity

Threat:

There remains a potential internal validity threat stemming from respondent biases, especially *social desirability bias*, where participants might provide overly positive responses due to personal relationships or awareness of the survey author's identity and role, as well as a desire to portray the organization and its activities in positive light.

Mitigation:

To explicitly mitigate social desirability bias, the survey was fully anonymized, ensuring the author could not determine who had responded, respondents' identities, or link individuals to responses. Anonymization (insofar as it is possible in this setting) is known to significantly reduce the tendency for socially desirable responses by increasing participants' perceived confidentiality and comfort to express genuine opinions [228]. Moreover, the survey invitations were sent broadly and inclusively to all framework users, specifically including respondents who had previously expressed criticism or reservations about the methodology or had difficult personal relationships with the author. This inclusive strategy ensures the representation of critical perspectives and helps balance potentially overly positive feedback [225].

While personal messages from survey authors and researchers might inadvertently introduce a subtle emotional appeal potentially influencing respondents' neutrality, nevertheless, empirical evidence indicates that clearly stated personal requests for sincerity and neutrality enhance respondent authenticity by explicitly signaling that honest, even *negative*, feedback is welcome [226].

Given that the author is known in a leadership role to all participants, a decision was made to include a personal plea at the beginning of the survey, intended positively as a request for fairness and clarity, and ensuring the anonymity of the survey, guaranteeing that no individualized results

will be shared with managers or supervisors, and requesting absolute neutrality in the responses to enhance the approach and its impact, as shown in Figure 48. We believe this strategy can therefore be viewed as mostly beneficial for transparency and genuine engagement, and further reduces internal threats.

As explained in detail in the Guiding Principles section 7.3.1.1 (cf. *Explicit Triangulation*), we evaluated performing additional validation through qualitative follow-up interviews or other triangulation methods, which could have further strengthened confidence in mitigating remaining subtle biases. We decided not to pursue such further steps, primarily as the expected incremental improvements in the survey results would have been outweighed by the loss of anonymity and return to potential social desirability biases. Furthermore, the expected improvement would not justify the additional effort and expert time consumed. We believe our results are sound enough and trust our findings.

7.3.4.3 External Validity

Threat:

Given the specific industrial and highly specialized context (a global automotive Tier-1 supplier, company-specific MBSE-based reuse framework), an argument might be made that the generalizability of results to other contexts or industries may be limited.

Mitigation:

External validity in highly specialized, industrial settings typically focuses less on broad generalizability and more on contextual transferability of insights and methods [246]. The survey's detailed questionnaire, explicitly evaluating specific usability, understandability, and impact aspects of the reuse framework, ensures that results clearly specify contexts and conditions.

Thus, external validity is sufficiently addressed by clearly defining context, enabling practitioners in other safety-critical or MBSE-driven domains (such as aerospace or medical systems) to judge potential applicability to their settings.

Given that the approach would have to be specifically instantiated into any new domain and/or reuse context, we believe this threat is sufficiently addressed to allow a conclusion of generalizability of the approach to other contexts or industries with similarly positive impact.

7.3.4.4 Construct Validity

Threat:

A potential concern exists regarding whether the questionnaire correctly and comprehensively captures the constructs "usability," "understandability," and "impact" of the reuse framework.

Mitigation:

The questionnaire was carefully structured, explicitly asking respondents to evaluate each construct independently and providing multiple clearly defined questions per construct (e.g., explicit items related to ease of use, clarity, usefulness). Olson [226] shows that explicit structuring and detailed definitions of survey items significantly reduce ambiguity and improve the reliability of captured constructs. Furthermore, Konda & Mandava [18] highlight the validity of clearly defined constructs in expert surveys, especially for assessing the effectiveness of reuse frameworks. One exploratory question – never intended for quantitative analysis, and limited to participants with no prior reuse experience – was excluded due to interpretive ambiguity across language groups, thereby preserving the semantic consistency and cultural neutrality of the measured constructs. Thus, the construct validity is supported by clearly defined questionnaire items tailored to the attributes which were to be measured.

7.4 Summary

This chapter, representing the DS-II phase of this DRM-based thesis, marks the culmination of over a decade of academic research and sustained industrial application. It documents the generalization and subsequent large-scale, real-world instantiation of the reuse methodology developed in this work within a complex organizational and technological environment, carried out over an extended time frame at a globally leading automotive Tier-1 supplier.

Spanning more than eight years, the longitudinal case study highlights not only the technical advancements achieved, but also the critical accompanying enhancements in organizational systems engineering capabilities and institutional support. A systematically designed expert survey provides robust quantitative and qualitative confirmation of the methodology's impact, revealing a rare level of agreement among experienced practitioners.

Despite confidentiality constraints, the findings present compelling evidence of the method's industrial viability and sustained relevance. The chapter illustrates a rare depth of industrial validation, positioning the work as both methodologically rigorous and practically transformative.

8 Conclusion, Limitations and Outlook

8.1 Summary of the Research

This thesis addresses a persistent and pressing challenge in the development of complex systems: the lack of systematic and effective reuse practices for technical solutions, particularly in safety-critical domains such as the automotive industry. Although reuse is widely recognized as a means of improving quality, reducing effort, and accelerating development, practical reuse often remains ad-hoc, artifact-centric, and disconnected from the broader system context in which solutions must operate. Practitioners frequently focus on reusing isolated elements (e.g., code, diagrams) while overlooking the requirements, assumptions, rationale, and evidence needed for safe and meaningful reuse – especially in multidisciplinary and regulated environments.

To address this problem, the research identifies four core challenges:

- 1) Systematically capturing solutions with critical contextual information for reuse.
- 2) Providing an approach that ensures completeness, consistency, and effectiveness of captured solutions.
- 3) Integrating the reuse approach seamlessly into practitioners' existing tools and environments.
- 4) Providing guidance for evaluation and support for prioritized tooling enhancement and selection.

In response, the thesis proposes a novel approach: the **Model-based Design Pattern Library (MODPAL)**. The MODPAL approach operationalizes the idea of design patterns – long used as a way to capture reusable knowledge – by embedding structured reuse information directly into the models and artifacts already used by practitioners. This concept of *operationalizing design patterns* bridges the gap between classical pattern templates and the practical realities of engineering environments. The result is a structured, tool-agnostic yet context-sensitive methodology for reuse that is both rigorous and usable in real-world settings.

MODPAL consists of several interrelated components:

- A **Design Pattern Template** tailored for Safety Mechanisms, enabling the consistent and complete capture of reusable solutions.
- An **Artifact Reference Structure**, defining the attributes, relationships, and representation forms needed to support correct reuse in any model-based development environment.
- A set of **Guidelines** for instantiating the approach within an organization's tools and practices, including defined roles (e.g., *pattern developer* and *user*).
- A **reuse evaluation scheme** that enables systematic assessment of how well reuse-relevant attributes are captured and supported within a given tool environment.

The thesis makes four main claims regarding the advantages of this approach:

- 1) It enables **systematic capture** of reusable solutions, countering the current ad-hoc practices.
- 2) It allows organizations to **evaluate how well** their current tools support reuse-relevant attributes.
- 3) It provides a means to **guide and prioritize tooling enhancements** for improved reuse support.
- 4) It supports **selection of suitable toolchains or frameworks** by evaluating their adequacy for capturing domain-relevant reuse attributes.

These claims are validated through a series of case studies:

- **Chapter 5** applies the approach in a domain-independent research CASE tool, showing a measurable improvement in reuse capability through targeted tool enhancements.
- **Chapter 6** demonstrates instantiation in a domain-specific automotive safety modeling framework, where domain adequacy enabled high coverage of reuse-relevant attributes with minimal effort.
- **Chapter 7** generalizes the approach to a large-scale industrial setting, where it was used over multiple years to enable the systematic reuse of complex vehicle system solutions, culminating in an expert survey confirming its effectiveness and organizational impact.

The structure of the thesis closely follows Blessing and Chakrabarty's **Design Research Methodology (DRM)**:

- **Chapter 1** (Clarification) defines the problem and outlines the research goals.
- **Chapters 2 & 3** (Descriptive Study I) provide a contextual analysis of safety-critical system development practices and design pattern reuse.
- **Chapters 4–6** (Prescriptive Study) develop the MODPAL approach and demonstrate its feasibility in multiple tooling environments.
- **Chapter 7** (Descriptive Study II) generalizes the approach and provides industrial validation in a longitudinal case study.
- **Chapter 8** (Conclusion) reflects on the contributions, findings, limitations, and future directions.

Through this structure, the thesis delivers both a theoretical advancement – operationalizing design patterns for reuse – and a practical framework with validated impact across multiple levels of abstraction, organizational roles, and tool environments

8.2 Contributions of the Thesis

The contributions of this thesis span three domains: conceptual, methodological, and empirical. Together, they define, implement, and validate a novel approach to enabling systematic solution reuse in engineering practice — grounded in industry needs and evaluated across research and industrial contexts.

8.2.1 Conceptual Contributions

The thesis introduces the concept of **operationalizing design patterns** as a practical mechanism for system solution reuse. While design patterns have long served as a vehicle for capturing expert knowledge in software engineering, their application in the structured reuse of technical system solutions has remained largely theoretical or superficial. This work reconceptualizes pattern-based reuse by embedding it directly into engineering artifacts – enabling practitioners to capture, share, and apply reusable system knowledge within their existing development environments.

This constitutes a conceptual shift from traditional textual pattern cataloging toward a **model-anchored, tool-integrated reuse strategy**, where reuse is not limited to single artifacts, but includes surrounding rationale, constraints, assumptions, and structured argumentation. The notion of capturing this reuse knowledge directly “where the work happens” – in a way that aligns with how Engineers model, implement, and justify system behavior – is a defining conceptual contribution of the thesis.

8.2.2 Technical and Methodological Contributions

At the methodological level, the thesis defines and implements the **MODPAL (Model-based Design Pattern Library) approach**, a structured, tool-agnostic reuse framework built on systems engineering foundations.

Key technical contributions include:

- **A Design Pattern Template for Safety Mechanisms**

Developed through an extensive literature review and refinement process, this template extends existing design pattern schemas with attributes tailored for the reuse of safety-relevant technical solutions. It provides a completeness benchmark and reuse “definition of done.”

- **An Artifact Reference Structure**

This formal structure captures the relationships between pattern attributes, engineering artifacts, and modeling constructs. It defines how reuse knowledge can be embedded directly into architectural models and safety case elements.

- **Reuse Workflow and Instantiation Guidelines**

The thesis provides concrete guidance for instantiating the MODPAL approach into any development environment – including structured roles for Pattern Developers and Users, modeling conventions, and validation checks to ensure argumentation soundness.

- **Tool Support and Practical Integration**

A major methodological innovation lies in the embedding of the reuse framework into actual engineering tools. Unlike prior approaches requiring specialized platforms or format translations, MODPAL integrates reuse directly into practitioners' modeling environments, avoiding the need for migration or retraining.

- **Reuse Environment Evaluation Framework**

The approach also includes a structured method for evaluating the reuse readiness and extensibility of existing modeling toolchains. This provides practitioners with actionable guidance for improving their development environments in a targeted way.

8.2.3 Empirical Contributions

The thesis is distinguished by its depth of practical validation. Contributions include:

- **Technical Validation in Academic Case Studies**

In Chapter 5, MODPAL was instantiated into a generic research CASE tool (AutoFOCUS3), demonstrating feasibility and producing measurable improvements in reuse maturity. With seven Safety Mechanisms implemented and over 40 others evaluated, the completeness and applicability of the approach in this scoped context was confirmed.

- **Validation in a Domain-Specific Framework**

In Chapter 6, the approach was applied to the SAFE automotive modeling platform, where its domain adequacy enabled high attribute coverage (89%) with minimal extension effort. This validated the core hypothesis that domain-specific modeling frameworks improve reuse quality.

- **Longitudinal Industrial Evaluation**

In Chapter 7, the approach was generalized and applied in a multi-year industrial case study at a globally leading Tier-1 supplier. The methodology was used to structure the reuse of architectural solutions across programs and was integrated into real workflows. A survey of 15 experts – weighted and calibrated for role, expertise, and organizational context – confirmed the approach's usability, effectiveness, and impact.

- **Evidence of Strategic and Organizational Impact**

Correlation analysis from the expert survey revealed that successful reuse adoption correlates strongly with MBSE maturity and organizational support, but less so with experience or tenure. These findings offer actionable insights into the systemic enablers of reuse at scale.

Taken together, these contributions represent a significant advancement in how system solutions – thoroughly developed and evaluated through the scoped example of safety mechanisms, before generalization in an industrial setting – can be reused systematically, effectively, and at scale. The work not only defines a rigorous and practical methodology, but also demonstrates its value across both research and real-world settings.

8.3 Key Findings and Impact

This section summarizes the key observations, insights, and broader implications drawn from the research and its industrial application. It brings together practical findings from the longitudinal case study, highlights systemic enablers and barriers for reuse, and outlines the organizational and methodological impact observed across multiple years of deployment.

8.3.1 Observed Improvements in Engineering Practice

Over the course of the multi-year industrial case study, a number of recurring improvements in systems and software engineering practice were observed – many of which can be attributed to the structure, discipline, and insight prompted by the reuse methodology:

- **Reduction of friction losses:** By introducing a structured reuse process, the approach facilitated the consistent restructuring of design and development artifacts, which in turn reduced inconsistencies and misalignments, ultimately lowering the technical debt across projects.
- **Reduction of accidental complexity:** Inadequate tooling and fragmented workflows were gradually improved, with the reuse approach serving as a catalyst for targeted tool enhancements.
- **Reducing the encoding gap:** The method helped to bridge the gap between domain concepts and generic modeling languages or tools, reducing loss of intent and enabling more automation.
- **Improved integration readiness:** Practitioners were encouraged to take a more holistic view of the reuse process, which led to more deliberate workflows, improved integration into configuration management and CI/CD pipelines, and fewer downstream issues.
- **Time-to-market gains:** The existence of reusable architectural solutions enabled teams to respond more effectively to new product requirements and customer RFQs – both through systematic reuse and, when necessary, through controlled “clone-and-own” adaptation.

These gains – albeit in some cases qualitative – represent a strong indicator of the systemic benefits that structured reuse can bring, especially when aligned with sound model-based systems engineering practices and upskilling.

8.3.2 Tooling and Modeling Framework Adequacy

The reuse methodology also served as a diagnostic and evaluative lens for assessing the capabilities of the modeling environments in use. The case studies revealed that:

- **Modeling frameworks significantly influence reuse maturity:** Domain-specific frameworks (e.g., SAFE-MF) enabled higher degrees of reuse readiness than domain-agnostic

ones (e.g., AF3), confirming the hypothesis that richer and more context-aware meta-models support better information capture.

- **Tooling impacts reuse effectiveness:** Development tools and meta-models typically reflect their original intent (e.g., fast code generation, simulation), often at the expense of reusability. As such, reuse requires explicit support through tooling enhancements or selection of frameworks with reuse in mind.
- **Best-in-breed federation:** The findings advocate for a shift from monolithic, one-size-fits-all modeling frameworks to a loosely coupled federation of tools and models, emphasizing openness, semantic interfaces, and modular interoperability – aligning with modern MBSE initiatives like SysML v2 and OPENMBEE⁴⁶.

8.3.3 Organizational and Cultural Factors

The longitudinal case study (Chapter 7) offered deep insight into systemic barriers to reuse in large organizations. The survey results revealed the following key factors:

- **MBSE maturity is a critical enabler:** A particularly significant finding from the expert survey was the strong correlation between Model-Based Systems Engineering (MBSE) expertise and positive assessments across nearly all evaluation metrics – most notably, perceived effectiveness, usability, and impact of the reuse methodology ($r = 0.67$ to 0.83). This suggests that MBSE capability – comprehensively covering skills and expertise, tooling and frameworks, and processes and governance – is not merely supportive, but a key predictor of reuse success in complex engineering environments. Notably, these correlations were stronger than those tied to years of professional experience, underscoring that domain-specific modeling literacy, rather than tenure alone, is the primary enabler of successful reuse adoption. This highlights the strategic value of targeted MBSE upskilling initiatives over traditional seniority-based capability assumptions.
- **Organizational support is essential:** The Organizational Support Index (OSI) also showed strong correlation with positive reuse assessments. This confirms that reuse cannot be driven purely by technical approaches; process alignment, managerial backing, and cultural acceptance are equally important.
- **Lack of knowledge access remains the biggest barrier:** Across all experience levels and roles, survey participants highlighted poor access to reusable, structured knowledge as the single most important obstacle to effective reuse.

This underscores the need to combine rigorous technical reuse methodologies with structured knowledge management practices, and to prioritize transparency and architectural continuity across projects and organizational boundaries as a strategic management priority.

⁴⁶ Originated as a NASA/JPL-led initiative, OPENMBEE brings together expert industry practitioners and toolmakers to reimagine MBSE around the core principles of model-centrism, openness, modularity, interoperability, and a best-in-breed ecosystem. www.openmbee.org

8.3.4 Reuse as a Catalyst for Broader Change

Perhaps the most significant finding from the industrial experience was the way in which the reuse methodology acted as a catalyst for broader improvements:

- **Better tool selection and extension:** Reuse challenges directly led to concrete tooling improvements, encouraging teams to systematically assess their modeling frameworks against reuse needs.
- **Evidence-driven practice:** Structured definition and evaluation mechanisms introduced through the methodology – such as reuse completeness checks, expert user surveys, and guided pattern capturing – created new avenues for performance tracking and continuous improvement.
- **Upskilling and capability building:** The development and industrial application of the reuse methodology coincided with, and actively contributed to, broader organizational efforts aimed at enhancing systems engineering maturity. While these initiatives involved a wide set of stakeholders and competencies, the methodology and its growing visibility played a catalytic role; helping to reinforce the need for consistent systems engineering terminology, motivating structured upskilling programs aligned with INCOSE standards, and supporting the harmonization of modeling frameworks across organizational units. This alignment between methodological innovation and organizational capability building appears to have amplified the practical effectiveness of the reuse initiative.
- **Domain discovery over tool limitation:** Practitioners became more confident in addressing domain challenges without being constrained by the limits of their tools. This mindset shift enabled better system understanding and more robust design decisions.

These impacts affirm that reuse methodologies can play a central role not just in artifact reuse, but in driving systemic improvements in engineering culture, capability, and infrastructure.

8.4 Threats to Validity and Limitations

This section discusses the key limitations of the research and summarizes relevant threats to validity. Detailed treatments of evaluation validity are provided in sections 4.9.2 (*Prescriptive Study* evaluations) and 7.3.4 (expert survey in the *Descriptive Study-II* phase). Here, we offer a condensed overview of the main concerns and boundaries of the work.

8.4.1 Threats to Validity

For the technical evaluation in the Prescriptive Study phase (Chapters 5 and 6), primary threats included the subjectivity of attribute-based evaluation scales and limited formal statistical grounding. These were mitigated by consistent evaluation criteria, collaborative scoring, and validation by independent users. The goal was to assess feasibility and localized impact, not to produce generalizable statistical findings.

In the Descriptive Study II phase (Chapter 7), the most significant threat was the relatively small final sample size (15 experts). However, the response rate was high, covering 60% of all exposed users, with respondents representing a highly experienced and diverse expert population. Bias was further mitigated through anonymous survey design, inclusion of critical voices, and calibrated expertise-weighted analysis. Construct and external validity were addressed through detailed, context-specific questionnaires and transparency about the scope of the findings.

8.4.2 Limitations of Safety Mechanism Reuse Scope

While the approach was validated for Safety Mechanism reuse, it does not currently support fully compositional safety case automation or behavioral formalization. Safety argument integration remains partially manual, relying on expert review. The envisioned modular reuse of Safety Elements out of Context (as defined in ISO 26262) or COTS integration with pre-certified safety cases, though conceptually supported, is not fully implemented within the current framework.

Furthermore, error propagation modeling, compositional timing analysis, and the automated verification of assumptions between pattern instances and system models are considered out of scope for this thesis. These areas offer opportunities for future research and formalization.

8.4.3 Limitations for General Applicability and Industrial Transferability

Though the approach has demonstrated successful industrial deployment, broader applicability depends on organizational and technical factors. Key enablers include sufficient MBSE capability and maturity; aligned modeling practices; and accessible, structured system solution and architecture knowledge. Organizations lacking these prerequisites may face significant barriers to adoption.

Moreover, instantiation requires non-trivial investment in abstraction, formalization, and model restructuring. The approach also presupposes openness or extensibility of the existing toolchain. In tightly siloed or legacy-dominated environments, this may pose a challenge. Care must be taken to preserve consistency and avoid fragmentation when adapting the approach.

Finally, while the methodology generalizes beyond safety mechanisms, tailoring to other solution classes or domains demands committing to additional – potentially substantial – work. These up-front investment efforts must be considered, planned and integrated into long term objectives.

In summary, while the thesis demonstrates strong internal validity and contextual generalizability, certain methodological, technical, and organizational boundaries remain. These should inform both interpretation of results and future adoption efforts.

8.5 Outlook and Open Questions

A recent (2023) study by Pandey et al. [14] highlighted a key disconnect between academic conceptualizations of design patterns and their practical application in the automotive industry, where reuse typically occurs in conjunction with broader architectural structures and domain-specific constraints. The methodology and empirical work presented in this thesis directly address this gap by enabling the structured integration of system-level artifacts – including architectural

models, development elements, and safety cases – into a cohesive, tool-supported framework. In doing so, the approach operationalizes design patterns in a form that aligns with actual industrial needs and practices, bridging the divide between theoretical rigor and engineering utility.

While the feasibility, effectiveness, and potential of systematic reuse in model-based systems engineering have been demonstrated, long-term success depends not only on technical soundness, but on sustained organizational commitment, structural enablers, and the strategic alignment of stakeholders across domains and development phases.

A central insight from the longitudinal industrial case study is the persistent challenge of **fragmented and inaccessible historical engineering knowledge**. Even when technical capabilities exist, reuse at scale remains hindered by organizational silos, inconsistent modeling practices, and poorly governed knowledge management. Addressing these foundational issues is not merely a question of methodology, but of governance, resourcing, and cultural change. The emergence of **federated modeling architectures**, as supported by technologies and initiatives such as **SysML v2**, **OPENMBEE**, **OML**⁴⁷, or other semantic interoperability frameworks, offers promising avenues to better align reuse efforts across tools and teams. However, these require active coordination, shared ontologies, and long-term planning.

In this context, **AI-assisted reuse** represents a significant opportunity. Language models and generative techniques can already support the identification, extraction, normalization, and contextual adaptation of reusable assets from legacy datasets. As modeling standards become more formalized and interconnected, AI-driven approaches can help bridge gaps in traceability, propose candidate reuse elements, and even suggest structural improvements to artifacts. However, such capabilities will only be effective if the surrounding engineering environment is semantically rich, well-structured, and consistently maintained – goals that depend directly on the successful adoption of the kinds of targeted modeling rigor advocated in this thesis.

To that end, **reuse maturity models** such as the one proposed by Wu et al. [185] offer a useful framework for evaluating organizational readiness. The author’s experience suggests that combining such models with tangible, user-centered methodologies – like the one developed here – provides a practical path toward long-term reuse institutionalization. Still, one critical limitation of maturity models is that they typically overlook **the behavioral economics of reuse adoption**, especially at executive levels.

A critical consideration for any organization aiming to adopt systematic reuse is the recognition that meaningful institutionalization requires not only technical tailoring, but also substantial up-front investment. Tailoring the approach to new domains or solution classes entails not only methodological effort, but also resource allocation, stakeholder alignment, and the cultivation of new capabilities. **These investments can create a counterintuitive dynamic**: in the short term, diverting key resources to develop reusable frameworks often leads to an observable drop in delivery productivity, as these teams step away from direct project contributions. Rollout efforts can introduce further slowdowns, as teams adapt to new structures and practices. Crucially, if organizations abandon the reuse initiative during this transitional dip, they usually end up

⁴⁷ OML (Open Modeling Language) is an OWL2-aligned semantic modeling language designed to define, share, and integrate complex domain knowledge across tools and systems. It supports precise, machine-readable ontologies for model-based engineering and digital engineering applications.

reinforcing “short-termism” – because halting the initiative yields an immediate productivity rebound, falsely validating the decision to give up. This negative reinforcement loop has been observed across industries, and presents one of the greatest cultural obstacles to sustained reuse adoption. Overcoming it requires not only strategic commitment but also clear communication of long-term goals and success metrics, so that organizations are equipped to navigate the temporary setbacks that often precede lasting improvement.

Executive decision-makers must therefore be equipped not only with ROI projections and maturity scores, but also with a realistic understanding of these behavioral tipping points. The challenge lies not only in designing better reuse mechanisms, but in preparing organizations to endure the temporary discomforts that accompany lasting transformation.

Finally, although this thesis demonstrated the applicability of the approach across multiple toolchains and problem domains, several **open research questions** remain – particularly regarding compositional safety argumentation, or the formal integration of fault modeling or timing behavior into pattern definitions, but also the broader issues of reuse in large organizations discussed above. These topics represent promising directions for future research.

8.6 Cauda

This thesis is the culmination of a non-linear journey spanning over a decade of academic research and sustained industrial application. Combining theoretical and methodological contributions with a practical framework and a rare depth of industrial validation positions the work as both methodologically rigorous and practically transformative. The resulting insights not only validate the proposed methodology but also offer actionable guidance for organizations striving to scale reuse effectively in complex, real-world engineering environments.

Bibliography and List of references

- [1] European Space Agency (ESA). "Ariane 5 Flight 501 Failure, Report by the Inquiry Board". <http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf>. 1996 (Retrieved March 5th, 2015).
- [2] K.K. Breitman, J.C.S.P. Leite, and A. Finkelstein. „The World’s Stage: A Survey on Requirements Engineering Using a Real-Life Case Study”. Journal of the Brazilian Computer Society, vol. 6, no. 1, pp. 13-38, July 1999.
- [3] L.M. Cysneiros and J.C.S.P. Leite. “Integrating Non-Functional Requirements into Data Models”. Proc. Fourth Int’l Symposium Requirements Engineering. June 1999.
- [4] D.R. Lindstrom. “Five Ways to Destroy a Development Project”. IEEE Software, pp. 55-58, Sept. 1993.
- [5] Martin, H.; Baumgart, S.; Leitner, A. and Watzenig, D. (2014). “Challenges for Reuse in a Safety-Critical Context: A State-of-Practice Study.” SAE Technical Papers. 1. 10.4271/2014-01-0218.
- [6] Nancy G. Leveson. “SAFWARE – System Safety and Computers.” Addison-Wesley Publishing, Massachusetts, USA. 1995.
- [7] M. Collins and E. Aldrin. E.M. Cortright (ed.) "A Yellow Caution Light". NASA SP-350, Apollo Expeditions to the Moon. Chapter 11.4. NASA. Washington, DC. 1975.
- [8] S. Baase. “A gift of fire: Social, legal, and ethical issues for computing and the Internet”. Prentice Hall. 2008.
- [9] T. Wray. “The everyday risks of playing safe”. New Scientist, 119, pp. 61-65. 1988.
- [10] R. Walmsley et al. “NATS System Failure 12 December 2014 – Final Report: Independent Enquiry”. UK Civil Aviation Authority (CAA) and National Air Traffic Services (NATS). Official final report. May 13th, 2015.
- [11] Johnston, P., & Harris, D. (2019). "The Boeing 737 MAX Saga: Lessons for Software Safety." *Reliability Engineering & System Safety*, 196, Article 106757. <https://doi.org/10.1016/j.ress.2019.106757>
- [12] United States Federal Aviation Authority. *FAA Final Summary Report on Boeing 737 MAX*. 2020. https://www.faa.gov/sites/faa.gov/files/2022-08/737_RTS_Summary.pdf
- [13] Banerjee, P. *CrowdStrike Cyber Incident vs. Past Major Cyber Incidents: Analysis and Solutions*. International Journal for Multidisciplinary Research (IJFMR), 6(4), July 2024. <https://www.ijfmr.com/papers/2024/4/25310.pdf>
- [14] Pandey, A., et al. (2023). "Design Patterns Understanding and Use in the Automotive Industry: An Interview Study." LNCS. *ResearchGate*. https://www.researchgate.net/publication/376169420_Design_Patterns_Understanding_and_Use_in_the_Automotive_Industry_An_Interview_Study

- [15] Sen, A. "The Role of Opportunism in the Software Design Reuse Process," *IEEE Trans. Softw. Eng.*, vol. 23, no. 7, pp. 418–436, 1997.
- [16] T. P. Kelly and J. A. McDermid. "Safety case construction and reuse using patterns." 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP 1997). (pp. 55-69). Springer London. 1997.
- [17] Gleirscher, M. and Kugele, S. (2019). "Assurance of System Safety: A Survey of Design and Argument Patterns." *ArXiv* abs/1902.05537.
- [18] Konda, B. M., & Mandava, K. K. (2010). A Systematic Mapping Study on Software Reuse. Master's Thesis, Blekinge Institute of Technology. <https://www.diva-portal.org/smash/get/diva2:831759/FULLTEXT01.pdf>
- [19] Kahneman, D., Krueger, A. B., Schkade, D. A., Schwarz, N., & Stone, A. A. (2006). "Would You Be Happier If You Were Richer? A Focusing Illusion." *Science*, 312(5782), 1908–1910. <https://doi.org/10.1126/science.1129688>
- [20] International Standards Organization – ISO 26262:2011. "Road Vehicles – Functional Safety." www.iso.org, 2011.
- [21] M. Khalil, B. Schätz and S. Voss. "A Pattern-based Approach towards Modular Safety Analysis and Argumentation." *Embedded Real Time Software and Systems Conference (ERTS 2014)*, Toulouse, France. 2014.
- [22] Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. University of Chicago Press.
- [23] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, Boston, MA, USA, 1997.
- [24] Blessing, L. T. M., & Chakrabarti, A. (2009). DRM: A Design Research Methodology. In *DRM, a Design Research Methodology* (pp. 13–42). Springer London.
- [25] Wieringa, R. J. (2009). "Design science as nested problem solving." In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology* (pp. 1–12). ACM. <https://doi.org/10.1145/1555619.1555624>
- [26] Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). "A design science research methodology for information systems research." *Journal of Management Information Systems*, 24(3), 45–77. <https://doi.org/10.2753/MIS0742-1222240302>
- [27] Österle, H., & Otto, B. (2010). "Consortium Research." *Business & Information Systems Engineering*, 2(5), 283–293. <https://doi.org/10.1007/s12599-010-0114-5>
- [28] Vaishnavi, V. K., and Kuechler, W. (2004). "Design Research in Information Systems." Accessed April 13, 2025. <https://desrist.org/design-research-in-information-systems/>.
- [29] ISO/IEC/IEEE. (2023). "Systems and software engineering — System life cycle processes" (ISO/IEC/IEEE 15288:2023). International Organization for Standardization.

- [30] INCOSE. (2023). INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities (5th ed.). Wiley.
- [31] B. J. Bonnett. "Position paper on software safety and security critical systems". In proceedings of IEEE Compcon '84. Vol. 84, p. 191. 1984.
- [32] Autonetics Division of North American Rockwell. Inc.; "Part I - Preliminary Maintenance Manual of the Minuteman D-17A Computer and Associated Test Equipment." P.O. Memo 71. Anaheim, California, USA. January 1960.
- [33] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. "Basic concepts and taxonomy of dependable and secure computing." IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004.
- [34] N. Storey. "Safety-Critical Computer System". Addison-Wesley, Harlow, England, 1996.
- [35] A. van Lamsweerde. "Requirements Engineering - From System Goals to UML Models to Software Specifications". John Wiley & Sons, Chichester, UK. 2009.
- [36] International Electrotechnical Commission – IEC61508. "Functional safety for electrical / electronic / programmable electronic safety-related systems", 1st Edition. International Electrotechnical Commission (IEC). Geneva, Switzerland. 1998.
- [37] J. Börcsök. „Funktionale Sicherheit: Grundzüge sicherheitstechnischer Systeme“. VDE-Verlag. 2011.
- [38] T. A. Henzinger, EPFL & J. Sifakis, Verimag. "The Discipline of Embedded Systems Design, IEEE Computer Society 2007 – an update of The Embedded Systems Design Challenge". Proc. 14th Int'l Symp. Formal Methods, LNCS 4085, pp. 1-15., Springer, 2006.
- [39] U.S. Navy Archives. Adm. Grace Hopper's Biography.
<http://www.public.navy.mil/surfor/ddg70/Pages/graceHopper.aspx>.
(Retrieved March 5th, 2015)
- [40] M. Suerken and T. Peikenkamp. "Model-based Application of ISO 26262: The Hazard and Risk Assessment". SAE Technical Paper 2013-01-0184, April 2013.
- [41] L. L. Pullum. "Software fault tolerance techniques and implementation". Artech House, Inc., Norwood, MA, USA, 2001.
- [42] B. Randell. "Dependability, structure and infrastructure." Technical Report CS-TR-877, School of Computing Science, University of Newcastle upon Tyne, November 2004.
- [43] F. Bachmann, L. Bass, and M. Klein. "Illuminating the Fundamental Contributors to Software Architecture Quality", Technical Report, CMU/SEI-2002-TR-025, Software Engineering Institute (SEI), Carnegie Mellon University. August 2002.
- [44] W. Wu and T. Kelly. "Safety Tactics for Software Architecture Design". In Proceedings of the 28th Annual International Computer Software and

- Applications Conference - Volume 01 (COMPSAC '04), Vol. 1, pp. 368-375. IEEE Computer Society, Washington, DC, USA.2004.
- [45] A. Long. "Beauty and the Beast – Use and Abuse of Fault Tree as a Tool" (pdf). Hernandez Engineering Inc. MSFC, AL35812 USA. fault-tree.net, retrieved 16 January 2010. Alternative Link- Univ. of West England. (Retrieved March 7th, 2015).
- [46] MISRA-C:2004. "Guidelines for the use of the C language in critical systems", ISBN 978-0-9524156-2-6, MIRA. October 2004.
- [47] MISRA AC AGC. "Guidelines for the application of MISRA-C:2004 in the context of automatic code generation", ISBN 978-1-906400-02-6, MIRA. November 2007.
- [48] T. Rückstieß, C. Osendorfer, P. van der Smagt. "Minimizing Data Consumption with Sequential Online Feature Selection". International Journal of Machine Learning and Cybernetics. 2012.
- [49] R. Lemos, C. Gacek and A. Romanovsky. "Architecting Dependable Systems". LNCS 2677, pp. 37-60, pp. 129-149, Springer, London, 2003.
- [50] European Committee for Electro-Technical Standardization – CENELEC EN 50126 "Railway applications - The specification and demonstration of reliability, availability, maintainability and safety (RAMS)". CENELEC. 1999. New version (2013) in Revision.
- [51] T. Kelly and R. Weaver. "The Goal Structuring Notation. A Safety Argument Notation". *Proc. of the Dependable Systems and Networks (DSN) 2004 Workshop on Assurance Cases*. 2004.
- [52] The Hon. Lord Cullen. "The Public Inquiry into the Piper Alpha Disaster, Vols. 1 and 2". *Report to Parliament by the Secretary of State for Energy by Command of Her Majesty*. November 1990.
- [53] D. Whyte. "Moving the goalposts: The deregulation of safety in the post piper alpha offshore oil industry". Contemporary Political Studies – Belfast; 2; 1148-1160. Political Studies Association of the United Kingdom (PSA) – 47th Annual conference. 1997.
- [54] K. Heiler, "Is the Australian Mining Industry Ready for a Safety Case Regime". 31st International Conference of Safety in Mines Research Institute, Brisbane, Australia. Oct. 2005.
- [55] U.K. Health and Safety Executive – HSE. "A guide to the Offshore Installations (Safety Case) Regulations 1992". Health and Safety Executive, HSE Books. 1992.
- [56] Leveson, N. "The Use of Safety Cases in Certification and Regulation". Journal of System Safety, 47.6 (2011): 1-5. The International System Safety Society. 2011.
- [57] M. Khalil and E. Metzker. ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D3.1.3 / D3.4 "Proposal for extension of Meta-model for safety-case modeling and documentation". The SAFE / Safe-E Consortium – www.safe-project.eu. 2013.

- [58] T. Kelly, I. Habli, et al. "Goal Structuring Notation (GSN). GSN COMMUNITY STANDARD VERSION 1". Origin Consulting (York) Limited, on behalf of the Contributors. November 2011.
- [59] M. Broy and K. Stolen. "Specification and development of interactive systems. FOCUS on Streams, Interfaces and Refinement". Monographs in Computer Science. Springer Science & Business Media, 2001.
- [60] B. Fischhoff, P. Slovic, and S. Lichtenstein. "Fault Trees: Sensitivity of Estimated Failure Probabilities to problem Representation". Journal of Experimental Psychology: Human Perception and Performance, vol. 4. 1978.
- [61] K.J. Vicente and J. Rasmussen. "Ecological Interface Design: Theoretical Foundations". IEEE Transactions. Systems, Man, and Cybernetics, vol. 22, no. 4. July/Aug. 1992.
- [62] Z. Kunda. "Social Cognition: Making Sense of People". MIT Press. 1999.
- [63] R.S. Nickerson. "Confirmation Bias; A Ubiquitous Phenomenon in Many Guises". Review of General Psychology (Educational Publishing Foundation). 2 (2); pp. 175–220. 1998.
- [64] S. Dekker. "The Field Guide to Understanding 'Human Error'". Ashgate Publishers. 3rd Ed. 2014.
- [65] G. Despotou and T. Kelly. "First Contact with Safety Cases: a Quick Guide to Common Questions and Misconceptions". The Safety Critical Systems Club Newsletter, Volume 21, Number 3. May 2012.
- [66] C. Haddon-Cave. "The Nimrod Review – HC 1025". London: The Stationery Office Limited. Oct. 28, 2009.
- [67] A. Stevenson, (Ed.). "Oxford dictionary of English". Oxford University Press. 2010.
- [68] H. Stachowiak. "Allgemeine Modelltheorie". Springer. 1973.
- [69] B. Schätz. "Model-based development of software systems: from models to tools." Habilitation Diss. 2009.
- [70] B. Schätz, M. Broy, F. Huber, J. Philipps, W. Prenninger, A. Pretschner and B. Rumpe. "Model-Based Software and Systems Development – White Paper". Technical University of Munich. 2004.
- [71] J. Bézivin. "On the Unification Power of Models". Journal on Software and Systems Modeling (SoSyM) 4. No. 2, pp. 171–188. 2005.
- [72] Object Management Group (OMG). "Model Driven Architecture (MDA) – MDA Guide rev. 2.0". OMG Document ormsc/2014-06-01. 2014.
- [73] K. Pohl, H. Hönniger, R. Achatz and M. Broy. "Model-Based Engineering of Embedded Systems – The SPES 2020 Methodology". Springer, Berlin. 2012.
- [74] M. Broy and R. Steinbrüggen. "Modellbildung in der Informatik". Springer. 2004.
- [75] Object Management Group (OMG). "OMG Unified Modeling Language (OMG UML), Interfaces". Version 2.4.1. OMG Document 11-08-05. August 2011.

- [76] Institute for Electric and Electronic Engineers (IEEE). "IEEE Standard 1471 – Recommended Practice for Architectural Description of Software Intensive Systems". IEEE Standard 1471. New York, USA. 2000.
- [77] International Organization for Standardization (ISO). "ISO/IEC/IEEE 42010 – Systems and Software Engineering: Architecture description". ISO/IEC/IEEE Standard 42010. Geneva, Switzerland. 2011.
- [78] International Organization for Standardization. ISO/IEC/IEEE 12207:2017 – Systems and Software Engineering – Software Life Cycle Processes. ISO, IEC, and IEEE, 2017.
- [79] W. Böhm, S. Henkler, F. Houdek, A. Vogelsang, and T. Weyer. "Bridging the Gap between Systems and Software Engineering by Using the SPES Modeling Framework as a General Systems Engineering Philosophy". Conference on Systems Engineering Research (CSER 2014). Published in *Procedia Computer Science* 28, pp. 187-194. 2014.
- [80] International Standards Organization – ISO/IEC/IEEE 42010:2022. "Software, systems and enterprise — Architecture description". www.iso.org, 2022.
- [81] Object Management Group (OMG). "Model Driven Architecture (MDA) – Object Management Group – White Paper – Draft 3.2". OMG Document omg/011-00-05. November 27, 2000.
- [82] Object Management Group (OMG). "Model Driven Architecture (MDA)". OMG Document ormsc/2001-07-01. July, 2001.
- [83] M. Brambilla, J. Cabot and M. Wimmer. "Model-driven software engineering in practice." *Synthesis Lectures on Software Engineering* 1, no. 1, pp. 1-182. Morgan ClayPool. 2012.
- [84] D. Parnas. "Designing Software for Ease of Extension and Contraction". *IEEE Transactions on Software Engineering*. March 1979.
- [85] INCOSE. (2007). "Systems Engineering Vision 2020". International Council on Systems Engineering.
- [86] INCOSE. (2020). "INCOSE Vision 2035: The Future of Systems Engineering". International Council on Systems Engineering.
- [87] NASA. (2007). "NASA Systems Engineering Handbook" (Rev. 2). NASA/SP-2007-6105 Rev2.
- [88] OMG. (2019). "OMG Systems Modeling Language (SysML) Specification, Version 1.6". Object Management Group.
- [89] Friedenthal, S., Moore, A., & Steiner, R. (2014). „A Practical Guide to SysML: The Systems Modeling Language" (2nd ed.). Morgan Kaufmann.
- [90] A. Joshi, M.P.E. Heimdahl, S. P. Miller and M. W. Whalen. "Model-based safety analysis – Final Report". NASA Langley Formal Methods. 2006. – Presented earlier in shorter form as: A. Joshi, S. P. Miller, M. W. Whalen and M.P.E. Heimdahl. "A proposal for model-based safety analysis." In *Proc. of the 24th Digital Avionics Systems Conference (DASC 2005)*. Vol. 2, pp. 13-pp. IEEE. 2005.

- [91] P. Struss. "A Conceptualization and General Architecture of Intelligent Decision Support Systems". MODSIM2011, 19th International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand, pp. 2282-2288. December 2011.
- [92] C. Price and N. Snooke. "An Automated Software FMEA". Proceedings of the International System Safety Regional Conference, Singapore. April 2008.
- [93] P. Struss and A. Fraracci. "Automated Model-based FMEA of a Braking System". In: 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (Safeprocess 2012). Mexico City. 2012.
- [94] M. Broy, S. Kirstan, H. Krcmar, B. Schätz and J. Zimmermann. "What is the benefit of a model-based design of embedded software systems in the car industry?" Software Design and Development: Concepts, Methodologies, Tools, and Applications: 310. 2013
- [95] L. Grunske and Jun Han. "A comparative study into architecture-based safety evaluation methodologies using AADL's Error Annex and failure propagation models." In 11th IEEE High Assurance Systems Engineering Symposium, HASE 2008. pp. 283-292. IEEE, 2008.
- [96] Kharatyan, A., Tekaat, J., Japs, S., Anacker, H., & Dumitrescu, R. (2021). "Metamodel for Safety and Security Integrated System Architecture Modeling." *Proceedings of the Design Society: International Conference on Engineering Design (ICED21)*, 1, 2027–2036.
<https://doi.org/10.1017/pds.2021.464>
- [97] C. Alexander. "The timeless way of building. Vol. 1". Oxford University Press. 1979.
- [98] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. "Systematic mapping studies in software engineering." In Proc. of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE). Bari, Italy. 2008.
- [99] S. Teufl, M. Khalil, and D. Mou. "Requirements for a model-based requirements engineering tool for embedded systems: Systematic literature review and survey." White paper, fortiss. Munich, Germany. 2013.
http://www.download.fortiss.org/public/projects/af3/research/2013/MbRE_tool_requirements_for_embedded_systems.pdf
- [100] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. "Pattern-oriented software architecture: a system of patterns". John Wiley & Sons, Inc. New York. 1996.
- [101] D. Riehle and H. Züllighoven. "A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor". In Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. pp. 9-42. Reading, MA. Addison-Wesley. 1995.
- [102] K. Beck and W. Cunningham. "Using Pattern Languages for Object-Oriented Programs". Presented at the Conference on Object-Oriented

- Programming Systems, Languages, and Applications (OOPSLA'87), Workshop on the Specification and Design for Object-Oriented Programming, Orlando, Florida. 1987.
- [103] J. Coplien. "Advanced C++ programming styles and idioms". Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 1992.
- [104] J. Coplien. "Idioms and Patterns As Architectural Literature". IEEE Software, vol. 14, pp. 36-42. 1997.
- [105] P. Coad. "Object-Oriented Patterns". Communications of the ACM, vol. 35, Nr. 9, pp. 152-159. 1992.
- [106] E. Gamma. "Object-Oriented Software Development based on ET++: Design Patterns, Class Library, Tools". (In German). Ph.D. Thesis. University of Zurich. Institut für Informatik. Zurich. 1991.
- [107] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Abstraction and Reuse of Object-Oriented Design". Presented at ECOOP'93 - Object-Oriented Programming, 7th European Conference, Kaiserslautern, Germany. 1993.
- [108] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software". Pearson Education. 1994.
- [109] D. Manolescu, M. Voelter, and J. Noble. "Pattern Languages of Program Design 5". Addison-Wesley Professional. 2005.
- [110] B. Appleton. "Patterns and Software: Essential Concepts and Terminology". 1997.
- [111] R. Damaševicius, G. Majauskas, and V. Štuikys. "Application of design patterns for hardware design". In DAC '03: Proceedings of the 40th annual Design Automation Conference, pages 48–53, New York, NY, USA, 2003. ACM.
- [112] N. Yoshida. "Design patterns applied to object-oriented SOC design". In 10th Workshop on Synthesis and System Integration of Mixed Technologies (SASIMI 2001), Nara, Japan, Oct. 2001.
- [113] R. Damaševicius and V. Štuikys. "Application of UML for hardware design based on design process model". In ASP-DAC '04: Proceedings of the 2004 Asia and South Pacific Design Automation Conference, pp. 244–249, Piscataway, NJ, USA, 2004. IEEE Press.
- [114] F. Rincon, F. Moya, J. Barba, and J. C. Lopez. "Model reuse through hardware design patterns". In DATE '05: Proceedings of the conference on Design, Automation and Test in Europe, pages 324–329, Washington, DC, USA, 2005. IEEE Computer Society.
- [115] B. P. Douglass. "Doing Hard Time: Developing Real-Time System with UML, Objects, Frameworks, and Pattern". Addison-Wesley, New York. 1999.
- [116] B. P. Douglass. "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems". Addison-Wesley, New York, Boston, MA, USA. 2002.

- [117] R. Hanmer. "Patterns for Fault Tolerant Software". Wiley Publishing. 2007.
- [118] M. Pont. "Patterns for Time-Triggered Embedded Systems: Building reliable applications with the 8051 family of microcontrollers". ACM Press, New York, 2001.
- [119] L. Rising. "Design patterns in communications software". Cambridge University Press, New York, NY, USA. 2001.
- [120] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. "Security Patterns: Integrating Security and Systems Engineering". John Wiley & Sons, March 2006.
- [121] N. Yoshioka, H. Washizaki, and K. Maruyama. "A survey on security patterns". Progress in Informatics, 5. pp. 35–47. 2009.
- [122] A. Armoush. "Design Patterns for Safety-Critical Embedded Systems". Doctoral Dissertation, Universitätsbibliothek. RWTH Aachen. 2010.
- [123] T.P. Kelly. "Arguing Safety – A Systematic Approach to Managing Safety Cases". PhD Thesis. Department of Computer Science. The University of York. 1998.
- [124] R. Weaver. "The Safety of Software – Constructing and Assuring Arguments". PhD Thesis, Department of Computer Science – The University of York. 2003.
- [125] F. Ye. "Justifying the Use of COTS Components within Safety Critical Applications". PhD Thesis, Department of Computer Science – The University of York. 2005.
- [126] UK Health and Safety Executive (HSE). "Risk Management: ALARP at a glance". <http://www.hse.gov.uk/risk/theory/alarpglance.htm> . 2014. (Retrieved March 31st, 2015).
- [127] L. Bass, P. C. Clements and R. Kazman. "Software Architecture in Practice." 2nd edition, Addison-Wesley. London. 2003.
- [128] L. Bass, M. Klein and F. Bachmann. "Quality Attribute Design Primitives". Technical Report, CMU/SEI-2000- TN-017, Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA. 2000.
- [129] I. Habli and T. Kelly, "Process and Product Certification Arguments – Getting the Balance Right," SIGBED Review, vol. 3, no. 4, pp. 1–8. 2006.
- [130] C. Ainhauser, R. Trindade and V. Rupanov. ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D3.6.b "Safety Code Generator Specification". The SAFE / Safe-E Consortium. www.safe-project.eu. 2013.
- [131] National Aeronautics and Space Agency – NASA – STD. "8719. 13 B – Software Safety Standard." <http://www.hq.nasa.gov/office/codeq/doctree/871913.htm>. 2004. (Retrieved March 31st, 2015).
- [132] The SAFE Consortium. Deliverable D3.5.d "Final release of System, SW, HW reference meta-model definition". ITEA2 European research project – www.safe-project.eu. 2014.

- [133] K. Wolf and C. Liu. "New Clients with Old Servers: A Pattern Language for Client / Server Frameworks". In *Pattern Languages of Program Design*, J.Coplien and D. Schmidt, Eds. pp. 51-64. Reading, MA. Addison-Wesley. 1995.
- [134] B. Rubel. "Patterns for Generating a Layered Architecture". In *Pattern Languages of Program Design*, J.Coplien and D. Schmidt, Eds. pp. 119-128. Reading, MA. Addison-Wesley. 1995.
- [135] S. Adams. "Functionality Ala Carte". In *Pattern Languages of Program Design*, J.Coplien and D. Schmidt, Eds. pp. 7-8. Reading, MA. Addison-Wesley. 1995.
- [136] R. Lajoie and R. Keller. "Design and Reuse in Object-Oriented Frameworks: Patterns, Contracts and Motifs in Concert". In *Pattern Languages of Program Design*, J.Coplien and D. Schmidt, Eds. pp. 295-312. Reading, MA. Addison-Wesley. 1995.
- [137] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and A. Shlomo. "A Pattern Language: Towns, Buildings, Construction". New York: Oxford University Press, 1977.
- [138] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. "Object-Oriented Modelling and Design". (Vol. 199, No.1). Englewood Cliffs: Prentice-Hall. 1991.
- [139] G. Booch. "Object-Oriented Analysis and Design with Applications". Redwood City: The Benjamin/Cummings Publ. Comp., 2nd ed. 1. 1994.
- [140] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. "Object-Oriented Software Engineering - A Use Case Driven Approach". Addison-Wesley. 1992.
- [141] Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). "Design science in information systems research". *MIS Quarterly*, 28(1), 75–105.
- [142] Iivari, J. (2007). "A paradigmatic analysis of information systems as a design science". *Scandinavian Journal of Information Systems*, 19(2), 39–64.
- [143] A. H. Maslow. "The Psychology of Science." p. 15. 1966.
- [144] D. Schmidt, Model-driven engineering, *IEEE comput.* 39 (2) (2006) 41–47.
- [145] I. Crnkovic, M.R.V. Chaudron, S. Larsson, Component-based development process and component lifecycle, in: *Proceedings of the International Conference on Software Engineering Advances, ICSEA 2006*, p. 44. IEEE Computer Society. 2006.
- [146] W. Frakes, K. Kang. "Software reuse research: Status and future". *IEEE Trans. Softw. Eng.* 31 (7) pp. 529–536. 2005.
- [147] B. Hamid "A Model-Driven Approach for Developing a Model Repository: Methodology and Tool Support." *Future Generation Computer Systems*, vol. 68. pp. 473-490. ISSN 0167-739X. 2017.
- [148] R. Hawkins, K. Clegg, R. Alexander, T. Kelly. "Using a Software Safety Argument Pattern Catalog: Two Case Studies". In F. Flammini, S. Bologna,

- and V. Vittorini (Eds.): SAFECOMP 2011, LNCS 6894, pp. 185–198. Springer, Heidelberg. 2011.
- [149] J. Rushby. "Formalism in safety cases". In *Making Systems Safer*, pp. 3-17. Springer London. 2010.
- [150] I. Habli and T. Kelly. "Balancing the Formal and Informal in Safety Case Arguments." *VeriSure: Verification and Assurance Workshop*. 2014.
- [151] L. L. Pullum. "Software fault tolerance techniques and implementation". Norwood, MA, USA. Artech House Inc. 2001.
- [152] I. Koren and C. M. Krishna. "Fault-Tolerant Systems". San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. 2007.
- [153] R. Hanmer. "Patterns for Fault Tolerant Software". Wiley Publishing. 2007.
- [154] H. Gomaa and M. Hussein. "Model-based software design and adaptation". In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, p. 7. IEEE Computer Society. 2007.
- [155] E. Verhulst, and B. H.C. Sputh. "ARRL: A Criterion for Composable Safety and Systems Engineering". *Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security SAFECOMP*. 2013.
- [156] E. Denney and G. Pai. "A lightweight methodology for safety case assembly." In *Computer Safety, Reliability, and Security*. pp. 1-12. Springer Berlin Heidelberg. 2012.
- [157] E. Denney, G. Pai, and J. Pohl. "Advocate: An assurance case automation toolset". In *proc. of Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 31st International Conference on Computer Safety, Reliability and Security SAFECOMP*. pp 8-21. 2012.
- [158] E. Armengaud. "Automated safety case compilation for product-based argumentation". *Embedded Real Time Software and Systems Conference (ERTS 2014)*, Toulouse, France. 2014.
- [159] P. Oliveira Antonino, T. Keuler, and E. Y. Nakagawa. "Towards an Approach to Represent Safety Patterns". In *ICSEA 2012, The Seventh International Conference on Software Engineering Advances*, pp. 228-237. 2012.
- [160] R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly. "Weaving an Assurance Case from Design: A Model-Based Approach". *16th IEEE International Symposium on High Assurance Systems Engineering – HASE2015*. Florida. 2015.
- [161] A. Hauge and K. Stølen, "An analytic evaluation of the SaCS pattern language—Including explanations of major design choices", *PATTERNS 2014*, pp. 79-88. 2014

- [162] A. A. Hauge. “SaCS: A Method and a Pattern Language for the Development of Conceptual Safety Designs” Doctoral Dissertation. Series of dissertations submitted to the Faculty of Mathematics and Natural Sciences, University of Oslo. No. 1568. ISSN 1501-7710. 2014.
- [163] B. Hamid, SEMCO Project, System and software Engineering for embedded systems applications with Multi-CONcerns support. 2017.
<http://www.semcomdt.org>.
- [164] Gallina, B. (2015). “Towards enabling reuse in the context of safety-critical product lines”. In *Proceedings of the Fifth International Workshop on Product Line Approaches in Software Engineering (PLEASE '15)*. IEEE Press, 15–18.
- [165] Gleirscher, Mario & Kugele, Stefan. (2016). A Study of Safety Patterns: First Results. 10.13140/RG.2.2.23347.22562.
- [166] Cloutier, R., & Verma, D. (2007). “Applying the concept of patterns to systems architecture”. *Systems Engineering*, 10(2), 138-154.
<https://doi.org/10.1002/sys.20070>
- [167] Cloutier, R. (2006). “Applicability of patterns to architecting complex systems”. Doctoral dissertation, Stevens Institute of Technology.
<https://doi.org/10.1109/SYBOSE.2006.1652297>
- [168] Cloutier, R., Muller, G., & Verma, D. (2010). “The concept of reference architectures”. *Systems Engineering*, 13(1), 14-27.
<https://doi.org/10.1002/sys.20129>
- [169] Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley.
- [170] Nakagawa, E. Y., Oliveira, P., & Becker, M. (2011). “Reference Architecture and Product Line Architecture: A Subtle but Critical Difference”. In *Proceedings of the 5th European Conference on Software Architecture (ECSA 2011)*, Essen, Germany, 207–211.
https://doi.org/10.1007/978-3-642-23798-0_22
- [171] Haskins, C. (2008). “Using Patterns to Transition Systems Engineering from a Technological to Social Context.” INCOSE Publications, 11(2), 147–155. DOI: 10.1002/sys.20091.
- [172] Schindel, W. N., & Peterson, T. (2013). “Introduction to Pattern-Based Systems Engineering (PBSE): Leveraging MBSE Techniques.” In *Proceedings of the 23rd INCOSE International Symposium*, 23(1), 1639–1653. DOI: 10.1002/j.2334-5837.2013.tb03084.x.
- [173] Pfister, F., Chapurlat, V., Huchard, M., & Nebut, C. (2011). “A Design Pattern Meta-Model for Systems Engineering.” In *Proceedings of the 18th IFAC World Congress*, vol. 44, no. 1, pp. 11967–11972. DOI: 10.3182/20110828-6-IT-1002.03005.
- [174] Ernadote, D. (2015). “An Ontology Mindset for System Engineering.” In *Proceedings of the IEEE International Symposium on Systems Engineering (ISSE 2015)*, pp. 454–460. DOI: 10.1109/SysEng.2015.7302797.

- [175] Ernadote, D. (2017). "Ontology-Based Pattern for System Engineering." In *Proceedings of the 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2017)*, pp. 248–258. DOI: 10.1109/MODELS.2017.4.
- [176] Husung, S., & Stetter, R. (2016). „Modellbasierte Lösungsmuster zur Verbesserung der Wiederverwendbarkeit im Systems Engineering“ [Model-based solution patterns to improve reusability in Systems Engineering]. In: Stetter, R., & Schmidt, G. (Eds.), *Modellbasierte virtuelle Produktentwicklung*. Springer Vieweg, Berlin, Heidelberg, pp. 133–146. https://doi.org/10.1007/978-3-662-48413-4_9
- [177] Li, Z., Faheem, F., & Husung, S. (2023). *Systematic use of model-based solution patterns using the example of a load cell*. Proceedings of the 60th Ilmenau Scientific Colloquium, Technische Universität Ilmenau.
- [178] Anacker, H., Dumitrescu, R., Kharatyan, A., & Lipsmeier, A. (2020). "Pattern Based Systems Engineering – Application of Solution Patterns in the Design of Intelligent Technical Systems." *Proceedings of the Design Society: DESIGN Conference*, 1, 1195–1204. <https://doi.org/10.1017/dsd.2020.107>
- [179] Husung, S., Zimmermann, K., Oberender, C., & Lindemann, C. (2023). "Collaborative Model-Based Systems Engineering Using Dataspaces and SysML v2". *IEEE International Symposium on Systems Engineering (ISSE 2023)*. <https://doi.org/10.1109/ISSE59840.2023.10426152>
- [180] Wu, Q., Gouyon, D., Levrat, É., & Boudau, S. (2020). "Use of Patterns for Know-How Reuse in a Model-Based Systems Engineering Framework." *IEEE Systems Journal*, 14(4), 4765–4776. <https://doi.org/10.1109/JSYST.2020.2975116>
- [181] Wu, Q. (2019). "A model-based systems engineering approach for reuse and capitalization of engineering know-how through patterns." *Doctoral dissertation*, Université de Lorraine.
- [182] Wu, Q., Gouyon, D., Boudau, S., & Levrat, É. (2019). "Capitalization and Reuse with Patterns in a Model-Based Systems Engineering (MBSE) Framework." In *Proceedings of the IEEE International Symposium on Systems Engineering (ISSE 2019)*, pp. 1–8. DOI: 10.1109/ISSE46696.2019.8984571.
- [183] Khalil, M. (2013) "Pattern-Based Methods for Model-Based Safety-Critical Software Architecture Design: A PhD Thesis Proposal." In *Software Engineering – SE Aachen 2013*. ZeMoSS (Workshops). pp. 493-499. 2013.
- [184] M. Khalil. "Design Patterns to the rescue: guided model-based reuse for automotive solutions". 25th ACM Pattern Languages of Programming Conference (PLoP2018). Portland, Oregon, USA. 2018.
- [185] Wu, Q., Gouyon, D., & Levrat, É. (2021). "Maturity Assessment of Systems Engineering Reusable Assets to Facilitate MBSE Adoption." 17th *IFAC Symposium on Information Control Problems in Manufacturing (INCOM*

- 2021). Budapest, Hungary. IFAC-PapersOnLine, 54(1), 851–856. DOI: 10.1016/j.ifacol.2021.08.187.
- [186] Object Management Group. *Unified Architecture Framework® (UAF®)*. 2020. Object Management Group, <https://www.omg.org/uaf/>.
- [187] M.Didonet Del Fabro, J.Bezivin, F. Jouault, E. Breton, and G.Gueltas. “AMW: A generic Model Weaver”. In International Conference on Software Engineering Research and Practice (SERP05). 2005.
- [188] Di Ruscio, D. (2007). "Specification of model transformation and weaving in model driven engineering." PhD thesis, Universita degli Studi dell'Aquila (February 2007). <http://www.di.univaq.it/diruscio/phdThesis.php>
- [189] Bur, M.. *Digital Engineering: Concepts, Methodologies and Implementation*. PhD Thesis, Norwegian University of Science and Technology, 2019.
<https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2633544>
- [190] SAIC. “Digital Engineering: Model Federation and the Future of Systems Engineering.” *SAIC Blog*, 2021. <https://www.saic.com/features/digital-engineering>
- [191] Modak. *Data Fabric: A New Architectural Approach for Data-Driven Enterprises*. Whitepaper, 2023.
<https://www.modak.com/resources/whitepapers/data-fabric-whitepaper>
- [192] Boudjemila, A., et al. "Towards a Model-Based Interoperability Framework for Digital Engineering." *International Workshop on Model-Driven Engineering for Digital Twins*. In Proceedings of the 2023 ACM/IEEE MODELS 26th International Conference on Model Driven Engineering Languages and Systems. Sweden. 2023.
- [193] W. Zimmer, “Relationships between Design Patterns,” in Pattern Languages of Program Design. Addison-Wesley, 1994, pp. 345–364.
- [194] J. Noble, “Classifying Relationships between Object-Oriented Design Patterns,” in Proc. of Australian Software Engineering Conference (ASWEC’98), pp. 98–107. 1998.
- [195] H. Mili, G. El-Boussaidi. “Representing and applying design patterns: what is the problem?” *Model Driven Engineering Languages and Systems*, Lecture Notes in Computer Science. Volume 3713, pp 186-200. Springer Berlin Heidelberg. 2005.
- [196] M. Khalil, A. Prieto and F. Hölzl. "A pattern-based approach towards the guided reuse of safety mechanisms in the automotive domain." In *Model-Based Safety and Assessment – IMBSA 2014*. pp. 137-151. Springer International Publishing. 2014.
- [197] M. Khalil and E. Metzker. ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D3.1.3b / D3.4b “Updated proposal for extension of Meta-model for safety-case modeling and documentation”. The SAFE / Safe-E Consortium – www.safe-project.eu. 2014.

- [198] Prieto Rodriguez, A. A. (2015). "Exploration of a pattern-based approach for the reuse of safety mechanisms in embedded systems." Master's Thesis. Technical University of Munich. Submitted on Sept. 15th, 2015.
- [199] A. Bauer, M. Broy, J. Romberg, B. Schätz, P. Braun, U. Freund, N. Mata, R. Sander, and D. Ziegenbein. "Automode - notations, methods, and tools for model-based development of automotive software". No. 2005-01-1281. SAE Technical Paper. 2005.
- [200] S.Voss, B. Schätz, M. Khalil, C. Carlan. "Towards Modular Certification using integrated model-based Safety Cases". VeriSure 2013, Verification and Assurance Workshop, at CAV 2013. St. Petersburg, Russia. 2013.
- [201] S. Voss and B Schätz. "Deployment and scheduling synthesis for mixed-critical shared-memory applications". In: Engineering of Computer Based Systems (ECBS), 2013. 20th IEEE International Conference and Workshops on the. pp. 100-109. 2013.
- [202] C. Carlan. "Implementierung unterschiedlicher Redundanzkonzepte zur automatischen Generierung entsprechender logischer Strukturen für ein sicheres Verhalten gemischt-kritischer Systeme im CASE-Tool AutoFocus3". Bachelor Thesis (German). Technische Universität München. Faculty of Informatics. Chair of Software and Systems Engineering. 2012. http://download.fortiss.org/public/carlan/BA_Carmen_Carlan.pdf
- [203] M. Khalil. "Pattern libraries guiding the model-based reuse of automotive solutions". 10th International Conference on Systems Analysis Modeling (SAM2018). Copenhagen, Denmark. 2018.
- [204] S. Teufl, D. Mou and D. Ratiu. "Mira: A tooling-framework to experiment with model-based requirements engineering". In Requirements Engineering Conference (RE), 2013 21st IEEE International (pp. 330-331). IEEE. 2013.
- [205] D. Mou and D. Ratiu. "Binding requirements and component architecture by using model-based test-driven development". In *Twin Peaks of Requirements and Architecture (Twin Peaks)*, 2012 IEEE First International Workshop on the (pp. 27-30). IEEE. 2012.
- [206] Zverlov, S., Khalil, M., and Chaudhary, M. (2016). "Pareto-efficient deployment synthesis for safety-critical applications". Embedded Real Time Software and Systems Conference (ERTS 2016), Toulouse, France. 2016.
- [207] L. Forest, D. Picard, A. Gauthier and V. Rupanov. ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D4.2.7.b / D4.5b "Plug-In for Safety Code Generation". The SAFE / Safe-E Consortium. www.safe-project.eu. 2014.
- [208] J. Kemmerich and M. Khalil (Eds.). ITEA/EUROSTARS SAFE / Safe-E Project. Deliverable D6.b / D6.2b "Methodology and application rules Documentation". The SAFE / Safe-E Consortium. www.safe-project.eu. 2014.
- [209] The EAST-ADL Association. <http://www.east-adl.info/>
- [210] The AUTOSAR Association. www.autosar.org.

- [211] J. Birch, R. Rivett, I. Habli, B. Bradshaw, J. Botham, D. Higham, P. Jesty, H. Monkhouse, and R. Palin. "Safety cases and their role in ISO 26262 functional safety assessment". In *Computer Safety, Reliability, and Security*, pp. 154-165. Springer Berlin Heidelberg. 2013.
- [212] S. Toulmin. "Uses of Argumentation". Cambridge University Press, 1958; 2nd ed. 2003.
- [213] The CHROMOSOME Middleware project.
<http://www.fortiss.org/en/research/projects/chromosome/>. fortiss GmbH. 2015.
- [214] Yin, R. K. (2018). "Case study research and applications: Design and methods" (6th ed.). SAGE Publications.
- [215] Runeson, P., & Höst, M. (2009). "Guidelines for conducting and reporting case study research in software engineering". In *Journal of Empirical Software Engineering*, 14(2), 131–164. <https://doi.org/10.1007/s10664-008-9102-8>
- [216] Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). "Selecting empirical methods for software engineering research." In *Guide to Advanced Empirical Software Engineering* (pp. 285–311). Springer.
https://doi.org/10.1007/978-1-84800-044-5_11
- [217] J. Eder, S. Zverlov, S. Voss, M. Khalil and A. Ipatiov. "Bringing DSE to life: exploring the design space of an industrial automotive use case." ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017). Austin, Texas, USA. 2017.
- [218] Systems Architecture Framework (SAF). GFSE – German Chapter of INCOSE. <https://saf.gfse.org/>. Accessed 18 Apr. 2025.
- [219] Khalil, M. (2023). "How much systems modeling do I really need in the era of the Software-Defined-Vehicle?" *Keynote presented at the MDENet Symposium 2023*. Retrieved from https://mde-network.com/wp-content/uploads/2023/12/MDENetSymposium2023_Keynote_Continental_MKhalil-1.pdf
- [220] J. Eder, A. Bayha, S. Voss, A. Ipatiov, and M. Khalil. "From Deployment to Platform Exploration: Automatic Synthesis of Distributed Automotive Hardware Architectures". ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018). Copenhagen, Denmark. 2018.
- [221] J. Eder, S. Voss, A. Bayha, A. Ipatiov, and M. Khalil. "Hardware architecture exploration: Automatic exploration of distributed automotive hardware architectures." In *Journal of Software and Systems Modeling* (SOSYM). Vol. 19. 911–934. 2020.
- [222] International Organization for Standardization and International Electrotechnical Commission. (2021). "ISO/IEC 26580:2021: Software and Systems Engineering — Methods and Tools for the Feature-Based Approach to Software and Systems Product Line Engineering." ISO, 2021.

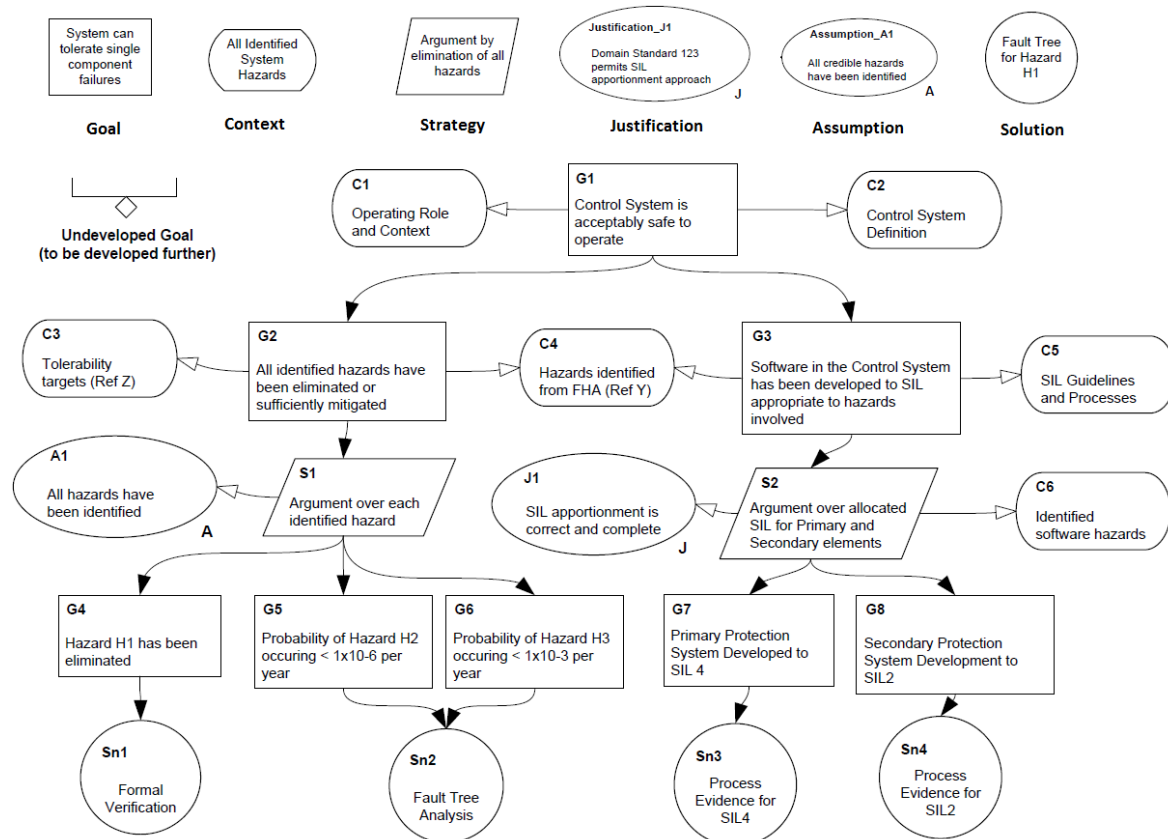
- [223] Fowler, F. J. Jr. (2013). *Survey Research Methods* (5th ed.). SAGE Publications. <https://us.sagepub.com/en-us/nam/survey-research-methods/book235902>
- [224] Patton, M. Q. (2014). *Qualitative Research & Evaluation Methods* (4th ed.). SAGE Publications. <https://us.sagepub.com/en-us/nam/qualitative-research-evaluation-methods/book232962>
- [225] Kitchenham, B. A., & Pfleeger, S. L. (2002). "Principles of Survey Research Part 3: Constructing a Survey Instrument". *ACM SIGSOFT Software Engineering Notes*, 27(2), 20–24. <https://doi.org/10.1145/511152.511155>
- [226] Olson, K. (2010). *An Examination of Questionnaire Evaluation by Expert Reviewers*. *Field Methods*, 22(4), 295–318. <https://doi.org/10.1177/1525822X10379795>
- [227] Wohlin, C., & Runeson, P. (2021). *Experimentation in Software Engineering* (2nd ed.). Springer International Publishing. <https://link.springer.com/book/10.1007/978-3-030-58164-0>
- [228] Podsakoff, P. M., MacKenzie, S. B., Lee, J.-Y., & Podsakoff, N. P. (2003). „Common method biases in behavioral research: A critical review of the literature and recommended remedies.” *Journal of Applied Psychology*, 88(5), 879–903. <https://doi.org/10.1037/0021-9010.88.5.879>
- [229] Israel, M., & Hay, I. (2006). "Research ethics for social scientists: Between ethical conduct and regulatory compliance." Sage Publications.
- [230] Chen, Z., Zhong, P., Liu, M., Ma, Q., & Si, G. (2022). „An integrated expert weight determination method for design concept evaluation”. *Scientific Reports*, 12, Article 6358. <https://doi.org/10.1038/s41598-022-10333-6>
- [231] Dillman, D. A., Smyth, J. D., & Christian, L. M. (2014). *Internet, Phone, Mail, and Mixed-Mode Surveys: The Tailored Design Method* (4th ed.). John Wiley & Sons.
- [232] Singer, J., and Vinson, N. G. (2002). "Ethical Issues in Empirical Studies of Software Engineering." *IEEE Transactions on Software Engineering*, vol. 28, no. 12, 2002, pp. 1171–1180. DOI: 10.1109/TSE.2002.1158289.
- [233] Dillman, D. A., Smyth, J. D., & Christian, L. M. (2014). *Internet, Phone, Mail, and Mixed-Mode Surveys: The Tailored Design Method* (4th ed.). Wiley.
- [234] Groves, R. M., Fowler, F. J., Couper, M. P., Lepkowski, J. M., Singer, E., & Tourangeau, R. (2009). *Survey Methodology* (2nd ed.). Wiley.
- [235] Krosnick, J. A. (1991). "Response Strategies for Coping With the Cognitive Demands of Attitude Measures in Surveys." *Applied Cognitive Psychology*, 5(3), 213–236.
- [236] Tourangeau, R., Rips, L. J., & Rasinski, K. (2000). *The Psychology of Survey Response*. Cambridge University Press.
- [237] Couper, M. P., Traugott, M. W., & Lamias, M. J. (2007). Web survey design and administration. *Public Opinion Quarterly*, 65(2), 230–253. <https://doi.org/10.1086/322199>

- [238] Cooke, R. M., & Goossens, L. H. J. (2008). "Expert judgment elicitation using the classical model." *Reliability Engineering & System Safety*, 93(5), 755–777.
- [239] Aspinall, W. P. (2010). "A route to more tractable expert advice." *Nature*, 463(7279), 294–295.
- [240] Liu, H., & Hu, Y. (2015). "An evaluating method with combined assigning-weight based on maximizing variance". *Scientific Programming*, 2015, Article ID 290379.
- [241] Schein, E. H. (2010). *Organizational Culture and Leadership* (4th ed.). Jossey-Bass.
- [242] Denison, D. R., & Mishra, A. K. (1995). *Toward a theory of organizational culture and effectiveness*. *Journal of Organization Science*, 6(2), 204–223. <https://doi.org/10.1287/orsc.6.2.204>
- [243] Jiang, J. J., Klein, G., Hwang, H.-G., Huang, J., & Hung, S.-Y. (2004). *An exploration of the relationship between software development process maturity and project performance*. *Journal of Information & Management*, 41(3), 279–288. [https://doi.org/10.1016/S0378-7206\(03\)00052-1](https://doi.org/10.1016/S0378-7206(03)00052-1)
- [244] Frakes, W. B., & Kang, K. (2005). *Software reuse research: Status and future*. *IEEE Transactions on Software Engineering*, 31(7), 529–536. <https://doi.org/10.1109/TSE.2005.85>
- [245] Li, Y., Conradi, R., Slyngstad, O. P. N., Bunse, C., Torchiano, M., & Morisio, M. (2015). *A state-of-the-practice survey of risk management in development with off-the-shelf components*. *IEEE Transactions on Software Engineering*, 34(2), 271–286. <https://doi.org/10.1109/TSE.2008.101>
- [246] Flyvbjerg, B. (2006). *Five Misunderstandings about Case-Study Research*. *Qualitative Inquiry*, 12(2), 219–245. <https://doi.org/10.1177/1077800405284363>
- [247] Maged Khalil. 2019. "Improving solution reuse in automotive embedded applications using a pattern library-based approach". In *Proceedings of the 2019 ACM/IEEE MODELS 22nd International Conference on Model Driven Engineering Languages and Systems Companion* 653–659.
- [248] ITEA/EUROSTARS SAFE / Safe-E Project. D3.6 "Safety Mechanism Classification and Semantics Definition". The SAFE / Safe-E Consortium – www.safe-project.eu. 2014.

Appendices

A1 Appendix 1 – Additional Details on the Goal Structuring Notation – GSN

The following sections describe the basic structure and usage of the GSN standard, with the majority of the information and texts originating in the GSN Community Standard Version 1.0 [58], published in November 2011. This information is included here as it is relevant for facilitating readers' understanding of some of the implementation examples and results presented in this thesis. For the latest version of the GSN standard, please visit the Safety Critical Systems Club website. All other text blocks were generated by the author for the ITEA2/EUROSTARS SAFE/Safe-E Project Deliverable D3.1.3/D3.4 [57] and its update D3.1.3b/D3.4b [197], and have been copied from those documents directly into this appendix.



The Goal Structuring Notation – Elements and Example Argument Structure [58]

A1.1 Safety Case Elements

GSN uses the following basic elements to describe the role of artifacts within a safety case:

Goal

- rendered as a rectangle,
- presents a claim forming part of the argument.

- One or more sub-goals may be declared for a given goal. This structure then asserts that if the claims presented in the sub-goals are true, this is sufficient to establish that the claim in the main goal is true.

Strategies

- rendered as a parallelogram,
- describe the nature of the inference that exists between a goal and its supporting goal(s)
- and are used to describe the nature of the inference which is asserted as existing between sub-goals and the parent goal.

Solutions

- A **solution**, rendered as a circle, presents a reference to an evidence item or items.
- Multiple solutions may satisfy a goal.
- Multiple goals may be satisfied by one solution.

Contexts

- Claims can only be asserted to be true in a specified context. Context elements are used to make this relationship clear.
- A **context**, rendered as an oblong rectangle with rounded out sides, presents a contextual artifact. This can be a reference to contextual information, or a statement.
- Where used, contexts define or constrain the scope over which the claim is made.

Assumptions

- An assumption is an intentionally unsubstantiated statement. The scope of an assumption is the entire argument. Having connected an assumption to a goal, the assumption is taken to be connected to the entirety of the argument supporting this goal.
- Therefore, it is not necessary to restate the assumption in the supporting argument.
- rendered as an oval with the letter 'A' at the bottom-right.

Justifications

- rendered as an oval with the letter 'J' at the bottom-right,
- presents a statement of rationale,
- and does not alter the meaning of the claim made in the goal, but provides rationale for its inclusion or its phrasing.
- Should an equivalent justification be required elsewhere in the argument, it will need to be re-stated or re-linked.

Furthermore, it is possible to represent **undeveloped entities**, rendered as a hollow diamond applied to the center of an element, indicating that a line of argument has not been developed. It can apply to goals (as below) and strategies.

A specific instance of *undeveloped entities* is an *undeveloped goal*, rendered as a rectangle with the hollow-diamond ‘undeveloped entity’ symbol at the center-bottom, presenting a claim which is intentionally left undeveloped in the argument.

A1.2 Safety Case Element Relations and Possible Expressions

The core elements described in the previous section are linked using the following types of relationships:

SupportedBy

- rendered as a line with a solid arrowhead, allows inferential or evidential relationships to be documented.
- Inferential relationships declare that there is an inference between goals in the argument. Evidential relationships declare the link between a goal and the evidence substantiating it.
- Permitted connections are:
 - goal-to-goal,
 - goal-to-strategy,
 - goal-to-solution,
 - strategy to goal.

InContextOf

- rendered as a line with a hollow arrowhead, declares a contextual relationship.
- Permitted connections are:
 - goal-to-context,
 - goal-to-assumption,
 - goal-to-justification,
 - strategy-to-context,
 - strategy-to-assumption
 - and strategy-to-justification.

A1.3 Steps for building a safety case goal structure

The works of Kelly and McDermid, such as , and culminating in the GSN standard, are based on well-revised argumentation works and suggest a multi-step approach to formulating goal structures and hence building safety cases, as seen in the following subsections.

Top Down: When classically working from Safety Goals

For a classic staged approach to safety arguments, Kelly defines six steps in the top-down development of a goal structure:

1. Identify the goals to be supported;
2. Define the basis on which the goals are stated;
3. Identify the strategy used to support the goals;
4. Define the basis on which the strategy is stated;
5. Elaborate the strategy (and proceed to identify new goals – back to step 1), or step 6;
6. Identify the basic solution.

Bottoms-Up: When working from existing evidence

When analyses evidence already exists or when an existing safety case needs updating, it is possible to adapt Kelly's six steps for top-down GSN development, into a process which can be used to develop a goal structure from the bottom up:

1. Identify evidence to present as GSN solutions;
2. Infer 'evidence assertion' claims to be directly supported by these solutions, and present these as GSN goals;
3. Derive higher-level sub-goals that are supported by the evidence assertions;
4. Describe how each layer of sub-goals satisfies the parent goal (i.e. strategy);
5. Check that any necessary contextual information is included;
6. Check back down the structure for completeness;
7. Join the resulting goal structure to a known top goal or a set of sub-goals.

Using these methods in conjunction with the steps explained in our approach, it is possible to generate and document safety cases from model elements directly.

A1.4 On using Safety Cases

Safety cases can have more than one role, depending on when and how they are generated and used. Each of these roles defines what types of tests can be performed and to which means.

Descriptive Safety Cases

Use:

When goal structures are generated to argue about existing products or during products development a safety case evolves which describes the safety of the system under development and the fulfillment of its safety goals. This safety case describes the status quo and aims at providing a clear and justifiable argument that the safety goals are met.

Testing:

Using the set of relation rules identified earlier, e.g., each **Goal** must resolve to a **Solution**, in a model-based development environment it is possible to perform completeness and consistency checks on this type of safety case.

Prescriptive Safety Cases

Use:

In conjunction with the use of patterns, once a decision is made to solve a known problem or use a trusted solution pattern or purchase a COTS component, the developer can directly create an instance of the (partial) safety case template or skeleton for the known pattern/component. It thus becomes possible to know what the safety case *should* look like and to use this as development guideline.

Testing:

Using the generated safety case skeleton as a reference, the developer can test existing solutions against it to see whether their arguments hold.

The extent of the testing depth and automation rely on how high the integration of the safety case elements into the development artifact landscape is.

A2 Appendix 2 – Classification of safety mechanisms at a logical level

This appendix provides deeper details into the implementations of the MODPAL approach in the AF3 Tool presented in Chapter 5.3.6. After the first implementations proved the feasibility of the approach, and were published in [196], the implementation work was expanded in the context of a Master's thesis [198] supervised by the author. The following text relays combined research supported by the author in a supervisory role, copied from that Master's thesis. Some minor corrections were made to rectify typos, grammar or linguistic mistakes impacting the understandability of the texts. The source citations were reduced to a bare minimum to avoid cross-contaminating this thesis's bibliography. Please refer to the master's thesis for a full list of the references used there.

Classification of safety mechanisms at a logical level

Safety mechanisms can be classified under three categories: *failure detection*, *failure avoidance* and *failure handling*. However, they can also be categorized at a logical level depending on the structure of their component and surrounding architecture. The result of applying such categorization was the mapping of the safety mechanisms to the following categories: *one logical component*, *composition of logical components* and *redundant component* safety mechanism. Before this classification was obtained, previous taxonomies were proposed.

For the *first* classification of safety mechanisms at a logical level a checklist was employed, to see which properties they cover. After applying the checklist to the list of safety mechanisms used in this master's thesis, the results were registered in tables. They can be observed in table A.1 and table A.2. The goal of these tables was to analyze the set of safety mechanisms, in order to find a way to group them in classes, depending on the properties that they have in common.

A2.1 Initial categorization of safety mechanisms at a logical level

This categorization is orthogonal to the one mentioned in the beginning, shown in [248]. The derived categories are listed and explained as follows:

Category 1

This category is characterized for:

- The safety mechanism is composed by one logical component.
- Its functional behavior will not change in the course of time.

The number of inputs remains unmodified. The safety mechanisms mapped to this category are:

- | | |
|--|----------------------------------|
| • <i>Challenge Response Check</i> | • <i>Maximum Age Monitoring</i> |
| • <i>Comparison</i> | • <i>Message Read Back Check</i> |
| • <i>Cyclic Redundancy Check (CRC)</i> | • <i>Parity Check</i> |
| • <i>Data Timeout Monitoring</i> | • <i>Replication</i> |
| • <i>Gradient Check</i> | • <i>Reporting</i> |
| • <i>Logical Control Flow Monitoring</i> | |

Category 2

This category distinguishes from the others in:

- The safety mechanism is composed by one logical component.
- Its functional behavior will not change in the course of time.
- The number of inputs varies depending on which were set to be active by the user. In other words, inputs can be optional.

The safety mechanisms mapped to this category are:

- *Partitioning*
- *Range Check*

Category 3

This category is characterized for fulfilling the following properties:

- The safety mechanism is composed by one logical component.
- The functional behavior of the safety mechanism must be defined by the user.
- The number of inputs remains unmodified.

The safety mechanisms mapped to this category are:

- *Control Flow Interlock*
- *Reset*
- *Data Sequence Monitor*
- *Watchdog*

Category 4

This category is characterized for:

- The safety mechanism is composed by one logical component.
- The functional behavior of the safety mechanism must be defined by the user.
- The number of inputs varies depending on the implemented functionality.

The set of safety mechanisms matching these properties is comprised by:

- *Alive Supervision*
- *Error Filtering*
- *Data Interlock*
- *HW-Self-Test*
- *Deadline Supervision*
- *SW-Self-Test*
- *Default Value*

Category 5

This category is determined by the following properties:

- The safety mechanism is composed by at least two logical components.
- Logical components can be new or reused (another safety mechanism).
- For each new logical component the user must define:
 - The type of functional behavior, fixed or variable.
 - Number of inputs, marking the ones that could be optional.
 - The logical components to which it is connected through a channel.

If a safety mechanism is reused, the following conditions have to be fulfilled:

- The safety mechanism must be selected from the existing ones, otherwise it must be added to one of the categories.

- Inputs and outputs must be linked to the respective logical components.

The safety mechanisms mapped to this category are:

- *1-out-of-2 Voting*
- *2-out-of-3 Voting*
- *3-Level Safety Monitoring*
- *Actuator Monitoring*
- *Heterogeneous Duplex Redundancy*
- *N-out-of-M Voting*
- *Sensor Correlation*
- *Sensor Rationality*

Category 6

This category is characterized for:

- The safety mechanism is composed by at least two logical components.
- The comprising logical components must be defined by the user. Therefore, the following aspects have to be determined for each logical component:
 - The type of functional behavior, fixed or variable.
 - Number of inputs, marking the ones that could be optional.
 - The logical components to which it is connected through a channel.
 - If the logical component is *initial* or *final*, where *initial* is denoted as the logical component that receive an input from the architecture in which is integrated. On the other hand, *final* is used to denote the logical component that delivers the processed data as output to the architecture.
 - At least one initial and final logical component has to be defined.

The safety mechanisms mapped to this category are:

- *Degradation*
- *Monitor Actuator*
- *Protected Single Channel*
- *Sanity Check*

Category 7

This category is determined by the following properties:

- The safety mechanism is composed by at least two logical components. One of the main components is called *duplicated*, which is a copy of an existing component (source) from the architecture in which the safety mechanism is going to be added. Therefore, a precondition tied to this category is the existence of a source component. The other main component is the *deciding component*, which decides, depending on the values received from both components mentioned before, the output to be delivered.
- Additional logical components to the main ones have to be defined by the user, or in the case that they are reused safety mechanism, they have to be selected from the existing categories. In case of a new logical component that following tasks have to be carried out:
 - The type of functional behavior, fixed or variable.
 - Number of inputs, marking the ones that could be optional.
 - The logical components to which it is connected through a channel.
- The duplication can be repeated n-times. The user of such safety mechanism must determine how much *deciding components* are going to be applied and how they are going to be connected within the safety mechanism logical architecture.

The safety mechanisms mapped to this category are:

- *Acceptance Voting*
- *Homogeneous Duplex Redundancy*
- *N-out-of-M Redundancy*
- *N-Self-checking*

- *N-Version-Programming*
- *Recovery Block*
- *Recovery Block with Backup Voting*
- *Triple Modular Redundancy*

The following tables capture the hierarchical classification of the two collected pattern groups into seven initial categories.

	Safety Mechanism																															
	Failure Avoidance				Failure Detection																		Failure Handling									
Property	Partitioning	DataInterlock	ControlFlowInterlock	Replication	ParityCheck	CRC	Comparison	SoftwareSelftest	HWSelftest	RangeCheck	MessageReadBackCheck	ChallengeResponseCheck	ActuatorMonitoring	SensorCorrelation	SensorRationalityCheck	GradientCheck	AnalyticRedundancy	LogicalControlFlowMonitoring	DataSequenceMonitor	AliveSupervision	DeadlineSupervision	DataTimeoutMonitoring	MaximumAgeMonitoring	1oo2Voting	2oo3Voting	DefaultValue	ErrorFiltering	ErrorCorrectionCode	Reporting	Reset	Degradation	
OneComponentSM	x	x	x	x	x	x	x	x	x	x	x	x				x		x	x	x	x	x	x				x	x				
CompositionOfComponents													x	x	x		x							x	x		x	x				x
DuplicationSM																																
FixedBehavior	x			x	x	x	x			x	x	x				x	x	x					x	x					x			
BehaviorDefinedByUser		x	x					x	x				x	x	x				x	x	x				x	x	x	x			x	x
FixedQInputs			x	x	x	x	x				x	x		x		x	x	x	x				x	x	x	x			x	x	x	
VariableQInputs	x	x						x	x	x			x		x					x	x						x	x				
DynamicInputs		x	x								x	x	x	x	x		x	x	x	x	x				x	x					x	x
StaticInputs	x							x	x																							
BothInputs				x	x	x	x			x						x							x	x			x	x		x		
ReuseOtherSM													x	x	x		x								x	x						
1:1 Duplication																		x														
DuplicateAndDefined																																
1Output	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					
MoreThanOneOutput																														x		

Table 23: Registered properties of safety mechanisms collected from [248]

	Safety Mechanism														
	Hardware								Software				Combination		
Property	HomogeneousDuplexRed	HeterogeneousDuplexRed	TriplerModularRed	M-Out-Of-N	Monitor-Actuator	SanityCheck	Watchdog	SafetyExecutive	N-Version Programming	RecoveryBlock	AcceptanceVoting	N-SelfChecking	RecoveryBlockWithBackupVoting	ProtectedSingleChannel	3-LevelSafetyMonitoring
OneComponentSM															
CompositionOfComponents	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
DuplicationSM	x	x	x	x					x	x	x	x	x		
FixedBehavior	x	x	x	x					x	x	x	x	x		
BehaviorDefinedByUser					x	x	x	x		x	x	x	x	x	x
FixedQInputs	x		x		x	x	x	x	x	x	x	x	x	x	x
VariableQInputs		x		x											
DynamicInputs	x	x	x	x	x	x			x	x	x	x	x	x	
StaticInputs															
BothInputs							x	x							x
ReuseOtherSM	x	x	x	x					x	x	x	x	x		x
1:1 Duplication	x		x	x					x	x		x	x		
DuplicateAndDefined		x								x					

Table 24: Registered properties of safety mechanisms collected from [122]

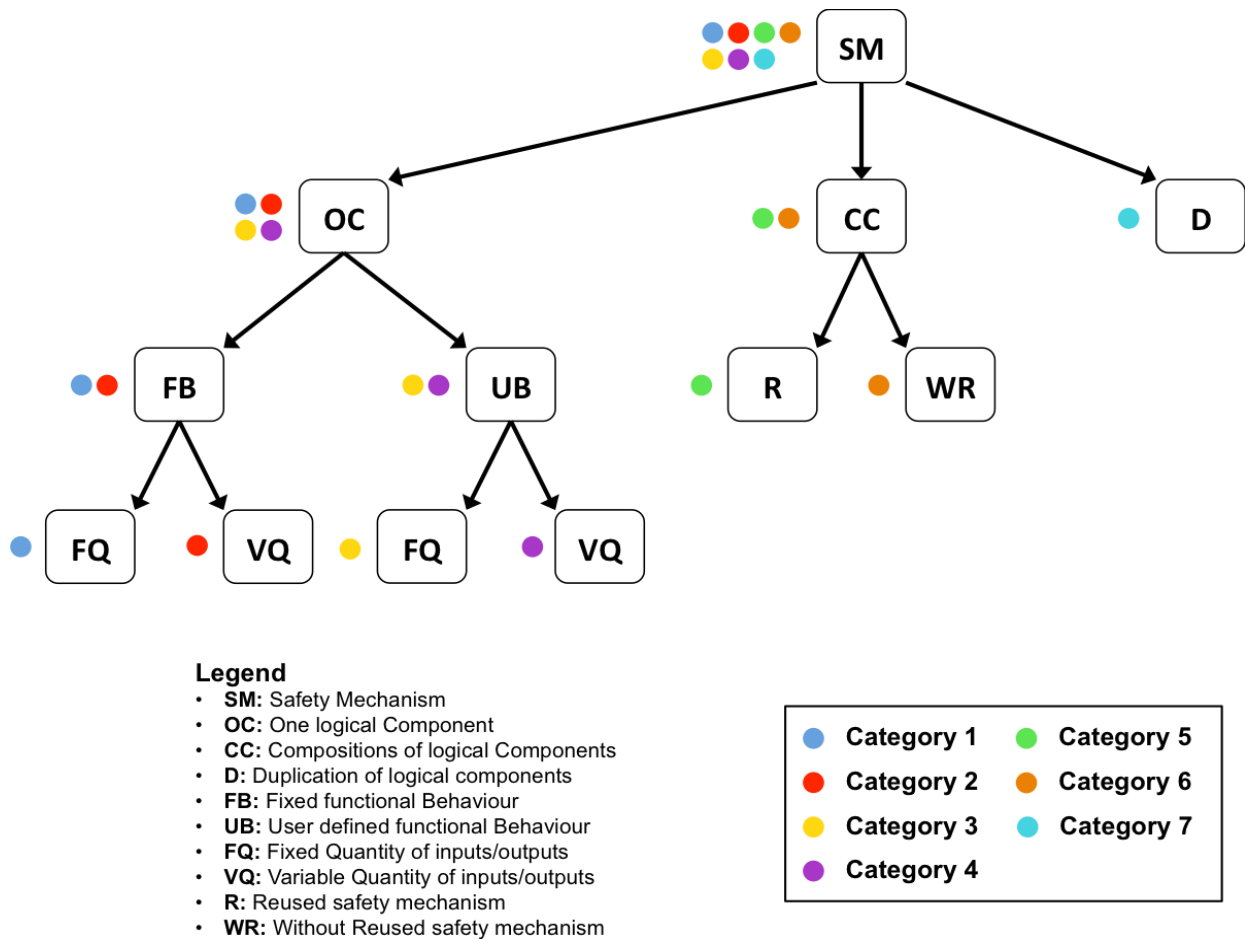


Figure 56: Initial hierarchical categorization of safety mechanisms at logical level

A2.2 Final categorization of safety mechanisms at a logical level

To have seven categories signifies that developers as well as users of safety mechanism patterns must handle an amount of highly detailed information at the time of documenting and/or using them. On the other hand, at the time of evaluating the work, needed to implement the functionalities in the CASE-tool AutoFOCUS 3 (AF3), it was realized that the integration was going to be too laborious. The construction of the wizard (procedure) used to apply a safety mechanism would be a nonessential investment of time and resources, due to its complexity. In the case of implementing the reuse process by following such classification, would also mean an unnecessary time employment on the part of users when interacting with AF3. To implement seven different varieties of a procedure implies that users must learn them, in order to successfully reuse a safety mechanism in a safety--critical embedded system.

To reduce the level of detailed contained within the categories, a couple of abstractions were carried out over the first classification. As a result, the number of categories was reduced to three: *one logical component*, *composition of logical components* and *redundant component* safety mechanism. This classification will have a significant influence over the proposed *augmentation reuse process* because it will determine how and in which order the logical components must be

instantiated, by making use of the existing logical components of the architecture of the embedded system.

The mapping of the safety mechanisms handled in this master's thesis to their respective category at a logical level is described below:

Category 1 – One logical component

This category is comprised by safety mechanisms composed by one logical component, independent from the surrounding architecture. The following safety mechanisms fall under this category:

- *Alive Supervision*
- *Challenge Response Check*
- *Comparison*
- *Control Flow Interlock*
- *CRC*
- *Data Interlock*
- *Data Sequence Monitor*
- *Data Timeout Monitoring*
- *Deadline Supervision*
- *Default Value*
- *Error Filtering*
- *Gradient Check*
- *HW-Self-Test*
- *Logical Control Flow Monitoring*
- *Maximum Age Monitoring*
- *Message Read Back Check*
- *Parity Check*
- *Partitioning*
- *Range Check*
- *Replication*
- *Reporting*
- *Reset*
- *SW--Self-Test*

Category 2 – Composition of logical components

This category is comprised by those safety mechanisms which internal structure consists in the interaction of several logical components. The contained logical components can belong to one of the three categories presented here. It means that safety mechanisms can also be part of those comprising logical components of the internal architecture of the main logical component. The following safety mechanisms can be classified under this category:

- *1-out-of-2 Voting*
- *2-out-of-3 Voting*
- *Degradation*
- *Sensor Correlation*
- *Sensor Rationality Check*

Category 3 – Redundant component

This category comprehends safety mechanisms that are used to employ redundancy within embedded systems. This classification has the same structure as the one from second category, because the safety mechanism is composed by several components. The difference is that one of those logical components is a replica of an existing one from the architecture of the embedded system. This means that the internal logical structure of the safety mechanism depends on the existence of an external logical component in order to function correctly. The safety mechanisms that match this description are:

- *3-Level Safety Monitoring*
- *Acceptance Voting*
- *Actuator Monitoring*
- *Analytic Redundancy*

- *Heterogeneous Duplex Redundancy*
- *Homogeneous Duplex Redundancy*
- *Monitor Actuator*
- *M-out-of-N*
- *N-Self-Checking*
- *N-Version Programming*
- *Protected Single Channel*
- *Recovery Block*
- *Recovery Block with Backup Voting*
- *Safety Check*
- *Safety Executive*
- *Triple Modular Redundancy*
- *Watchdog*

In the next figure the final consolidated hierarchical classification of safety mechanisms at a logical level is shown.

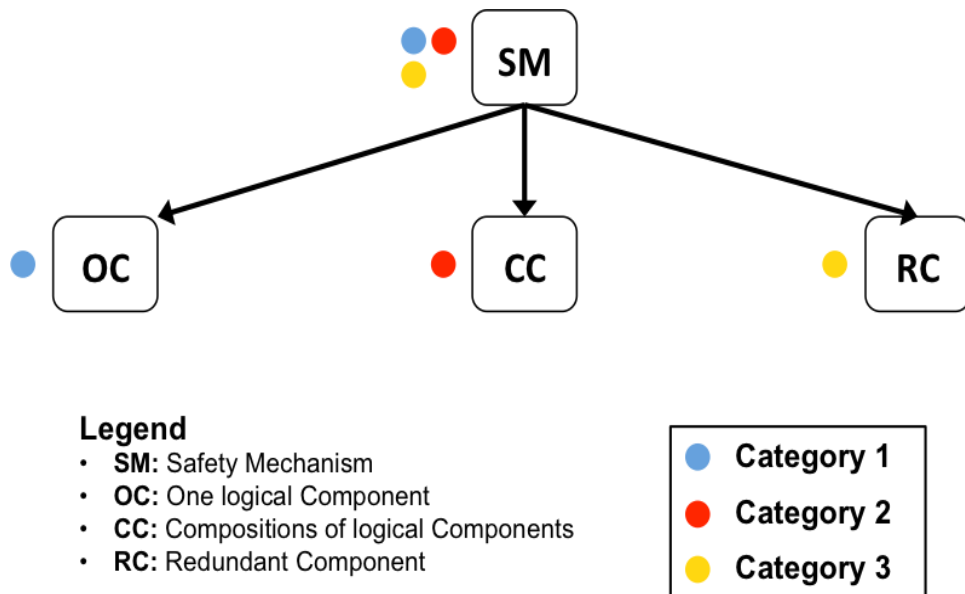


Figure 57: Final hierarchical categorization of safety mechanisms at logical level

A3 Appendix 3 Additional Implementation Examples in AF3

This appendix provides additional examples of the implementations of the MODPAL approach in the AF3 Tool to the ones presented in Chapter 5.3.6. After the first implementations proved the feasibility of the approach, and were published in [196], the implementation work was expanded in the context of a Master's thesis [198] supervised by the author. The following text relays combined research supported by the author in a supervisory role, copied from that Master's thesis. Some minor corrections were made to rectify typos, grammar or linguistic mistakes impacting the understandability of the texts. The source citations were reduced to a bare minimum to avoid cross-contaminating this thesis's bibliography. Please refer to the master's thesis for a full list of the references used there.

A3.1 Comparison pattern

The comparison pattern was selected to show an example of the simplest safety mechanism that can be documented and reuse as a pattern. This type of safety mechanism patterns can be found as an internal logical component that is used within another safety mechanism. For instance, the *Homogeneous Duplex Redundancy pattern* integrates the *range check* and *comparison* pattern.

Now it will be shown, how this safety mechanism pattern can be documented by applying the documentation process proposed in chapter 3. The *first step* is to record the safety requirements, which gave origin to the development of the safety mechanism. The CASE-- tool AF3 already offers the functionality to document requirements. It also provides the possibility of establishing trace links to other requirements, which are the product of a refinement. In the next figure can be observed how this can be done in AF3. There is information highlighted in red because it is the necessary data needed to advance in the documentation process. The requirement has to be marked as a safety requirement. In the *comparison* pattern this is the requirement that has to be fulfilled by the safety mechanism, and it is described as follows:

- **RI**: Values provided by system interface must be checked against implausible values.

The **R** is used to denote a requirement, and this notation is going to be used in the following pattern examples.

The *second* step in this documentation process consists in adding the related safety case to the pattern. AF3 offers the functionality to construct the safety case. With this inclusion the main elements (goals) of the safety case can now be supported through the existence of concrete evidence (requirements list). Before it is shown how the safety case is added to the safety mechanism pattern, it is important to clarify the abbreviations used within the safety case argument of AF3. **G** is to denote a goal, **S** to the denote strategy, **C** for context, **J** for justification, **A** for assumption, **Sn** for solution. More information can be found in [58]. This notation is going to be considered valid for the rest of the examples shown in this master's thesis.

The screenshot displays a software requirement management interface. At the top, a requirement R1 is highlighted with a red box, titled "Values provided by system interface must be checked against implausible values." Below this, the "General" tab is active, showing fields for ID (R1), Type (Requirement), Title, Description, Rationale, Author (Developer), Is Safety Requirement (checked), Source (Stakeholder), Document Reference, and Status (Identified). The bottom section shows the "New Safety Mechanism Pattern process steps:" with a list of steps from 1 to 6. Step 1, "Add Safety Requirements," is highlighted with a red box and a green checkmark. Step 2 information is also visible, stating: "In this step you have to add at least one safety-related requirement in order to proceed to the next step."

Figure 58: Requirement of comparison pattern

The next figure depicts the corresponding safety mechanism safety case. In it, it can also be observed that the safety case argument is completed and that the user can go on with the documentation process by enabling the option to add logical component and the placeholders of the required architecture.

The elements are described next thoroughly due to readability reasons:

- **G1:** The system must detect erroneous data values.
- **G2:** Implausible values must be detected in interface port.
- **G3:** Values to be checked must be compared against a determined value given by the user by using OP (operator).
- **S1:** Detect erroneous data values by applying a comparison safety mechanism.
- **J1:** Comparison is used to detect inconsistent data values.
- **C1:** Reference value for validation.
- **C2:** Value to be compared from other component.
- **C3:** OP Definition {lt, lte, eq, neq, gte, gt}

- **A1**: it is assumed that an output receiver exists. This assumption is valid within the whole safety case.
- **Sn1**: Comparator (Logical component)
- **Sn2**: Selected operator.

The relationships between the elements are listed next:

- **G1** is supported by **S1**
- **S1** is supported by **G2**, **G3**
- **G2** is supported by **Sn1**
- **G3** is supported by **Sn2**
- **S1** is in context of **J1**
- **S1** is in context of **A1**
- **G2** is in context of **C1**, **C2**
- **G3** is in context of **C3**

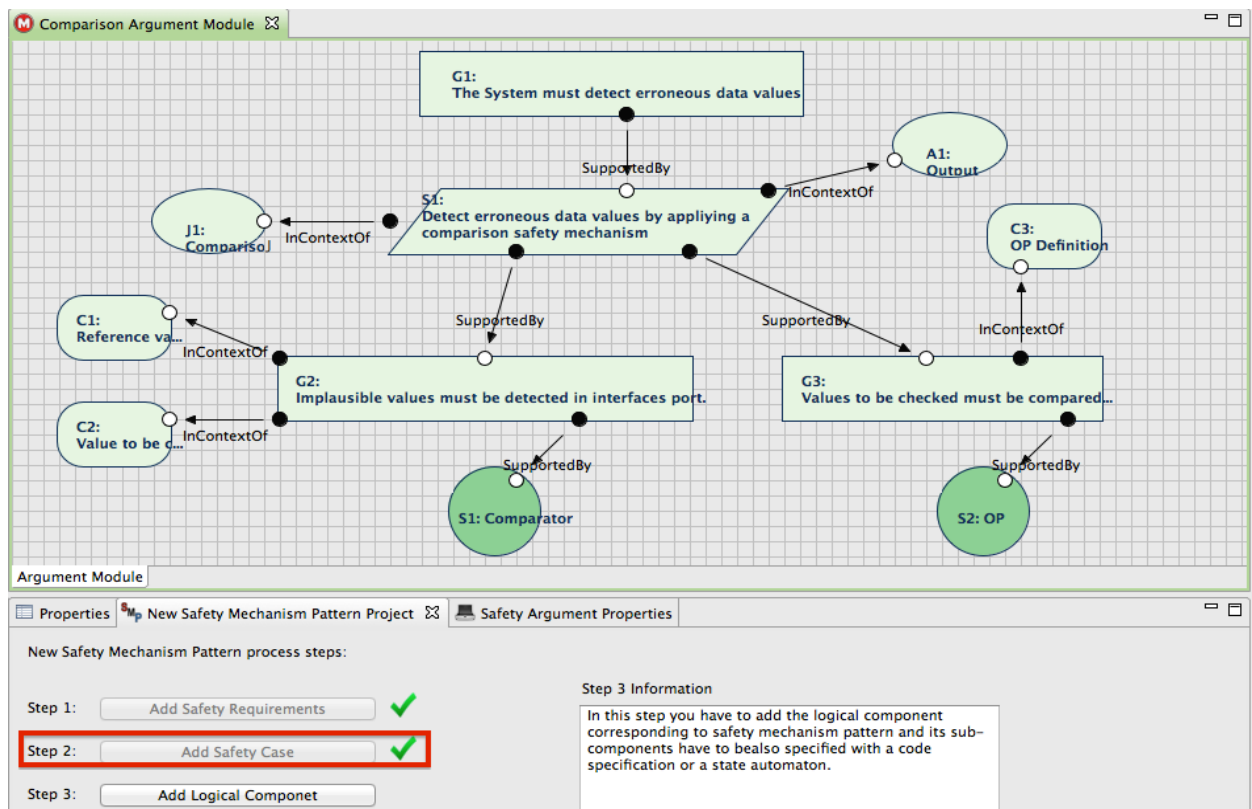


Figure 59: Safety case argument related to comparison pattern

Once the safety mechanism is added the trace links between goals and included requirements from step 1 can be established. In figure 4.3 it can be seen how this is done in AF3. In this case there is only one requirement, therefore the traceability link is:

- $G1 \textcircled{R} R1$

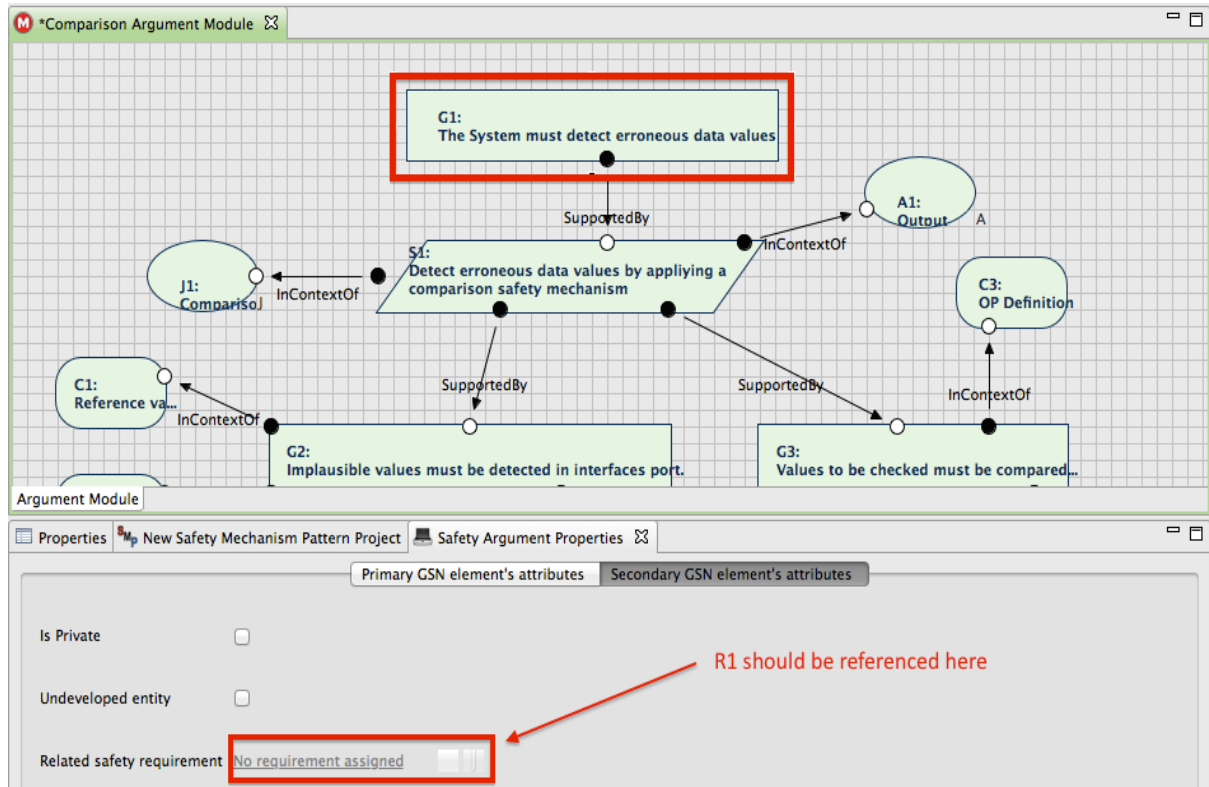


Figure 60: Trace link to safety requirement of comparison pattern

In the figure above, it can be observed that the functionality required to establish the trace link to the requirement is not available, because the option to do it is disabled. Therefore, it is shown “no requirement assigned”.

After the safety goal was successfully documented within the safety mechanism pattern, the process can be continued with the execution of the **third step**, which is the documentation of the safety mechanism pattern specifications. In AF3 there is the possibility to add a *formal specification*, which is the same as configuring/creating the logical component and its architecture, see step 4. Nevertheless, in the following bullet points the specifications of the comparison pattern are described:

- **Spec1:** The comparison of values must be done each time that a value is provided by another module.
- **Spec2:** The user must select the operator for the comparison, belonging to the domain {lt, lte, eq, neq, gte, gt}.
- **Spec3:** The output of the comparator must be a Boolean value, respectively {true, false}.

The next sub--step is to link each specification to its respective requirement, from which the specification is derived; the resulting links are listed as following:

- *Spec1* ® *R1*
- *Spec2* ® *R1*
- *Spec3* ® *R1*

In this case the three specifications are link to the only requirement defined for the comparison pattern. It can be seen that they are described more concretely. They will be the bases used for the implementation of the logical component. However, the work in step 3 is not completely done, each sub--goal of the safety case argument, that matches the characteristics of a specification, must be linked to its respective specification. The resulting links are listed next:

- *G2* ® *Spec1*
- *G3* ® *Spec2*
- *G2* ® *Spec3*: this is a link that cannot be seen clearly in the safety case argument, but it can be implied that after comparing a value by making use of a Boolean operator the result or output of the logical component is going to be also a Boolean.

The next step in the documentation process is the *fourth step*, which consists in constructing the logical component, needed to execute a function to check if a received/provided value is plausible. The other sub--step is to set the placeholders and description for those logical components that are needed by the safety mechanism in order to work correctly. In the next figure, it can be seen that the logical component that comprises the comparison pattern was constructed successfully, because now the user can continue the documentation process. It can also be observed a couple of logical components that belong to the required logical architecture, which are the *InputProvider* and the *OutputReceiver*, which are marked with an *I*, to let the user know that they have *to be instantiated* at the time of reusing the safety mechanism. The functionality that allows marking those logical components as to be instantiated is not available in AF3 yet. For this reason, the next figure is used to show how it could see in the future.

After the building of the logical component, the *Comparator*, it has to be linked to its respective specification in order to keep the traceability property along the safety mechanism pattern as determined in chapter 3. In this case the component is connected to all specifications, because is the only component that was added:

- *Comparator* ® *Spec1*
- *Comparator* ® *Spec2*
- *Comparator* ® *Spec3*

This notation it is also going to be considered valid for the following safety mechanism patterns. The work that is still to be done is to establish a connection between the safety case argument and the logical component, to show how solutions are supported by concrete evidence. This means that the following connections have to be created in order to keep traceability:

- *Sn1* ® *Comparator*
- *Sn2* ® *Comparator*

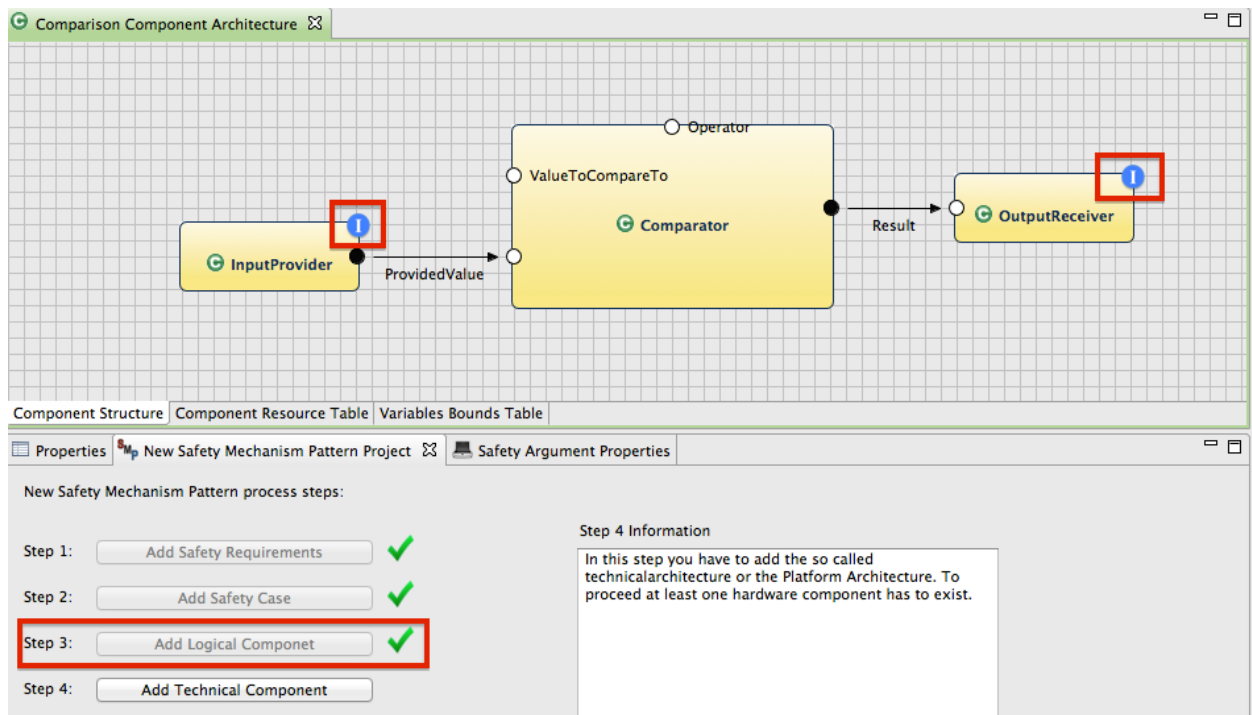


Figure 61: Logical component and surrounding architecture of comparison pattern

Once the relationships between logical component and safety case argument are completed, the developer can decide to define the technical component in which the comparator has to be deployed. The focus of this master's thesis is to show how the safety mechanism pattern must be documented to achieve the minimum level of completeness for its adequate use. Therefore, the documentation process for the comparison pattern and the following patterns will be limited to steps 1 – 4 and 7.

The *fifth step* consists in defining the technical component in which the logical component is going to be deployed, by following the deployment rules defined in the *sixth step*. As stated before, these are optional steps.

An imperative task that must be carried out in the documentation process is the one related to the *seventh step*, which is the recording of the relevant and additional data that cannot be documented through those models created in the previous steps.

In the following figure, it can be observed how this information is recorded in AF3.

Comparison Pattern Data

Overview

▼ **Safety Mechanism Pattern**

Fill out the following additional data regarding the safety mechanism pattern. Fields marked with * must be filled out in order to proceed with pattern documentation process.

Also known as: Comparator, Plausibility Check

Classification*: Failure Detection

Applicability*: It is applicable for data elements (values) that are required or provided by interfaces defined by port prototypes. Its context is comprised by two logical components that could be located in the same technical component or in different technical components. These two logical components are the Input Provider and the Output Receiver, which were mentioned in step number 4. The Input Provider will determine with which type of input the comparator component is going to work. For example, if it is with integers or doubles.

Consequences*: If the comparison pattern is applied, erroneous values coming from the input providers will be detected. On the other hand this could cause that the right operation of the embedded system could be compromised, because it could be interrupted. Another consequences of detecting errors is that they will not be propagated to the following modules or channels integrated in the embedded system. This check could also mean to sacrifice a bit of performance. because extra operations have to be done before continuing with the rest of the tasks involved. as consequence the general

Implications: Not available

Sample:

Known uses: ACC speed plausibility check.

Safety Mechanism Pattern Data

Properties New Safety Mechanism Pattern Project

New Safety Mechanism Pattern process steps:

Step 1: Add Safety Requirements ✓

Step 2: Add Safety Case ✓

Step 3: Add Logical Component ✓

Step 4: Add Technical Component

Step 5: Add Deployment

Step 6: Add Pattern Data ✓ Save SMP

Steps completed!

Now press "Save SMP" to add the safety mechanism pattern to the library.

Figure 62: Additional data to be recorded with the comparison pattern

The stored data from the figure above is presented as following:

- **Also known as:** Comparator, Plausibility Check
- **Classification:** Failure Detection
- **Applicability:** It is applicable for data elements (values) that are required or provided by interfaces defined by port prototypes. Its context is comprised by two logical components that could be located in the same technical component or different technical components. These two logical components are the *Input Provider* and the *Output Receiver*, which were mentioned in step number 4. The Input Provider will determine with which type of input the comparator component is going to work. For example, if it is with integers or doubles.
- **Consequences:** If the comparison pattern is applied, erroneous values coming from the input providers will be detected. On the other hand, this could cause that the right operation of the embedded system could be compromised, because it could be interrupted. Another consequence of detecting errors is that they will not be propagated to the following modules or channels integrated in the embedded system. This check could also mean to sacrifice a bit of performance

because extra operations have to be done before continuing with the rest of the tasks involved, as a consequence the general execution time of the function will increase.

- **Implications:** not available due to a lack of information to carry out the corresponding analysis. This attribute was also denoted as optional. To see an example of how the information should be registered within this attribute, see *Homogeneous Duplex Redundancy pattern* (section 4.1.4).
- **Sample:** see Figure 61.
- **Known uses:** *Adaptive Cruise Control (ACC) speed plausibility check.*

A3.2 Range Check pattern

The screenshot displays a software requirement management interface. At the top, two requirement IDs are visible: R1 and R2. The main form is titled 'Requirement' and contains the following fields:

- ID:** R1
- Type:** Requirement
- Title:** Data values must be checked against a valid range of permitted values.
- Description:** Data values must be within a specified interval, otherwise the data is considered as erroneous.
- Rationale:** Detect erroneous data values that could compromise the safe operation of the embedded system.
- Author:** Developer
- Is Safety Requirement:** ☒
- Source:** Stakeholder
- Document Reference:**
- Status:** In Analysis

Below the form is a 'Checklist' section titled 'New Safety Mechanism Pattern process steps:'. It contains six steps:

- Step 1: Add Safety Requirements (checked with a green checkmark)
- Step 2: Add Safety Case
- Step 3: Add Logical Component
- Step 4: Add Technical Component
- Step 5: Add Deployment
- Step 6: Add Pattern Data

To the right of the checklist is a 'Step 2 Information' box with the text: 'In this step you have to add at least one safety-related requirement in order to proceed to the next step.'

Figure 63: Requirements of range check pattern

This pattern is shown as a variation of the *one component* safety mechanism pattern because some of the inputs, that the logical component uses, are optional. This permits users of the pattern to have a certain level of flexibility when incorporating this safety

mechanism in an embedded system. It can be considered as an alternative to the *comparison* pattern.

The documentation process of the *range check pattern* begins with the **first step**, which is the recording of the requirements related to the development of the safety mechanism. In the figure above, the documented requirements in AF3 can be observed.

The requirements are also listed as follows:

- **R1**: Data values must be checked against a valid range of permitted values.
- **R2**: The specified interval to catch erroneous data values can be extended by adding a tolerance. This requirement allows making more flexible the value of the data around the interval limits.

Once the requirements are successfully added to the pattern, the documentation process can continue with the execution of the **second step**, which consists in the modeling of the safety case argument that will be supported through the development artifacts of the safety mechanism pattern. The recorded safety case for this pattern can be observed in the next figure.

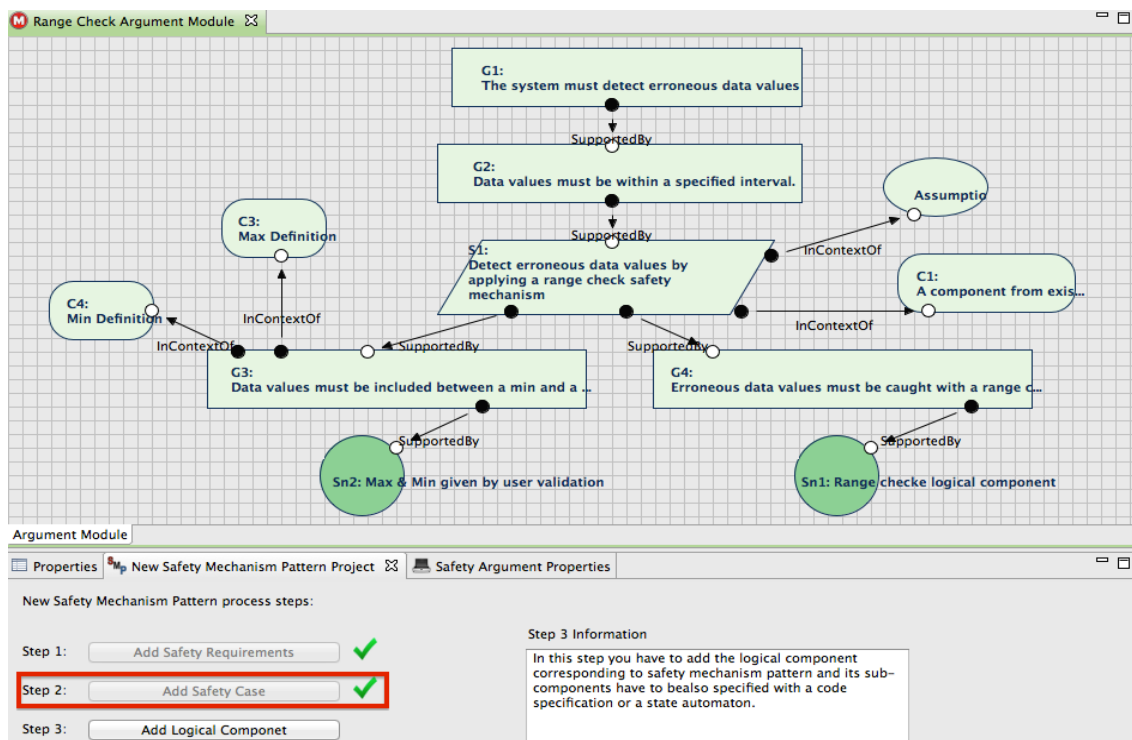


Figure 64: Safety case argument related to range check pattern

The elements of the safety case are presented thoroughly as follows:

- **G1**: The system must detect erroneous data values. A goal is normally used to introduce the idea of failure detection as a property that the system has to fulfill, like in the case of the comparison pattern.

- **G2**: This goal refines **G1**, because it states a more detailed of which failure detection modality is going to be applied.
- **G3**: Data values must be included within a specified interval.
- **G4**: Erroneous data values must be caught by the range check function.
- **S1**: Detect erroneous data values by applying a range check safety mechanism.
- **C1**: A component provides the value to be validated.
- **A1**: It is assumed that some logical component should exist to receive the result of the plausibility check.
- **C3**: Definition of the max value.
- **C4**: Definition of the min value.
- **Sn1**: Range check function (logical component).
- **Sn2**: Validation of the max and min values given by the user. The relationships between the elements are listed next:
- **G1** is supported by **G2**
- **G2** is supported by **S1**
- **S1** is supported by **G3, G4**
- **G3** is supported by **Sn2**
- **G4** is supported by **Sn1**
- **S1** is in context of **C1, C2, A1**
- **G3** is in context of **C3, C4**

Once the safety case documentation is completed, trace links must be established between the goals and the requirements of the safety mechanism, in order to support part of the safety case with concrete evidence (List of requirements). The links are listed as follows:

- **G1 ® R1**
- **G2 ® R2**

After the first traceability between the safety case and the development artifacts were created the documentation process can go on with the **third step**, which is the recording of those specifications derived from the requirements presented before. Even though there is not functionality that allows documenting the specification in AF3, this information has to be stored somewhere because it is the detailed representation of the safety mechanism needed to implement the logical component. The specifications of this pattern are:

- **Spec1**: Data values are checked against a valid range of values, where min is the lower accepted value and max the upper limit of the interval.
- **Spec2**: the user of the pattern gives Range values, which have to be subject to a validation of the interval limits. {max > min}
- **Spec3**: the tolerance given by the user of the pattern has to be added to the range check as following: check *provided value* (pv) in range $pv \geq min - (min * t)$ and $pv \leq max + (max * t)$, where t is the tolerance.
- **Spec4**: the result of applying the range check is delivered as a Boolean value, respectively {true, false}

A sub--step that must be carried out, is the linking of these specifications to their respective requirement, which is done as following:

- **Spec1** ® **R1**
- **Spec2** ® **R1**
- **Spec3** ® **R2**
- **Spec4** ® **R1**

Once this level of traceability is reached, the developer can proceed to the next sub--step, which is the connection of the sub--goals of the safety case with the corresponding specifications of the safety mechanism pattern. The resulting trace links are:

- **G4** ® **Spec1**
- **G4** ® **Spec2**
- **G3** ® **Spec3**
- **G4** ® **Spec4**

Having documented the specification as part of the pattern, it is time to proceed to the next step, which is the **fourth step**. It consists in constructing or in this case modeling the logical component that is going to be integrated in the embedded system and the required surrounding architecture, in order to guarantee its correct functioning. In the next figure, it can be observed the result of executing this activity in AF3.

There are elements modeled in the component architecture shown in the next figure that are marked with an **I**. These logical components should be instantiated through existing components of the embedded system, when the augmentation reuse process is applied. Otherwise, they must be developed by the user if the greenfield reuse process is chosen.

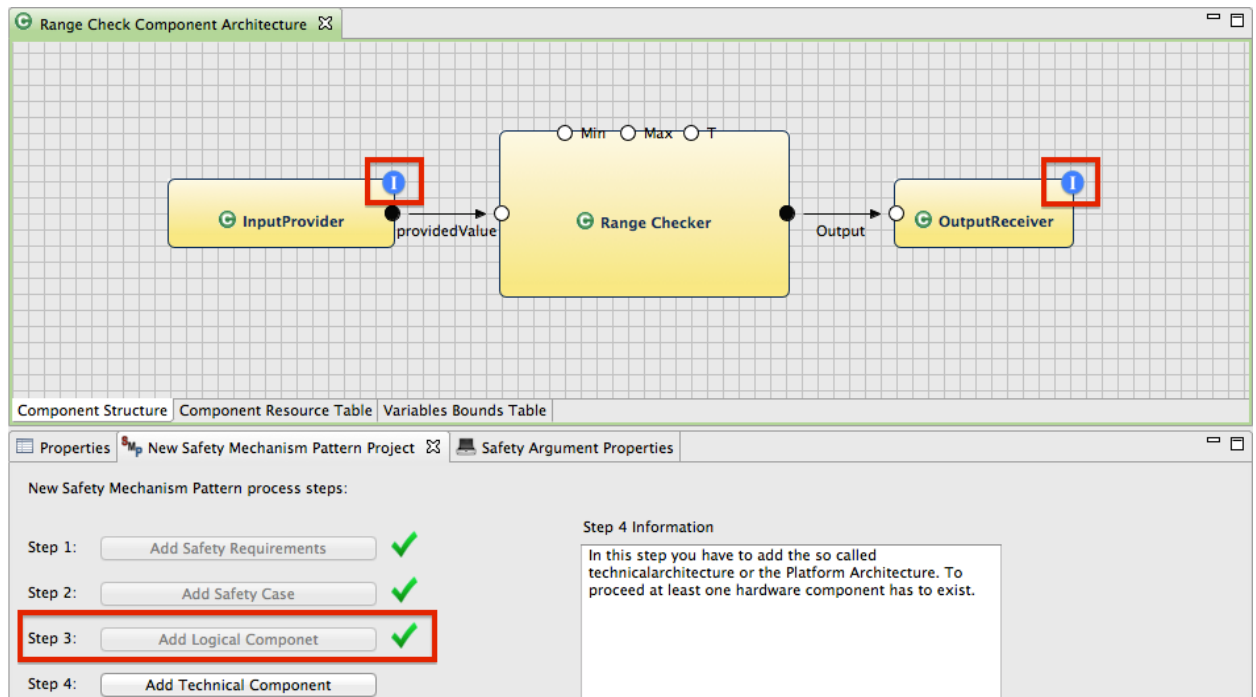


Figure 65: Logical component and surrounding architecture of range check pattern

There are a couple of sub-steps that must be executed while documenting the logical component. First, trace links must be created between the logical component and the specification that is being addressed by the safety mechanism, as follows:

- ***Range Checker* ® *Spec1***
- ***Range Checker* ® *Spec2***
- ***Range Checker* ® *Spec3***
- ***Range Checker* ® *Spec4***

In this way when a failure occurs, it can be traced back to one of the specifications and from there up to the safety requirement. The other sub-step consists in establishing traceability between the safety case and the pattern, in which the solutions are supported by the logical components. The resulting links are:

- ***Sn1* ® *Range Checker***
- ***Sn2* ® *Range Checker***

In the next figure, it is shown how this trace link can be established with the functionality offered by AF3. However, this functionality is not available in AF3 yet, because the option to set the trace link is disabled, as it can be seen in the figure. The so-called *internal evidence* should reference the logical component (Range Checker) added to the component architecture of the *range check pattern*.

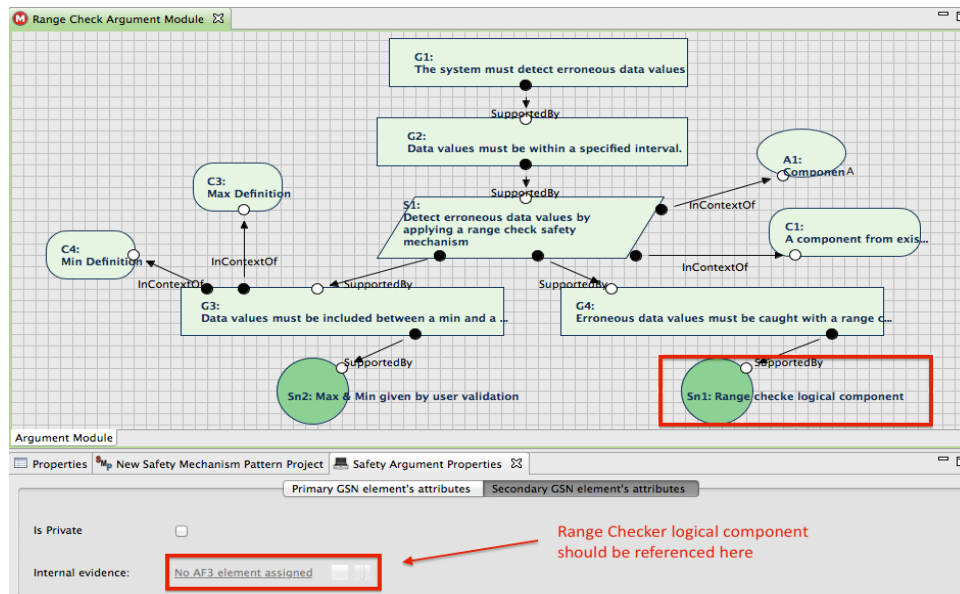


Figure 66: Trace link to logical component of range check pattern

As it was stated before, this master's thesis is focused on showing how developers can achieve the minimum level of completeness at the moment of documenting a safety mechanism pattern. Therefore, the *fifth* and *sixth steps* are not carried out for the range check pattern. However, it is important to emphasize that these steps are about defining the technical component (hardware) in which the logical component defined in step 4 is going to be deployed.

Figure 67: Additional data to be recorded with the range check pattern

The last step of the documentation process is the *seventh step*, which consists in filling out a set of additional information, which is going to be useful at the time of using the pattern. This information can also be documented in AF3, as shown in the figure above.

The content of each pattern attribute is described as follows:

- ***Also known as***: Plausibility Check
- ***Classification***: Failure Detection
- ***Applicability***: It is applicable for data elements (values) that are required or provided by interfaces defined by port prototypes. Its context is comprised by two logical components that could be located in the same technical component or different technical components. These two logical components are the *Input Provider* and the *Output Receiver*, which were mentioned in step number 4. The Input Provider will determine with which type of input the range check component is going to work and validate. For example, if it is with integers or doubles.
- ***Consequences***: If the range check pattern is applied, erroneous values coming from the input providers will be detected. On the other hand, this could cause that the right operation of the embedded system could be compromised, because it could be interrupted. Another consequence of detecting errors is that they will not be propagated to the following modules or channels integrated in the embedded system. This check could also mean to sacrifice a bit of performance because extra operations have to be done before continuing with the rest of the tasks involved, as a consequence the general execution time of the function will increase.

The idea of having a tolerance in this safety mechanism is to offer flexibility when values are very closed to the limits of the intervals, because a value near those limits could not represent such a significant risk for the further operation of the embedded system.

- ***Implications***: not available due to a lack of information to carry out the corresponding analysis. This attribute was also denoted as optional. To see an example of how the information should be registered within this attribute, see *Homogeneous Duplex Redundancy pattern* (section 4.1.4).
- ***Sample***: see figure 4.8
- ***Known uses***: ACC speed plausibility check, ACC distance plausibility check. In the ACC embedded system a speed plausibility check is done, it consists in verifying if the current measured speed remains under a certain limit compared to old measurement. After the comparison, the speed value is delivered or not to the component in charge of regulating the speed of the vehicle.

A3.3 1-out-of-2-Voting pattern

The *1-out-of-2-Voting* (*1oo2--Voting*) can be denoted as a *composition of components* safety mechanism pattern. This means that the documented logical component of the safety mechanism pattern is composed of several logical components, which are going to be presented in step 4 of the documentation process. This safety mechanism is usually used as a deciding component when redundancy is applied in an embedded system.

The screenshot displays a software requirement management interface. At the top, two requirement tabs are visible: 'R1 - The propagation of erroneous values must be prevented when a red...' and 'R2 - A voting function must detect the erroneous data values and forward...'. The 'Requirement' tab is active, showing details for 'R2'. The 'General' section includes fields for ID (R2), Type (Requirement), Title (A voting function must detect the erroneous data values and forward the ones that fulfil a certain quality criteria), Description, Rationale (Discard invalid values that could compromise the correct functioning of the embedded system.), Author (Developer), Source (Stakeholder), and Status (Identified). A checkbox labeled 'Is Safety Requirement' is checked. Below the requirement details, a 'New Safety Mechanism Pattern process steps' section shows a list of steps. Step 1, 'Add Safety Requirements', is highlighted with a green checkmark, indicating it is the current step. Step 2, 'Add Safety Case', is the next step. The 'Step 2 Information' box on the right states: 'In this step you have to add at least one safety-related requirement in order to proceed to the next step.'

Figure 68: Requirements of *1oo2-∇-Voting* pattern

The **first step** consists in recording those safety requirements involved in the development of the safety mechanism, as shown in the figure above, where one of the requirements of the *1oo2--Voting* pattern is shown documented in AF3.

These recorded requirements are:

- **R1:** The propagation of erroneous values must be prevented when a redundant channel is being used.
- **R2:** A voting function must detect the erroneous data values and forward the ones that fulfill a certain quality criteria.

As it can be observed in the next figure, the button to add the safety requirement is already disabled, because at least one requirement was added successfully. For this reason the button to add the safety case was set as enabled to proceed with the next step of the process.

The **second step** comprises the inclusion of the related safety case to the pattern, in order to be able to back up the history behind the argumentation chain of the safety case. The safety case structure related to the *1oo2--Voting pattern* can be observed in the next figure.

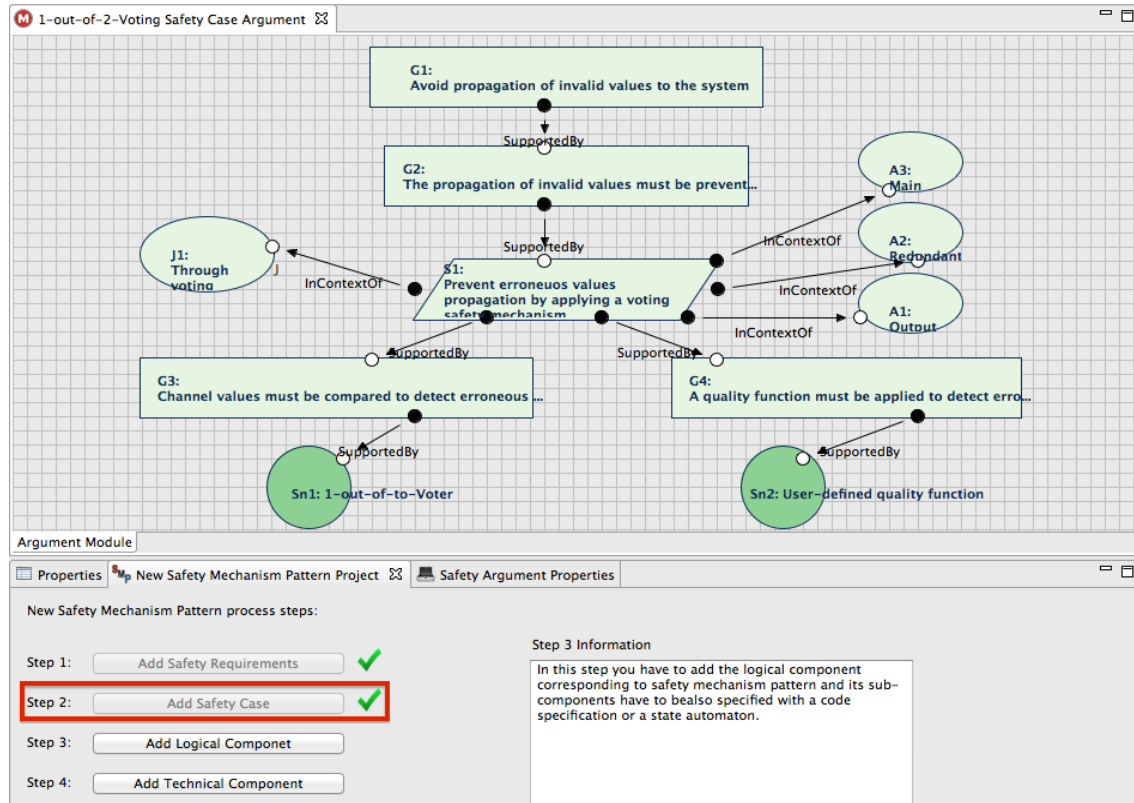


Figure 69: Safety case argument of 1oo2-Voting pattern

The elements of the safety case argument presented in the figure are described as following for a better readability:

- **G1:** Avoid propagation of invalid values to the system.
- **G2:** The propagation of invalid values must be prevented when using a redundant channel.
- **G3:** Channel values must be compared to detect erroneous data.
- **G4:** A quality function must be applied to detect erroneous values.
- **S1:** Prevent erroneous values propagation by applying a voting safety mechanism.
- **J1:** Through voting mechanism erroneous values are detected, and backup values can be forwarded, otherwise it serves as failure detection safety mechanism.
- **A1:** Main channel, which is a component that is normally providing the system e.g. with processed values, or values from a sensor, etc.

- **A2**: Redundant channel, which is the back up running system, which also processes data, but is not the one that is normally used, but stays as backup.
- **A3**: It is assumed that some logical component should exist to receive the result of the voting.
- **Sn1**: 1-out-of-2-Voter
- **Sn2**: Quality function defined by the user to be utilized within the voter.

The relationship between the elements are listed next:

- **G1** is supported by **G2**
- **G2** is supported by **S1**
- **S1** is supported by **G3**
- **S1** is supported by **G4**
- **S1** is in context of **J1, A1, A2, A3**
- **G3** is supported by **Sn1**
- **G4** is supported by **Sn2**

After laying out the safety case, trace links must be established between the goals of the safety case and the requirements of the safety mechanism pattern:

- **G1** ® **R1**
- **G2** ® **R2**

The **third step** of the 1oo2-Voting pattern documentation process is to record those specifications that give a more concrete and exact description of the pattern, which are used as the basis for the implementation of the logical component. They are listed as follows:

- **Spec1**: The propagation of invalid data values, originated in the main channel/component, must be prevented by replacing them through the values computed by a redundant channel/component, in which a 1-out-of-2-Voter component is added.
- **Spec2**: The voter must prove the quality of the values with an user-defined function.
- **Spec3**: The quality computed for each of the values must be under a given tolerance.

After recording the specifications, they have to be linked to their respective requirement due to traceability reasons, and the resulting connections of this pattern are:

- *Spec1* ® *R1*
- *Spec2* ® *R2*
- *Spec3* ® *R1*

In the same way this specification must be linked to sub-goals of the safety case, as shown next:

- *G3* ® *Spec1*
- *G4* ® *Spec2*
- *G3* ® *Spec3*: even though the tolerance is not mentioned explicit in *G3*, it is part of the voter functionality.

Going on in the process it can be proceed to document the logical component and its surrounding architecture, which is the *fourth step*. In the next figure it can be observed the voter component and the required surrounding architecture.

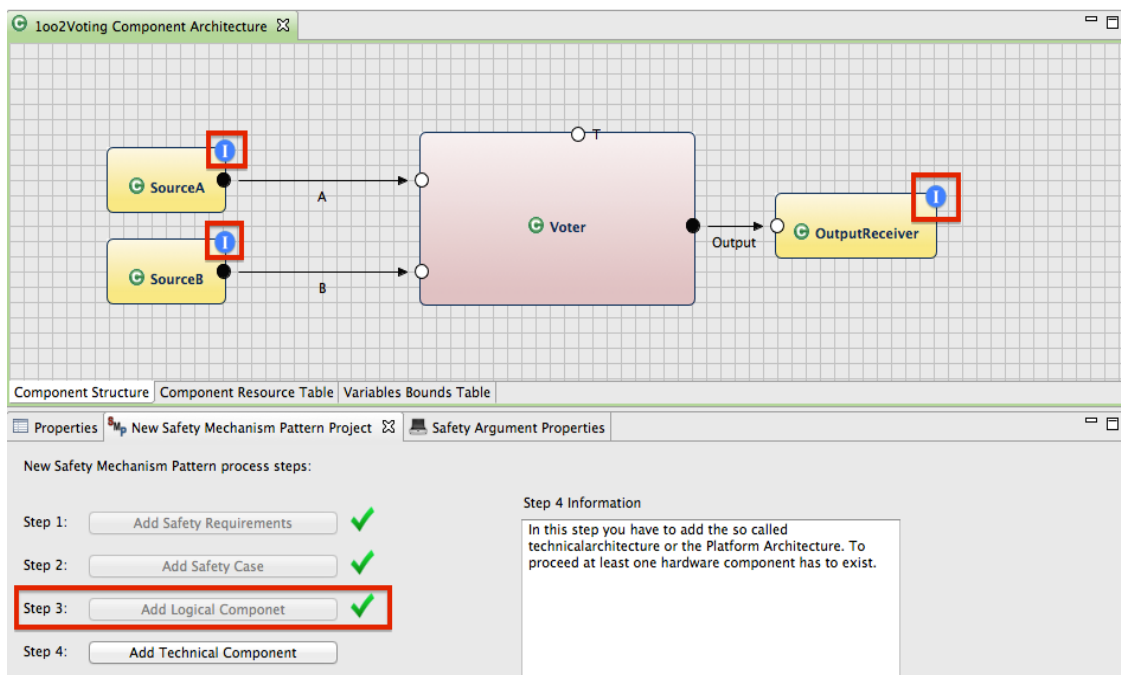


Figure 70: Logical component and surrounding architecture of 1oo2-Voting pattern

The architectural components required by the 1oo2-Voting pattern are marked with *I* in the figure, in order to let know the user of this pattern that it needs to be instantiated when e.g. the augmentation reuse process is applied.

At the beginning of this section it was stated that this pattern was different in a logical level, because it is composed internally by other logical components, which are shown in the next figure. In it, the letter *D* is used to denote those components that need *to be developed* by the user when the pattern is used. The functionality that allows marking those logical components as to be developed is not available in AF3 yet. They are nevertheless depicted in the figure, to show how it could see in the future. The *voter* component in this case is the one who have to check the computed qualities against a given tolerance, in order to decide which output is fit to be sent as output.

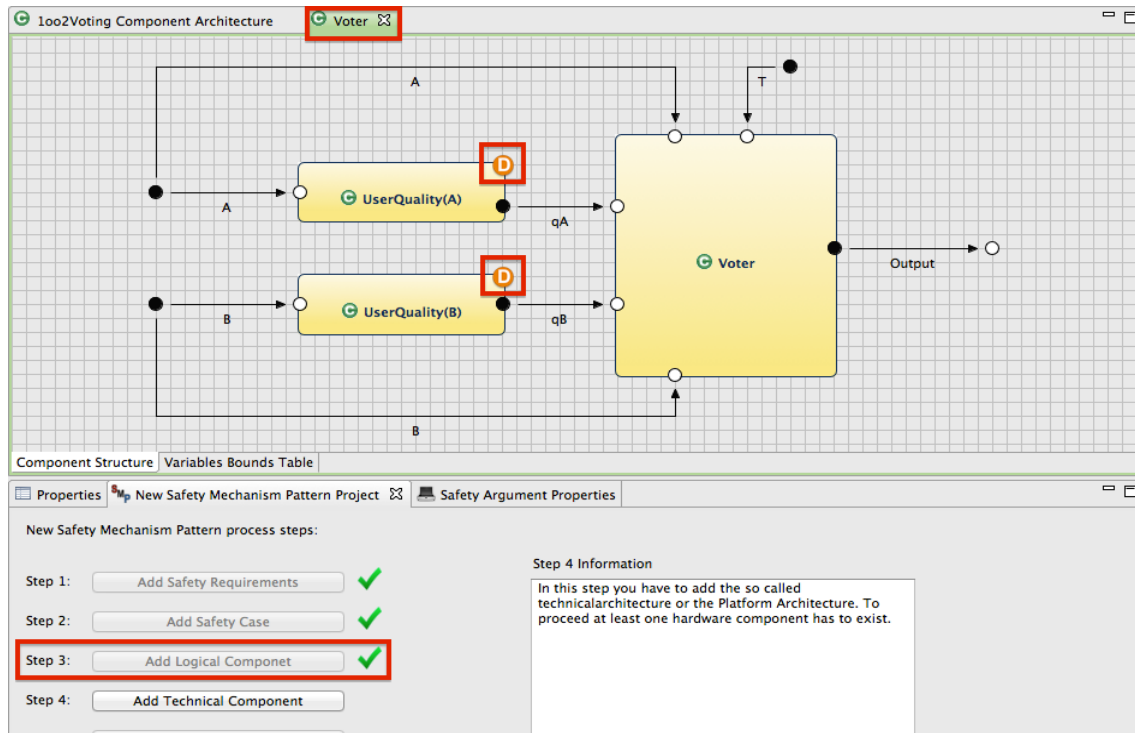


Figure 71: Internal view of *Voter* component

After the logical component of the pattern was defined, it has to be linked to its respective specifications due to traceability reasons, as follows:

- *Voter* ® *Spec1, Spec3*
- *UserQuality(A)* ® *Spec2*
- *UserQuality(B)* ® *Spec2*

These components must be also linked to the respective elements of the related safety mechanism, in order to support the solutions through concrete evidence. The resulting trace links are listed as follows:

- *Sn1* ® *Voter*
- *Sn2* ® *UserQuality(A)*
- *Sn2* ® *UserQuality(B)*

It is important not to forget that the surrounding logical components, which are required by the safety mechanism in order to work correctly, must also be linked to its corresponding element in the safety case. In this case, the context elements have to be linked to those architectural components:

- *A1* ® *SourceA*
- *A2* ® *SourceB*
- *A3* ® *OutputReceiver*

The *fifth* and *sixth* steps are optional, which consist in documenting the information regarding the technical component, in which the safety mechanism is going to be deployed, in order to achieve a more complete level of documentation.

To finish the documentation of the 1oo2--Voting pattern, a *seventh* step has to be carried out, which comprises the recording of additional data, which is going to be considered useful at the moment of reusing the safety mechanism. In the next figure, the documentation of this information in AF3 can be observed.

The image shows two screenshots from a software interface. The top screenshot is titled "1oo2Voting Pattern Data" and shows the "Overview" tab. It contains a form for documenting a safety mechanism pattern. The form has several fields, some of which are highlighted with red boxes: "Classification:*" (set to "Failure Handling"), "Applicability:*" (containing text about sensor inputs), "Consequences:*" (containing text about error messages and hardware failure), and "Implications:" (set to "Not available"). The bottom screenshot is titled "New Safety Mechanism Pattern Project" and shows a list of steps. Steps 1 through 5 are completed, indicated by green checkmarks. Step 6, "Add Pattern Data", is also highlighted with a red box and a green checkmark. A "Save SMP" button is also highlighted with a red box. A message box on the right says "Steps completed! Now press 'save SMP' to add the safety mechanism pattern to the library."

1oo2Voting Pattern Data

Overview

Safety Mechanism Pattern

Fill out the following additional data regarding the safety mechanism pattern. Fields marked with * must be filled out in order to proceed with pattern documentation process.

Also known as: Voter

Classification:* Failure Handling

Applicability:* This safety mechanism requires two distinct inputs in order to operate. If they result e.g. to be sensors, the logical component of the sensor should be included in the component architecture. Voting could also happen between values computed locally and values computed by another electronic control unit (ECU).

Consequences:* If the computed quality for each of the provided inputs does not stay under a given tolerance the voter will send an error message to the output receiver, which can cause a malfunctioning of the embedded system or an interruption of its service. But normally it is very difficult that this case occur, because the same random hardware failure, e.g. erroneous measurement of a sensor, does not occur normally at the same time. On the other hand, the performance of the embedded system is affected negatively, so that it consumes more time and resources to execute its functionality.

Implications: Not available

Sample: See component architecture

Known uses: Voting patterns, like the 1oo2-Voting are used inside many other safety mechanism patterns, e.g. N-Version Programming, which in this case will be

Safety Mechanism Pattern Data

Properties New Safety Mechanism Pattern Project

New Safety Mechanism Pattern process steps:

Step 1: Add Safety Requirements ✓

Step 2: Add Safety Case ✓

Step 3: Add Logical Component ✓

Step 4: Add Technical Component

Step 5: Add Deployment

Step 6: Add Pattern Data ✓

Save SMP

Steps completed!

Now press "save SMP" to add the safety mechanism pattern to the library.

Figure 72: Additional data to be recorded with the 1oo2--Voting pattern

The content of this pattern attributes from the figure above is shown completed as following:

- **Also known as:** Voter
- **Classification:** Failure handling
- **Applicability:** This safety mechanism requires two distinct inputs in order to operate. If they are e.g. to be sensors, the logical component of the sensor should be included in the component architecture. Voting could also happen between values computed locally and values computed by another ECU.
- **Consequences:** If the computed quality for each of the provided inputs does not stay under a given tolerance, the voter will send an error message to the output receiver, which can cause a malfunctioning of the embedded system or an interruption of its service. But normally it is very difficult that this case occurs, because the same random hardware failure, e.g. erroneous measurement of a sensor, does not occur generally at the same time. On the other hand, the performance of the embedded system is affected negatively, so that it consumes more time and resources to execute its functionality. Nevertheless, this is something that users have to know and take into account if they want to achieve a certain level of safety within the embedded system.

The voting component inside the logical component could be also considered as a single point of failure, because if it malfunctions, then voting cannot occur and therefore the safety of the embedded systems cannot be guaranteed.

- **Implications:** not available due to a lack of information to carry out the corresponding analysis. This attribute was also denoted as optional. To see an example of how the information should be registered within this attribute, see *Homogeneous Duplex Redundancy pattern*.
- **Sample:** see Figure 70 and Figure 71.
- **Known uses:** Voting patterns, like the 1oo2--Voting are used inside many other safety mechanism patterns, e.g. N--Version Programming, which in this case will be 2-- Version Programming.

A3.4 Homogeneous Duplex Redundancy pattern

This pattern was selected to show how redundancy is documented as a safety mechanism pattern. Moreover, this example will serve to present how the greenfield reuse process is done with the assistance of AF3, because the *Homogeneous Duplex Redundancy* (HDR) pattern makes use of a couple of safety mechanism patterns within its logical architecture.

As it was shown in the previous examples, the documentation of the pattern commences with the *first step*, which consists in recording the requirements, respectively the safety requirements. How they can be documented within AF3 is shown in the next figure.

The requirements of this pattern are also listed next for a better readability:

- **R1:** The system must count with a redundant component that guarantees the operation of the system when a failure occurs.
- **R2:** The processed input and output values provided by the interfaces must be checked against implausible values.
- **R3:** The selection regarding which component/channel must be carried out by a switch/deciding component.

The screenshot displays two windows from the AF3 software. The top window, titled 'Requirement (No comment)', shows the configuration for a requirement with ID 'R1'. The 'General' tab is active, showing fields for ID, Type (Requirement), Title (Redundant component), Description (They system must count with a redundant component that guarantees the operation of the system when a failure occurs.), Rationale (A component is duplicated to offer an alternative, in case that a failure or error is detected in the main component. In this way the risks of having a single point of failure are mitigated.), Author (Developer), Is Safety Requirement (checked), Source (Stakeholder), Document Reference, and Status (Identified). The bottom window, titled 'New Safety Mechanism Pattern Project', shows a checklist of steps. Step 1, 'Add Safety Requirements', is highlighted with a green checkmark, indicating it is completed. Other steps include 'Add Safety Case', 'Add Logical Component', 'Add Technical Component', 'Add Deployment', and 'Add Pattern Data'. A 'Step 2 Information' box on the right states: 'In this step you have to add at least one safety-related requirement in order to proceed to the next step.'

Figure 73: Requirements of the homogeneous duplex redundancy pattern

Due to the nature of this pattern, it is important to emphasize that requirements of the reused safety mechanisms, for example, range check pattern, must be included to the set of requirements as well. The developer in charge of documenting the pattern must analyze the list of imported requirements to avoid their repetition.

The **second step** in the documentation process consists in recording the safety case that is supported through the development artifacts, such as requirements, of the safety mechanism pattern. The safety case argument is depicted in the next figure.

To make the elements from the figure above and their description clear to the reader, they are listed next:

- **G1**: The system must guarantee the further functioning when a failure occurs.
- **G2**: A component or data processing channel must be duplicated.

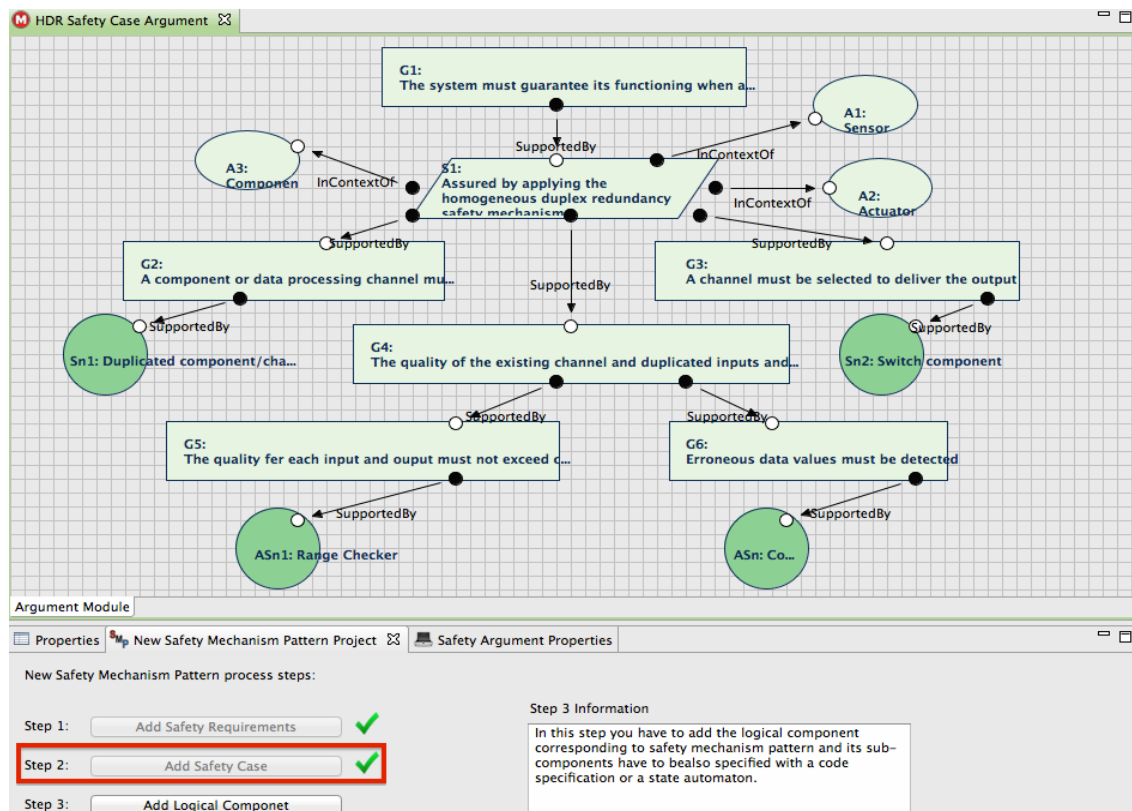


Figure 74: Safety case argument of homogeneous duplex redundancy pattern

- **G3**: A channel must be selected between the main channel/component and the redundant one to deliver an output free from errors.
- **G4**: The quality of the existing channel and duplicated inputs and outputs must be subjected to a plausibility check.
- **G5**: The quality for each input and output must not exceed certain value limits.
- **G6**: Erroneous data values must be detected.
- **S1**: Assured by applying the homogeneous duplex redundancy safety mechanism. In other words, a second redundant component must be included to

handle the occurrence of an error and secure the further working of the embedded system.

- *A1*: Component to be duplicated. It is a component that processes an input and delivers some output.
- *A2*: Sensor that provides the input to the component/channel to be duplicated.
- *A3*: Actuator that becomes the output processed by the component to be duplicated.
- *Sn1*: Duplicated component. It is an exact implementation of the component to which the pattern is going to be applied.
- *Sn2*: Switch component that is responsible for deciding which output is going to be used.
- *ASn1*: Range checker, is a safety mechanism pattern used to carried out the evaluation of the processed input and output of both components/channels.
- *ASn2*: Comparator, which is also a technical solution reused to detect erroneous data values.

Compared to the safety mechanism patterns presented before, it can be seen that the solution range checker and comparator are *away solutions* referenced from the respective safety case arguments. Therefore, they have to be also included as safety case arguments within the safety case of the homogeneous duplex redundancy pattern. Away solutions are explained well in [58].

The relationships between the elements are listed next:

- *G1* is supported by *S1*
- *S1* is supported by *G2*
- *S1* is supported by *G3*
- *S1* is supported by *G4*
- *G4* is supported by *G5*
- *G4* is supported by *G6*
- *G2* is supported by *Sn1*
- *G3* is supported by *Sn2*
- *G5* is supported by *ASn1*
- *G6* is supported by *ASn2*
- *S1* is in context of *A1, A2, A3*

After documenting the safety case related to the pattern, trace links between the goals and the requirements from step 1 must be established, as it is shown next:

- ***G1* ® *R1***
- ***G4* ® *R2***
- ***G4* ® *R3***

Once the safety cases were included the process can continue to the execution of the ***third step***, which is the recording of the specifications:

- ***Spec1***: An existing component in the embedded system architecture must be duplicated.
- ***Spec2***: the quality of the processed inputs and computed outputs of the respective component (main and redundant) must be checked by using a range checker. The user must give other inputs for the range checker.
- ***Spec3***: The outputs of the corresponding range checker must be compared to observe if the data processing for each component is done faultless. The comparator must receive four inputs, two processed inputs and two computed outputs.
- ***Spec4***: The result obtained from the external comparator, which contained the range checker and the comparator, serves as input for deciding component (switch), which determines with which channel/component the embedded system is going to fulfill their tasks to.
- ***Spec5***: In case of obtaining a negative evaluation of the value qualities from both channels/components, it must be marked as an error.

Due to traceability reasons a sub--step has to be carried, which consists in building trace links between those specifications and their corresponding requirements, like it is shown next:

- ***Spec1* ® *R1***
- ***Spec2* ® *R2***
- ***Spec3* ® *R2***
- ***Spec4* ® *R3***
- ***Spec5* ® *R2, R3***

In the same way these specifications must be connected to their counterpart in the safety case argument, which are those sub--goals resulted from a goal splitting or the employment of a strategy. These trace links are:

- ***G2* ® *Spec1***

- *G5* ® *Spec2*
- *G6* ® *Spec3*
- *G3* ® *Spec4*
- *G3, G6* ® *Spec5*

The specifications related to the used patterns have to be also added to the set of specifications.

After it is done, the progress of the documentation process can go on, in which the **fourth step** is executed. The result of documenting the logical component and the required architecture of the safety mechanism are depicted in the next figure.

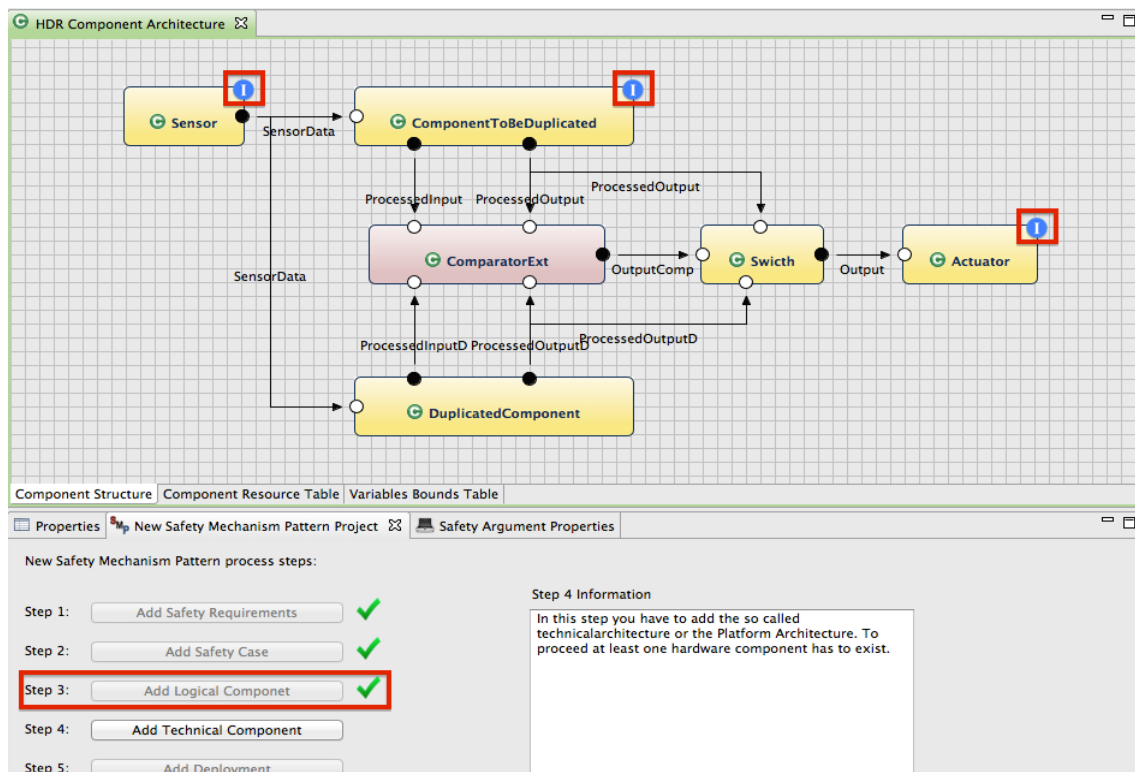


Figure 75: Logical components and surrounding architecture of the HDR pattern

The elements marked with an **I** in the figure above, are those who need to be instantiated by existing logical components of the embedded system, in which the safety mechanism is going to be integrated. This is in case that the augmentation reuse process is applied. There is a component, called *ComparatorExt*, which contains the respective range checkers for each processed input and each computed outputs. Then a comparator is used to detect erroneous data values. In the next figure, it can be seen how this *ComparatorExt* is internally modeled at a logical level.

Once the logical component and the required architecture are modeled and recorded, each of the elements has to be linked to its respective specification as follows:

- *DuplicatedComponent* ® *Spec1*
- *ComparatorExt* ® *Spec2, Spec3, Spec5*
- *Switch* ® *Spec4*

The logical components documented with the HDR pattern must be linked to the solutions of the safety case due to traceability reasons, as shown next:

- *Sn1* ® *DuplicatedComponent*
- *Sn2* ® *Switch*
- *ASn1* ® *Range Checker*
- *ASn2* ® *Comparator*

The required logical components modeled in this pattern must be also linked to its respective element from the related safety case:

- *A2* ® *Sensor*
- *A3* ® *Actuator*
- *A1* ® *ComponentToBeDuplicated*

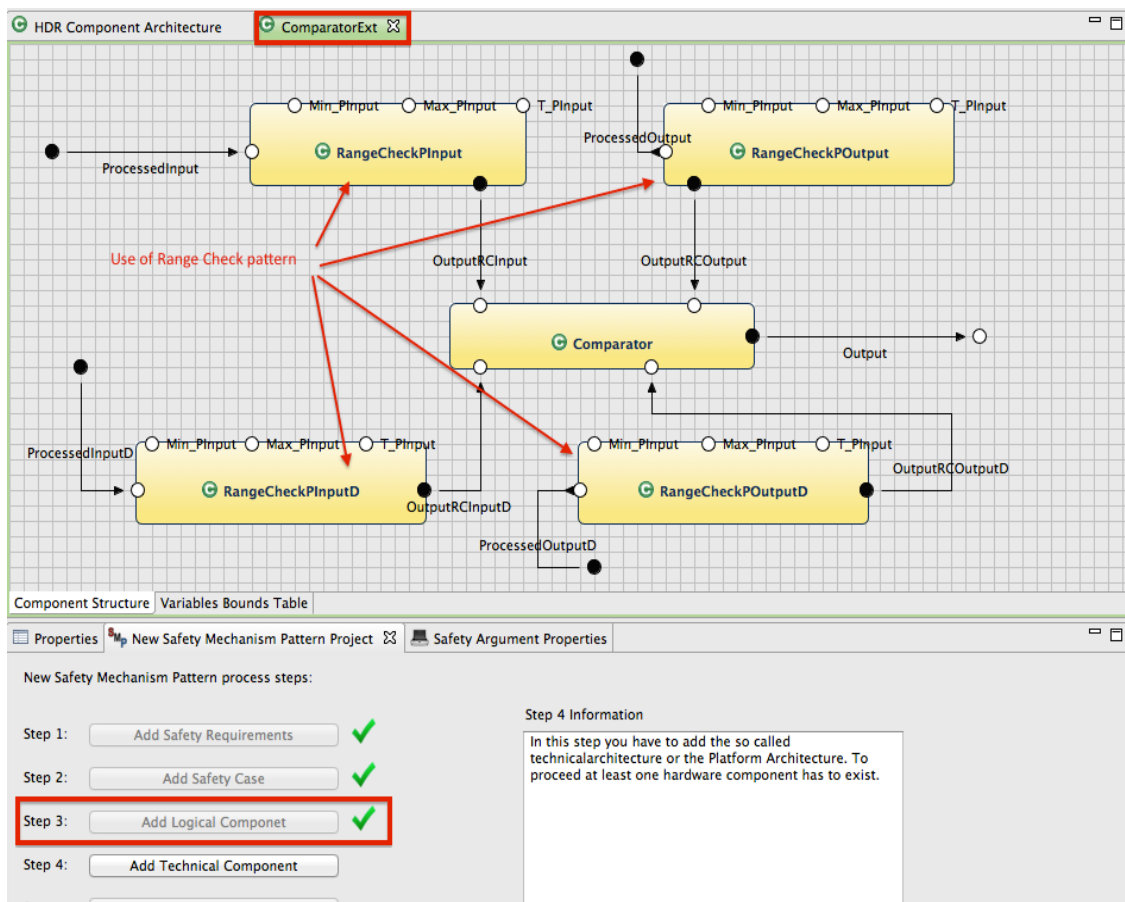


Figure 76: Internal view of the ComparatorExt logical component

The *fifth* and *sixth* steps are skipped as it was done in the previous examples of safety mechanism patterns.

The image shows two screenshots from a software application. The top screenshot is the 'HDR Pattern Data' window, specifically the 'Overview' tab. It contains a form for recording data about a 'Safety Mechanism Pattern'. The form includes fields for 'Also known as', 'Classification', 'Applicability', 'Consequences', 'Implications', 'Sample', and 'Known uses'. The 'Classification' field is set to 'Failure Detection & Failure Handling'. The 'Applicability' and 'Consequences' fields are highlighted with red boxes. The bottom screenshot is the 'New Safety Mechanism Pattern Project' window. It shows a list of steps for creating a new pattern. Steps 1 through 5 are completed, and Step 6, 'Add Pattern Data', is highlighted with a red box. A 'Save SMP' button is also highlighted with a red box.

HDR Pattern Data Overview

Fill out the following additional data regarding the safety mechanism pattern. Fields marked with * must be filled out in order to proceed with pattern documentation process.

Also known as: Homogeneous Redundancy Pattern, Standby-Spare Pattern, Dynamic Redundancy Pattern, Two-Channel Redundancy Pattern

Classification*: Failure Detection & Failure Handling

Applicability*: This pattern is used to increase the safety and reliability of the system by duplicating the same channel/component to deal with the random faults. The component to be duplicated is the active one, meanwhile the duplicated one remains in standby modus, but it does not mean that it is not computing or processing data. This pattern can be applied to any level of system design.

Consequences*: This pattern is not appropriate for systematic failures, because the implementation of both components/channels is identical. If a failure is caught by the pattern the switch component will switch to one of the faulty channels, and it will lead to a complete system failure. The other negative aspect of this pattern is that information can be lost when a switch between main and redundant channel is carried out, the data in question is related to the already processed input.

Implications:

- Reliability: According to the nature of the fault, it can be classified into two types: random which is handled by this pattern and systematic which cannot be handled here. For each module, the failure rate is equal to the sum of random and systematic failure rate: Formula.
- Safety: The homogeneous duplex pattern includes two design techniques: test by redundant hardware and fault detection and diagnosis (comparator and acceptance test). According to the hardware and software requirements in the standard IEC 61508-2, 3, the recommendations for

Sample: See component architecture

Known uses: The Submarine Command System (SMCS), Intelligent-Dual Sequential Ignition (IDS)

New Safety Mechanism Pattern Project

New Safety Mechanism Pattern process steps:

Step 1: Add Safety Requirements ✓

Step 2: Add Safety Case ✓

Step 3: Add Logical Component ✓

Step 4: Add Technical Component

Step 5: Add Deployment

Step 6: Add Pattern Data ✓

Steps completed!

Now press "save SMP" to add the safety mechanism pattern to the library.

Save SMP

Figure 77: Additional data to be recorded with the HDR pattern

The documentation of the *homogeneous duplex redundancy* finishes when the *seventh* step is carried out, which consists in recording relevant data that will become useful at the moment of using this safety mechanism pattern, as seen in the figure above. In the figure, it is shown how this information is recorded in AF3. The content of this pattern attributes are shown as follows:

- **Also known as:** Homogeneous Redundancy Pattern, Standby--Spare Pattern, Dynamic Redundancy Pattern, Two--Channel Redundancy Pattern
- **Classification:** Failure handling, Failure Detection
- **Applicability:** This pattern is used to increase the safety and reliability of the system by duplicating the same channel/component to deal with the random faults. The component to be duplicated is the active one. Meanwhile, the duplicated one remains in standby modus, but it does not mean that it is not computing or processing data. This pattern can be applied to any level of system design.

- **Consequences:** this pattern is not appropriate for systematic failures, because the implementation of both components/channels is identical. If a failure is caught by the pattern, the switch component will switch to one of the faulty channels, and it will lead to a complete system failure. The other negative aspect of this pattern is that information could be lost when the switch between main and redundant channel is carried out. The data in question is related to the already processed input.

On the other hand, it is assumed that it is very unlikely that a random failure occurs at the same time in each channel (main and redundant). For this reason, this type of failure is well handled by the pattern.

- **Implications:** the following information regarding this attribute was copied from [122].
 - **“Reliability:** According to the nature of the fault, it can be classified into two types: random which is handled by this pattern and systematic which cannot be handled here. For each module, the failure rate is equal to the sum of random and systematic failure rate:

$$\lambda = \lambda_r + \lambda_s$$

$$\Rightarrow R(t) = e^{-(\lambda t)} = e^{-(\lambda_r + \lambda_s)t} = R_r R_s$$

Therefore, the reliability modeling will be represented as two serial components.

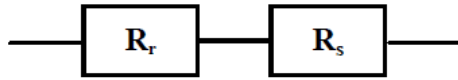


Figure 78: HDR pattern reliability modeling [122]

Assume that the switch circuit is carefully designed with *reliability* ≈ 1 . Then, the homogeneous duplex pattern will continue to work correctly as long as one of the two channels has no fault, but the two channels are identical and have the same architecture and the same systematic failure rate. So, the reliability modeling for this pattern will be as shown in the next figure.

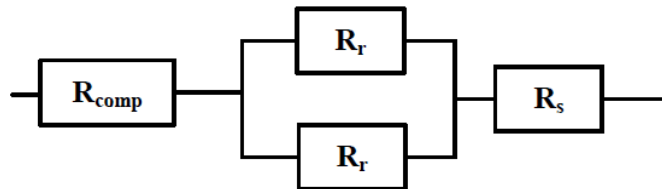


Figure 79: Reliability modeling for HDR pattern [122]

This gives:

$$R_{new} = R_{comp} (R_r^2 + 2CR_r(1 - R_r)) R_s$$

Assume that the comparator was carefully designed with $R_{comp} \approx 1$.

$$R_{new} \approx [R_r^2 + 2CR_r(1 - R_r)]R_s$$

➔ The percentage relative improvement in reliability is

$$RRI = \frac{R_{new} - R_{old}}{1 - R_{old}} \times 100\%$$

$$RRI = \frac{(R_r^2 + 2CR_r(1 - R_r))R_s - R_rR_s}{1 - R_rR_s} \times 100\%$$

– **Safety:** The homogeneous duplex pattern includes two design techniques: test by redundant hardware and fault detection and diagnosis (comparator and acceptance test). According to the hardware and software requirements in the standard IEC 61508--2, 3, the recommendations for these techniques are shown in the next table.

Techniques	SIL1	SIL2	SIL3	SIL4
Test by redundant hardware	R	R	R	R
Fault detection and diagnosis (Comparator and Acceptance Test)		R	HR	HR

Table 25: recommendations for safety integrity levels [122]

According to the last table, the average recommendations of this pattern for the different safety integrity levels are shown in the next table.

Pattern	SIL1	SIL2	SIL3	SIL4
Homogeneous Duplex Pattern	WR	R	MR	MR

Table 26: Recommendations of HDR pattern for safety integrity levels [122]

To compute RSI:

$$P_{UF(old)} = 1 - R = 1 - R_rR_s$$

The probability that the system will give an erroneous result that leads to unsafe failure is

$$P_{UF(new)} = 1 - R_{new} = 1 - R_{comp}(R_r^2 + 2CR_r(1 - R_r))R_s$$

➔ The percentage relative safety improvement is

$$RSI = \frac{P_{UF(new)} - P_{UF(old)}}{0 - P_{UF(old)}} \times 100\%$$

$$RSI = \frac{(1 - R_{comp}(R_r^2 + 2CR_r(1 - R_r))R_s) - (1 - R_rR_s)}{0 - (1 - R_rR_s)} \times 100\%$$

$$RSI = \frac{R_{comp}(R_r^2 + 2CR_r(1 - R_r))R_s - R_rR_s}{1 - R_rR_s} \times 100\%$$

Under the assumption that the comparator was carefully designed with $R_{comp} \approx 1$.

$$RSI = \frac{(R_r^2 + 2CR_r(1-R_r))R_s - R_rR_s}{1-R_rR_s} \times 100\%$$

It is clear from the equations above that the relative safety improvement is equal to the relative reliability improvement.

- **Cost:** The cost of this pattern can be classified in two parts:

○ Recurring Cost:

- Since there are two identical channels, then the recurring cost will be increased by 100% comparing to the basic module.
- The recurring cost will also include the cost of the comparator and fault detector component, which depends on the type of implementation, in addition to the cost of the switch circuit.

○ Development Cost: The development cost for the two modules (channels) is equivalent to the development cost of a single channel, because they are identical.”

- **Modifiability:** In general, the level of modifiability of this pattern is similar to the basic system (single channel) since it consists of two identical channels. The only difference exists in the comparator and fault detector. If it is required to modify the system, then the designer should take this modification into consideration and the comparator should be modified to use the new information in the fault detection process.

- **Impact on Execution Time:** During the normal operation without faults, if the comparator circuit was implemented as a hardware circuit that is running in parallel with the two channels, then there is no change in the time of execution. But if this component includes software acceptance test, then the time of execution will be affected by the time needed to execute this test. In the presence of a fault, the execution time will depend on the speed of fault detection and the required time to replace the active channel with the standby channel.

- **Sample:** see Figure 75 and Figure 76.
- **Known uses:** The *Submarine Command System* (SMCS), *Intelligent--Dual Sequential Ignition* (iDSI)

Reuse of the range check safety mechanism

The HDR pattern example is going to be used to show how the greenfield reuse process is executed in AF3. As it was stated before, this pattern makes use of the range check safety mechanism. To include it, the developer must initiate the corresponding wizard by making right click within the component architecture, or over the logical component, where the safety mechanism is going to be integrated. This action is shown in the next figure.

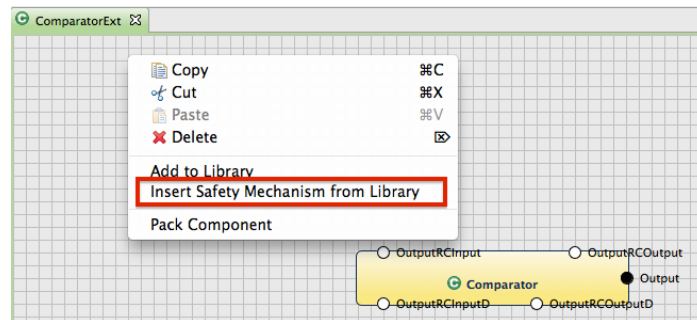


Figure 80: AF3 Popup menu with the option to insert a safety mechanism

Once the submenu is selected, the wizard depicted in the next figure is started.

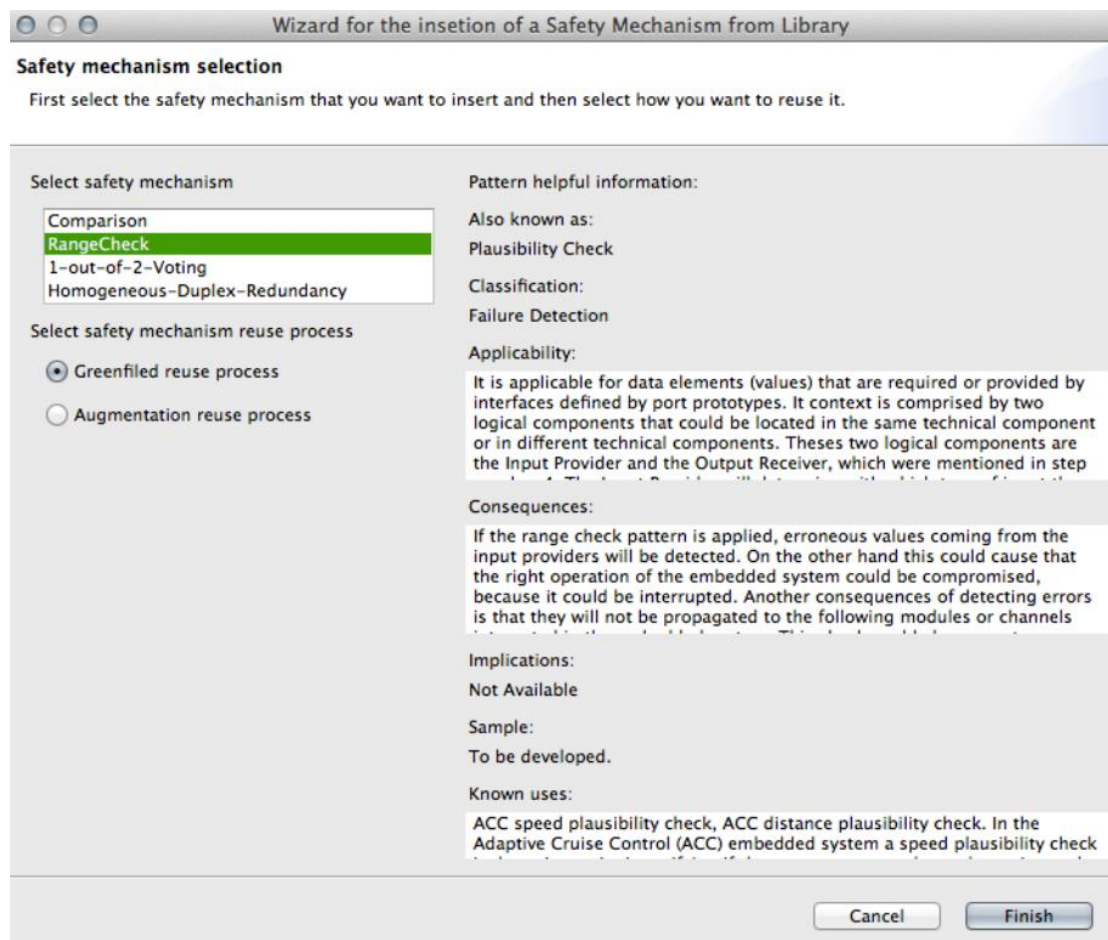


Figure 81: AF3 Wizard for the reuse of safety mechanism

The safety mechanism to be reused has to be selected by the developer, and then he/she must choose the way, in which the safety mechanism is going to be added at a logical level. Either through the augmentation or through the greenfield reuse process. The reuse process was automated in AF3; this means that the steps to be followed by the user at the moment of using a safety mechanism pattern are carried out in part automatically. The development artifacts, such as requirements and specifications, and the related safety mechanisms are added to the safety mechanism in an automated manner. However, if there are conflicts between e.g. the requirement of the safety mechanism that is being reused and the one in which it is being integrated, these have to be solved by the user, in this case the developer of the HDR pattern. In the next figure it is shown how the logical component, as well as the placeholders for the required architecture of the range check pattern, are included within the *ComparatorExt* logical component of the HDR pattern. Once the reuse of the range check is done we obtain the internal structure depicted previously in Figure 76.

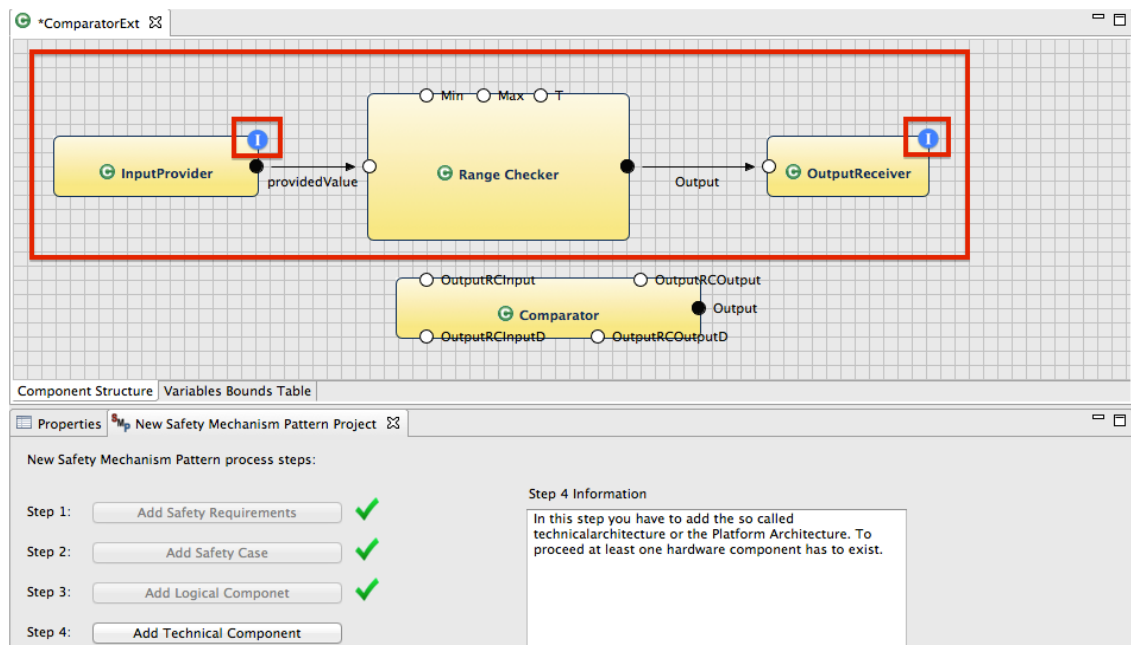
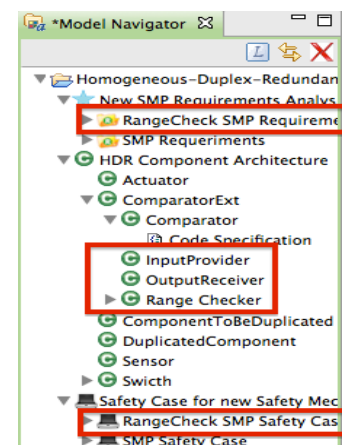


Figure 82: Insertion of the logical component of the range check pattern in the HDR pattern

Finally, in the figure to the right, it is depicted how the development artifacts of the used pattern are added to the development artifacts of the HDR pattern.

Figure 83: View of the range check development artifacts added to the HDR pattern



A4 Appendix 4 – Survey Questionnaire

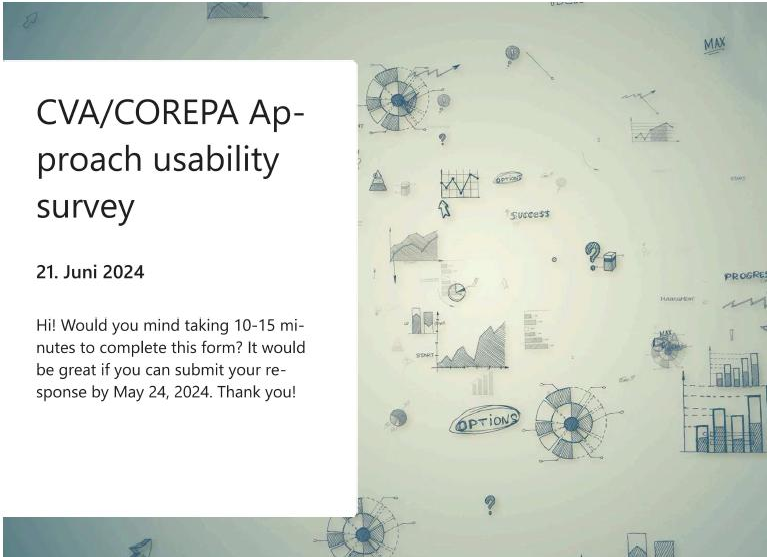
The following pages show a screen printing of the actual survey used in chapter 7.3.

21.06.24, 14:05 CVA/COREPA Approach usability survey

CVA/COREPA Approach usability survey

21. Juni 2024

Hi! Would you mind taking 10-15 minutes to complete this form? It would be great if you can submit your response by May 24, 2024. Thank you!



🔗

21.06.24, 14:05 CVA/COREPA Approach usability survey

* Erforderlich

Purpose

In this survey we aim to assess the MBSE-enabled architecture knowledge capturing and reuse approach introduced with COREPA, and expanded to CVA + Feature Catalogue.

Guiding points:

1. Your participation is completely voluntary. We need your honest, fair and unfiltered opinion to improve things.
2. Your answers are anonymous to everyone (including the survey conductors and management) and will be strictly used to provide an unbiased evaluation of the approach and guidance on future steps.
3. Please read the questions carefully. Some questions seem/sound similar or the same, but they aren't. We checked :)
4. Don't second-guess yourself or overthink your answers. Your initial instinctual answer is probably the right one.
5. Some questions allow multiple answers. This is indicated by a short hint under the question.
6. Beyond a couple of general experience questions at the beginning, you will in no way be requested to provide personal information about yourself or others. Please answer the questions accordingly.

Thank you for your time!

21.06.24, 14:05

CVA/COREPA Approach usability survey

General User Experience

In this section we are interested in understanding the respondents' overall experience

How many years have you been working in the field of Systems & Software Engineering? *

- ☐ <4 years
- ☐ 4-9 years
- ☐ 10-15 years
- ☐ 16+ years

What is your level of experience with the analysis, synthesis and optimization of architecture variants at vehicle level? *

Wählen Sie höchstens 2 Optionen aus.

- ☐ <5 years
- ☐ 5+ years
- ☐ I have been responsible for the architecture design of vehicle systems (and their deployment optimization) which have been successfully implemented and fielded.

21.06.24, 14:05

CVA/COREPA Approach usability survey

What is your level of knowledge of MBSE (Model-Based Systems Engineering) in general? *

Wählen Sie höchstens 2 Optionen aus.

- ☐ New to the topic, especially in this context.
- ☐ Familiar with a modeling tool (e.g., Rhapsody/PREEVision) / language (e.g., SysML or Arcadia/Capella), i.e. can use it myself with little or no assistance to solve my primary use-cases.
- ☐ Power-user of a modeling tool (e.g., Rhapsody/PREEVision) / language (e.g., SysML or Arcadia/Capella), i.e. can use most tool features, customize functionalities to my needs, and help others get started with using the tool.
- ☐ Familiar with multiple modeling paradigms, tools and frameworks
- ☐ I build/extend tools and develop modeling frameworks

How long have you used MBSE (Model-Based Systems Engineering) in the context of function design and system architecture development at vehicle level? *

- ☐ <3 years
- ☐ 4-10 years
- ☐ 11+ years

21.06.24, 14:05 CVA/COREPA Approach usability survey

Experience with solution reuse in general and the CVA/COREPA methodology specifically

Have you previously (before your involvement with CVA/COREPA) been involved in any initiatives related to solution reuse or reference architectures (defined as modular building blocks with a guideline to create entire architectures using them) within the organization? *

☐ Yes, involving a pragmatic informal approach emphasizing efficiency.

☐ Yes, involving a systematic framework with enforcement of reuse rules.

☐ No

Where you responsible for defining/introducing/enforcing the reuse or where you a user of the approach? *

☐ I was responsible

☐ I was a user

Did you perform any evaluations of the reuse approach used? *

☐ Yes

☐ No

21.06.24, 14:05 CVA/COREPA Approach usability survey

Which criteria did you evaluate the reuse approach against? *

☐ Effectiveness

☐ Usability

☐ All of the above

☐ Knowledge sharing

☐ Quality improvement

☐ Integration with existing workflow/methodology

☐ Adoption

☐ Support and resources

☐ Sonstiges

How long have you been using the solution reuse / reference architecture methodology introduced in CVA/COREPA? *

☐ Less than 1 year

☐ 1-3 years

☐ 3+ years

21.06.24, 14:05

CVA/COREPA Approach usability survey

How many different reference vehicle architectures did you develop using the methodology? *

- ☐ 1 reference vehicle
- ☐ 2-4 reference vehicles
- ☐ 4+ reference vehicles

21.06.24, 14:05

CVA/COREPA Approach usability survey

Evaluation of the methodology for usability

It was easy for me to learn how to use and apply the methodology. *

- ☐ Strongly agree
- ☐ Agree
- ☐ Somewhat agree
- ☐ Somewhat disagree
- ☐ Disagree
- ☐ Strongly disagree

The support, instructions and guidance provided for the methodology are easy to understand. *

- ☐ Strongly agree
- ☐ Agree
- ☐ Somewhat agree
- ☐ Somewhat disagree
- ☐ Disagree
- ☐ Strongly disagree

21.06.24, 14:05

CVA/COREPA Approach usability survey

The documentation provided for the methodology is adequate. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

Reusing solutions has caused disruptions in our architecture design process. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

21.06.24, 14:05

CVA/COREPA Approach usability survey

Evaluation of the methodology for usability

It was easy for me to understand and apply the method using the support provided. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05

CVA/COREPA Approach usability survey

The method integrates (or is integrated) well with our development workflow. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

The method behaves (can be applied and used to generate results) consistently across different projects and scenarios. (Can be skipped if you have only used it in one project. - please select N/A) *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05

CVA/COREPA Approach usability survey

Effectiveness

The method has increased the efficiency of reusing existing knowledge for creating and exploring architecture solutions. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05

CVA/COREPA Approach usability survey

Reusing solutions has saved time in the architecture development process. *

- ☐ Strongly agree
- ☐ Agree
- ☐ Somewhat agree
- ☐ Somewhat disagree
- ☐ Disagree
- ☐ Strongly disagree
- ☐ N/A - cannot make a statement

The results of applying the method provide insight to my stakeholders and capture automotive solution architecture knowledge within the organization. *

- ☐ Strongly agree
- ☐ Agree
- ☐ Somewhat agree
- ☐ Somewhat disagree
- ☐ Disagree
- ☐ Strongly disagree
- ☐ N/A - cannot make a statement

21.06.24, 14:05 CVA/COREPA Approach usability survey

Knowledge management

The method promotes knowledge sharing among team members. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

The method promotes knowledge sharing in the organization. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05 CVA/COREPA Approach usability survey

It was EASY to collect the information needed to properly apply the methodology and leverage its strengths. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

The team was able to collect ALL of the information needed to properly apply the methodology and leverage its strengths. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05

CVA/COREPA Approach usability survey

It was possible to collect ENOUGH information to leverage some strengths of the methodology and create value. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

Better access to data would vastly improve my ability to leverage the reuse methodology to easily generate architecture solutions. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05

CVA/COREPA Approach usability survey

Quality improvement

Reusing existing architecture knowledge has led to improvements in the quality of our architecture solutions. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05 CVA/COREPA Approach usability survey

The reuse approach has helped in identifying issues early and increasing confidence in the architecture results. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05 CVA/COREPA Approach usability survey

Adoption

I would recommend the methodology to other teams in our organizations trying to improve their architecture knowledge capturing and reuse. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05 CVA/COREPA Approach usability survey

It is likely that the method will be adopted by other teams addressing similar issues. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05 CVA/COREPA Approach usability survey

Organizational culture and support

Our organizational culture encourages collaboration and knowledge sharing. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05 CVA/COREPA Approach usability survey

Leadership supports and promotes initiatives related to solution reuse, concretely by providing resources and timelines allowing investment in reuse infrastructure, deployment and application. *

☐ Strongly agree

☐ Agree

☐ Somewhat agree

☐ Somewhat disagree

☐ Disagree

☐ Strongly disagree

☐ N/A - cannot make a statement

21.06.24, 14:05 CVA/COREPA Approach usability survey

Suggestions for improvement

If you used only one word or phrase to describe to the most pressing obstacle for leveraging the approach to capture and reuse architecture knowledge in the organization and create new solutions, what would it be? *

(e.g., missing tool support, it's a bad approach, too complicated, ...)

 Microsoft Forms

<https://forms.office.com/pages/designpagev2.aspx?origin=Notification&subpage=design&id=j1VLJS57ukCIH-BNeeYmWsJGKT1dphIFo0HXnjs...> 23/23

A5 Appendix 5 – Survey results

A5.1 Raw Responses

Note: In the interest of reproducibility, the following appendix section contains the raw pseudonymized Likert-scale responses collected through the survey. No aggregation has been applied here, but pseudonymization measures have been taken to preserve participant confidentiality, including sanitizing the tables from any timestamps or other digital watermarks which would break anonymity, and shuffling the participants to prevent the order of responses from indicating affiliation with respondent groups.

Table 27: Sanitized and analytically clustered survey responses (Table 1 of 2)

[illegible]

Table 28: Sanitized and analytically clustered survey responses (Table 2 of 2)

Questions \ Respondent ID	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5	Expert 6	Expert 7	Expert 8	Expert 9	Expert 10	Expert 11	Expert 12	Expert 13	Expert 14	Expert 15
Simplified Usability Segment															
It was easy for me to learn how to use and apply the methodology.			Somewhat agree						Somewhat agree	Somewhat agree		Agree	Somewhat agree	Agree	Somewhat agree
The support, instructions and guidance provided for the methodology are easy to understand.			Somewhat agree						Somewhat agree	Somewhat agree		Somewhat agree	Somewhat agree	Somewhat agree	Somewhat agree
The documentation provided for the methodology is adequate.			Agree						Somewhat disagree	Somewhat disagree		Somewhat agree	Somewhat disagree	Somewhat agree	N/A - cannot make a statement
It was possible to collect ENOUGH information to leverage some strengths of the methodology and create value.			Agree						Agree	Agree		Agree	Somewhat agree	Agree	Somewhat agree
Reusing solutions has caused disruptions in our architecture design process.			Somewhat disagree						Somewhat disagree	Somewhat disagree		Somewhat agree	Somewhat disagree	Somewhat agree	N/A - cannot make a statement
Full Usability Segment										Somewhat agree				Somewhat agree	Somewhat agree
It was easy for me to understand and apply the method using the support provided.		Strongly disagree	Agree	Strongly agree	Agree	Somewhat agree	Agree	Somewhat agree			Strongly agree				
The method integrates (or is integrated) well with our development workflow.		Disagree	Strongly agree	Agree	Disagree	Agree	Agree	Somewhat agree			Agree				
It was possible to collect ENOUGH information to leverage some strengths of the methodology and create value.		Somewhat disagree	Agree	Agree	Somewhat agree	Agree	Agree	Somewhat agree			Strongly agree				
The method behaves (can be applied and used to generate results) consistently across different projects and scenarios. (Can be skipped if you have only used it in one project - please select N/A)						N/A - cannot make a statement	Strongly agree	Somewhat agree			Strongly agree				
Effectiveness and Efficiency Segment															
The method has increased the efficiency of reusing existing knowledge for creating and exploring architecture solutions.		Somewhat disagree	Agree	Strongly agree	Agree	Agree	Agree	Agree	Agree	Agree	Strongly agree	Strongly agree	Somewhat agree	Somewhat agree	Somewhat agree
Reusing solutions has saved time in the architecture development process.		Disagree	Agree	Strongly agree	Strongly agree	Somewhat agree	Agree	Agree	Somewhat agree	Somewhat agree	Strongly agree	Agree	Somewhat disagree	Somewhat agree	Somewhat disagree
The results of applying the method provide insight to my stakeholders and capture automotive solution architecture knowledge within the organization.		Somewhat disagree	Agree	Agree	Somewhat disagree	Agree	Strongly agree	Agree	Agree	Agree	Agree	Agree	Somewhat agree	Agree	Agree
Impact Segments															
Knowledge Sharing Segment															
The method promotes knowledge sharing among team members.		Agree	Strongly agree	Strongly agree	Somewhat agree	Agree	Strongly agree	Agree	Strongly agree	Agree	Strongly agree	Strongly agree	Somewhat agree	Agree	Agree
The method promotes knowledge sharing in the organization.		Disagree	Agree	Strongly agree	Somewhat disagree	Agree	Strongly agree	Agree	Somewhat agree	Somewhat agree	Agree	Agree	Somewhat disagree	Somewhat agree	Somewhat disagree
Quality Improvement Segment															
Reusing existing architecture knowledge has led to improvements in the quality of our architecture solutions.		N/A - cannot make a statement	Agree	Agree	Agree	Agree	Strongly agree	Agree	Agree	Agree	Strongly agree	Agree	Somewhat	Agree	Agree
The reuse approach has helped in identifying issues early and increasing confidence in the architecture results.		N/A - cannot make a statement	Somewhat agree	Strongly agree	Somewhat agree	Agree	Strongly agree	Agree	Agree	Somewhat agree	Agree	Agree	Somewhat agree	Somewhat agree	Somewhat agree
Adoption Segment															
I would recommend the methodology to other teams in our organizations trying to improve their architecture knowledge capturing and reuse.		Somewhat agree	Agree	Strongly agree	Agree	Agree	Strongly agree	Strongly agree	Agree	Agree	Strongly agree	Strongly agree	Agree	Agree	Strongly agree
It is likely that the method will be adopted by other teams addressing similar issues.		Somewhat disagree	Disagree	Agree	Somewhat disagree	Somewhat agree	Somewhat agree	Somewhat agree	Somewhat agree	Somewhat agree	Somewhat agree	Somewhat disagree	Somewhat disagree	Somewhat agree	Somewhat agree
Open Question: Suggestions for Improvement															
If you used only one word or phrase to describe to the most pressing obstacle for leveraging the approach to capture and reuse architecture knowledge in the organization and create new solutions, ...	information and knowledge handling and sharing	lack of adoption of MBSE	reliable and consistent data collection	one order of magnitude more efficient than any other approach currently used in-house	BA support for collecting necessary data	Not enough internal champions	data access missing	MBSE gap	missing architecture knowledge from business units	knowledge sharing	support with information	Time constraints	Resistance from other teams	Information access	Missing data

A5.2 Aggregated Numerical Scores

Table 29: Survey Results: Aggregated Numerical Scores

	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5	Expert 6	Expert 7	Expert 8	Expert 9	Expert 10	Expert 11	Expert 12	Expert 13	Expert 14	Expert 15
Contextual Factors															
Organizational Support Index (OSI)	1,00	3,50	3,00	4,50	4,50	4,00	3,50	3,50	3,50	3,50	4,50	3,50	2,50	3,00	3,00
Self-Assessment Experience Factor	0,60	0,73	0,80	0,87	0,47	0,60	0,67	0,60	0,60	0,60	0,73	0,67	0,53	0,47	0,27
Objective Qualifiers Factor	0,27	0,60	0,70	1,00	0,20	0,73	0,87	0,67	0,67	0,53	0,80	0,50	0,40	0,27	0,27
Calibrated Experience Factor	0,47	0,68	0,76	0,92	0,36	0,65	0,75	0,63	0,63	0,57	0,76	0,60	0,48	0,39	0,27
MBSE expertise self-assessment	0,60	0,60	0,80	1,00	0,60	0,60	0,80	0,60	0,60	0,60	1,00	0,80	0,80	0,60	0,40
MBSE calibrated expertise	0,28	0,41	0,61	0,92	0,22	0,39	0,60	0,38	0,38	0,34	0,76	0,48	0,38	0,23	0,11
Metrics															
Unweighted Usability Score	2,25	5,50	4,50	5,50	3,67	4,67	5,25	4,00	4,25	4,00	5,75	4,50	3,75	4,50	4,00
Self-Assessment Weighted Usability	2,20	6,58	5,87	7,77	2,79	4,57	5,71	3,91	4,16	3,91	6,88	4,89	3,26	3,42	1,74
Calibrated Weighted Usability	1,77	6,30	5,76	8,52	2,22	5,13	6,60	4,22	4,49	3,86	7,36	4,55	3,03	2,93	1,80
Unweighted Effectiveness and Efficiency Scores	2,67	5,00	5,33	5,67	4,67	4,67	5,33	5,00	4,67	4,67	5,67	5,33	3,67	4,33	4,00
Self-assessment weighted Effectiveness and Efficiency	2,61	5,98	6,96	8,01	3,55	4,57	5,80	4,89	4,57	4,57	6,78	5,80	3,19	3,30	1,74
Calibrated weighted Effectiveness and Efficiency	2,10	5,73	6,83	8,78	2,83	5,13	6,71	5,28	4,93	4,51	7,25	5,39	2,96	2,82	1,80
Unweighted Knowledge Sharing Scores	3,50	5,00	5,50	6,00	3,50	5,00	6,00	5,00	5,00	4,50	5,50	5,50	3,50	4,50	4,00
Self-assessment weighted Knowledge Sharing	3,42	5,98	7,17	8,48	2,66	4,89	6,52	4,89	4,89	4,40	6,58	5,98	3,04	3,42	1,74
Calibrated weighted Knowledge Sharing	2,75	5,73	7,04	9,30	2,12	5,50	7,54	5,28	5,28	4,35	7,04	5,56	2,83	2,93	1,80
Unweighted Quality Improvement Scores	N/A	5,00	4,50	5,50	4,50	5,00	5,50	5,00	5,00	4,50	5,50	5,00	4,00	4,50	4,50
Self-assessment weighted Quality Improvement	N/A	5,98	5,87	7,77	3,42	4,89	5,98	4,89	4,89	4,40	6,58	5,43	3,48	3,42	1,96
Calibrated weighted Quality Improvement	N/A	5,73	5,76	8,52	2,73	5,50	6,92	5,28	5,28	4,35	7,04	5,05	3,23	2,93	2,02
Unweighted Adoption Scores	3,50	5,50	4,00	5,50	4,00	4,50	5,00	5,00	4,50	4,50	5,00	4,50	4,00	4,50	5,00
Self-assessment weighted Adoption	3,42	6,58	5,22	7,77	3,04	4,40	5,43	4,89	4,40	4,40	5,98	4,89	3,48	3,42	2,17
Calibrated weighted Adoption	2,75	6,30	5,12	8,52	2,43	4,95	6,29	5,28	4,75	4,35	6,40	4,55	3,23	2,93	2,25

A5.3 Supplementary Visualizations (3D Clustered Charts by Segment)

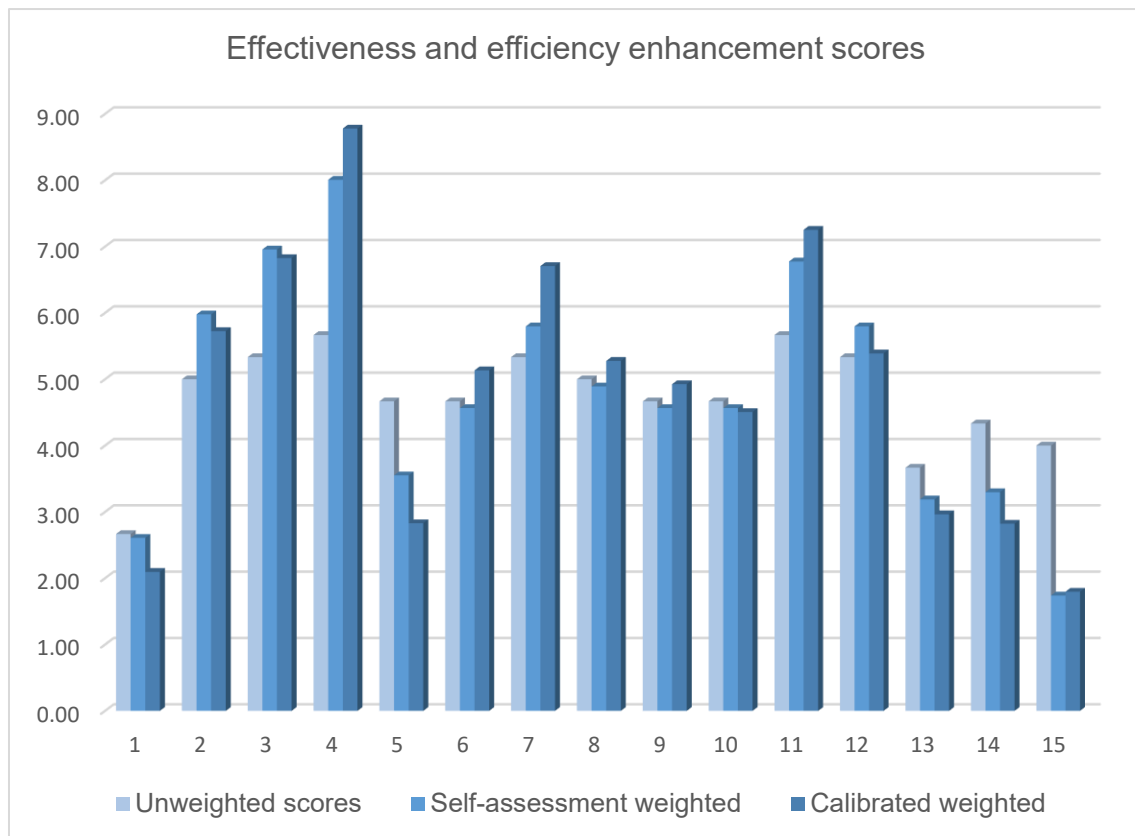


Figure 84: Aggregated weighted and unweighted Effectiveness and Efficiency evaluations for all 15 experts

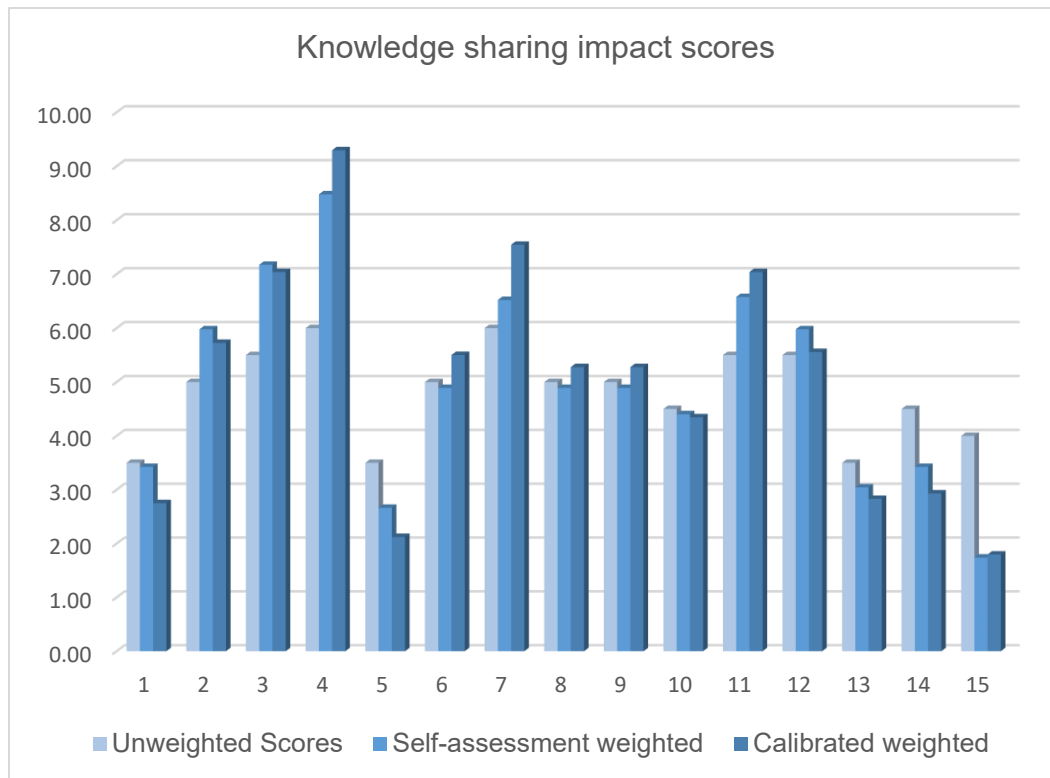


Figure 85: Aggregated weighted and unweighted Knowledge Sharing evaluations for all 15 experts

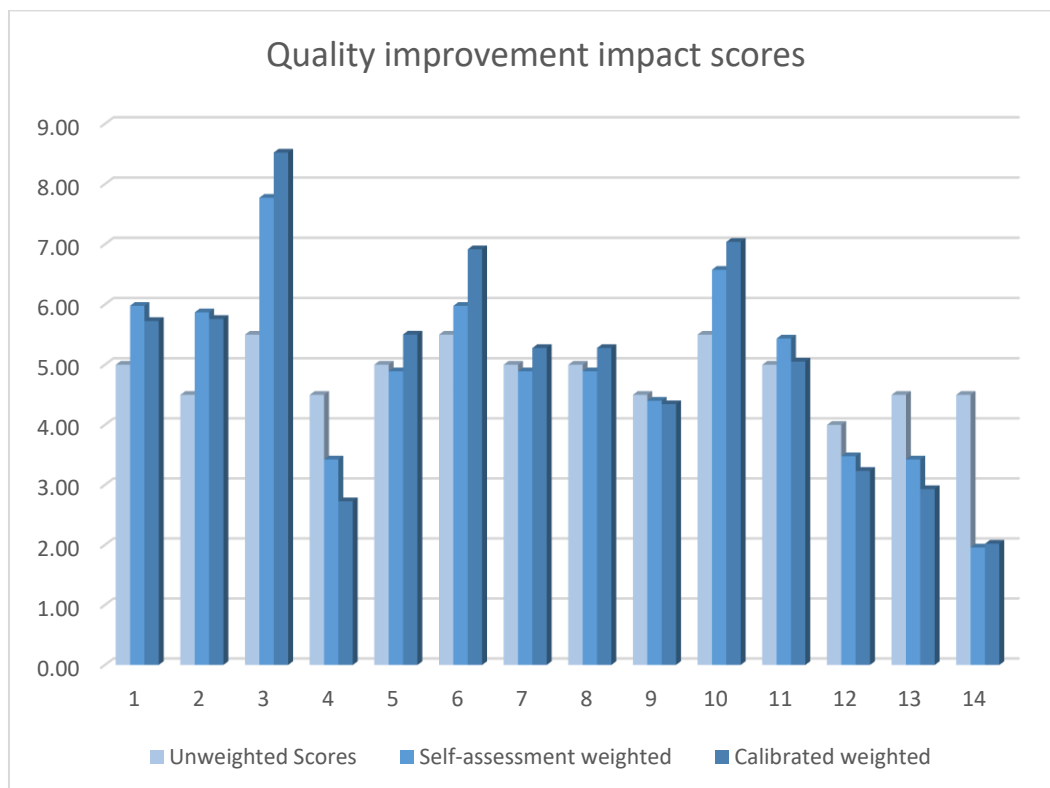


Figure 86: Aggregated weighted and unweighted Quality Improvement evaluations for all 15 experts

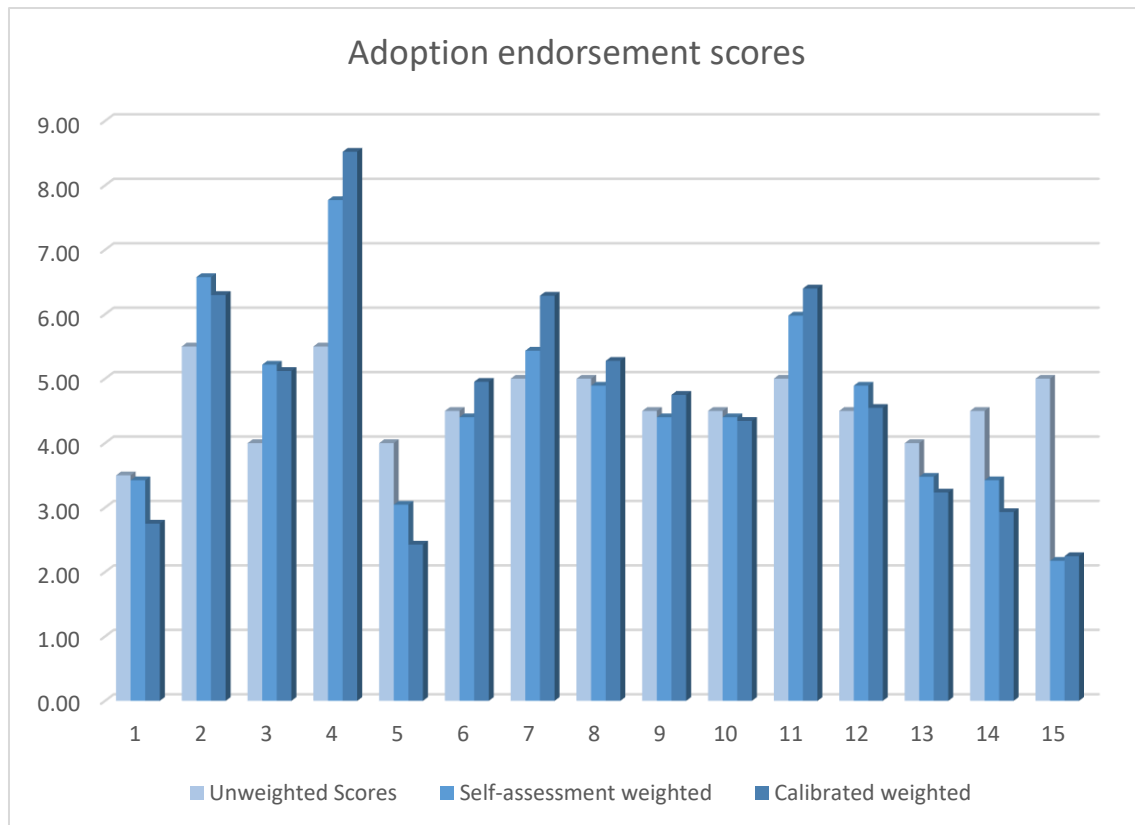


Figure 87: Aggregated weighted and unweighted Adoption evaluations for all 15 experts