CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Enabling Continuous Compliance with Product-Repository Integrated Traceability Management Tool

Master's thesis in Computer science and engineering

Jinlu Yu

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Enabling Continuous Compliance with Product-Repository Integrated Traceability Management Tool

Jinlu Yu

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Enabling Continuous Compliance with Product-Repository Integrated Traceability Management Tool

Jinlu Yu

Supervisor: Eric Knauss, Computer Science and Engineering
Examiner: Jennifer Horkoff, Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Enabling Continuous Compliance with Product-Repository Integrated Traceability Management Tool

Jinlu Yu
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Continuous compliance is becoming increasingly important as software-intensive systems are developed in regulated domains, where both systems and regulatory expectations evolve over time. Traditional compliance approaches often rely on manual evidence collection and periodic assessment, making it difficult to maintain compliance during continuous development. This thesis investigates how product-repository-integrated traceability management (PRITM) tools can support continuous compliance by managing compliance-relevant artifacts, traceability links and checks within a product repository.

The study follows a Design Science Research approach with three iterations. For RQ1, literature review and workshops were used to develop a reference model of continuous compliance. The main finding is that software development activities support continuous compliance by producing and updating artifacts that can serve as evidence of compliance. However, these artifacts only become useful evidence when they are traceable, maintained and connected to compliance requirements and compliance checks.

For RQ2, the reference model was instantiated via a TReqs plugin prototype named treqs-compliance. The prototype uses ISO 26262 Part 8 Clauses 7 and 8 as an example to demonstrate how standards, clauses, compliance requirements, and work products can be represented and maintained in a product repository. The prototype implements rule-based structural checks, lifecycle status handling, change-review detection, and compliance report generation. The main finding is that PRITM tools can support continuous compliance by keeping compliance artifacts close to development artifacts, managing traceability, and enabling lightweight automated checks.

For RQ3, the prototype and final reference model were used to reflect on where effort could be reduced compared to traditional compliance approaches. The results suggest that PRITM tools can mainly reduce effort for structural and traceability-related tasks, such as checking artifact existence, identifying required fields, and locating missing links. However, semantic content evaluation, runtime context checks, and structured compliance argumentation remain outside the implemented scope and still require human judgment or more advanced tool support.

This thesis contributes a reference model for understanding continuous compliance across activities and artifacts, with a particular focus on the DevOps lifecycle. It also provides a prototype demonstration showing how selected concepts can be implemented in a PRITM tool.

Keywords: continuous compliance, compliance as code, traceability, Requirements Engineering (RE), ISO 26262

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Eric Knauss, for leading me into this research project, even when I had very limited prior knowledge of this field. His continuous guidance, patience, and insightful feedback helped me make progress and finally complete this thesis. I would also like to thank my examiner, Jennifer Horkoff, for her constructive comments and suggestions throughout the process of writing the thesis.

I would like to thank TReqs Technologies for their technical guidance and support during the development of the prototype. Their input helped me to gain a better understanding of the tool ecosystem and the challenges of implementation. I would also like to thank the participants of the Software Center workshops for sharing their time, experience and insights. Their feedback was invaluable in shaping the reference model and enhancing the practical relevance of this work.

Jinlu Yu, Gothenburg, 2026-06-20

Contents

1

Introduction

Adherence to regulations and standards is a major concern for software developed in highly regulated domains, particularly for safety-critical, security-relevant, and privacy-sensitive systems. For example, in the automotive domain, standards for functional safety such as ISO 26262 impose rigorous requirements on development processes and safety assurance. Complementing such domain-specific regulation, recent European regulatory frameworks address additional aspects of software-intensive systems, including privacy (e.g. GDPR), cybersecurity (e.g. the Cyber Resilience Act), and artificial intelligence (e.g. the EU AI Act). Together, these developments further increase the importance of compliance competencies for companies developing complex software systems.

Achieving compliance is a resource-intensive endeavor, often requiring substantial engineering effort from senior engineers and domain experts. Compliance work typically involves significant time and cost to interpret regulations and standards, implement compliance-related processes, and document, review, and audit compliance artifacts.

Traditionally, compliance with standards has been assessed at specific milestones and treated as a one-time activity. However, in rapidly evolving development environments, where software systems, requirements, and regulations continuously change, such an approach is no longer sufficient. As a result, continuous compliance has become increasingly important. Santilli et al. [1] define continuous compliance as ensuring a systems compliance with standards throughout its lifecycle, supported by techniques such as traceability, versioning, and automated documentation. Similarly, Angermeir et al. [2] describe continuous security compliance as a set of practices that ensure product and process adherence to regulatory requirements, holistically integrated into the software lifecycle and aligned with continuous engineering principles.

These perspectives highlight the need for tool support that enables automated, traceability-driven compliance activities closely integrated with software development workflows. In this thesis, we investigate how product-repository-integrated traceability management (PRITM) tools can support continuous compliance. More specifically, the research explores how TReqs, as a representative PRITM tool, can be used to manage compliance-related artifacts, traceability links, and check results within a product repository.

This investigation is conducted through an in-depth case based on ISO 26262 Part

8 compliance, with particular attention to tool-supported software compliance. The case focuses on how requirements derived from the standard can be connected to repository artifacts and how selected structural compliance checks can be automated in the treqs-compliance plugin. While ISO 26262 originates from the automotive domain, the underlying challenges of maintaining traceability, managing compliance evidence, and responding to change are also relevant for other regulated software-intensive domains.

1.1 Problem Statement

Existing research has recognized the importance of continuous compliance as continuous software engineering is increasingly adopted in safety or security regulated environments [2]. However, existing approaches remain fragmented and insufficient to support continuous compliance across the entire software lifecycle.

Both software systems and their regulatory environments are subject to continuous change. Rouland et al. [3] argue that the constant evolution of software systems calls for automation tools to support continuous compliance processes, while the ongoing evolution of standards and regulations demands a formal and traceable approach to the elicitation and management of compliance requirements. In addition, Santilli et al. [1], through a systematic literature review, found that existing solutions typically address isolated compliance problems or specific development phases, lacking a holistic, end-to-end perspective.

As a result, there is a lack of practical approaches that integrate automated and traceable compliance support directly into software development activities and artifacts, enabling continuous compliance in evolving software systems.

1.2 Purpose of the Study

The purpose of this study is to investigate to what extent product-repository-integrated traceability management (PRITM) tools can support continuous compliance in evolving software systems.

Based on design science principles, this thesis presents and evaluates an approach that uses PRITM to integrate compliance-relevant information, artifacts, trace links and check results directly into product repositories. This approach is motivated by the observation that compliance evidence is often produced or changed alongside development artifacts. Storing these elements alongside code and requirements in the same repository can help to ensure that compliance-related artifacts and the trace links between them are consistent, version-controlled and updated in line with software development.

Using TReqs as a representative PRITM tool, the study examines how repository-level traceability can satisfy the information requirements of continuous compliance and reduce the manual effort involved in compliance activities. The treqs-compliance plugin is used as a case study because it provides a repository-based environment

for representing compliance-related artifacts, offers strong traceability management capabilities and can be extended with rule-based checks. This makes it a suitable example for investigating how PRITM tools can support continuous compliance in practice.

The results of this study are expected to benefit software engineers, compliance engineers, and organizations operating in regulated domains by providing practical insights into tool-supported, traceability-based solutions for continuous compliance.

1.3 Research Questions

RQ1: How do software development activities and their resulting artifacts support the collection and maintenance of compliance evidence for continuous compliance?

In this thesis, compliance is treated as an evidence-based argumentation process, in which compliance claims are supported by systematically collected, maintained, and traceable evidence derived from software development activities and artifacts.

Answering RQ1 is necessary for defining the conceptual scope of continuous compliance. Before designing repository-level support, it is important to understand which activities and artifacts are involved in continuous compliance and how they relate to evidence collection and maintenance. Thus, the answer to RQ1 will provide the basis for developing an initial reference model and guide the TTIM design used in the prototype.

RQ2: How can a product-repository-integrated traceability management (PRITM) tool such as TReqs support continuous compliance?

In this thesis, PRITM tools are assumed to support continuous compliance by managing compliance-relevant artifacts directly within product repositories and by maintaining traceability between standards, compliance requirements, system artifacts, checks, and evidence.

Answering RQ2 is necessary to understand how continuous compliance can be supported beyond a conceptual process model. In particular, it examines how treqs-compliance can be used to represent compliance-related artifacts, connect them through trace links, and support lightweight automated checks and evidence reporting within the development repository.

RQ3: To what extent can PRITM tools reduce the effort required to achieve continuous compliance compared with traditional approaches?

This research question evaluates whether and to what extent the PRITM-based approach reduces the effort required for achieving continuous compliance compared with traditional manual compliance practices. The evaluation focuses on the level of automation achieved, the role of integrated traceability in reducing manual assessment work, and the remaining activities that still require human judgment.

1.4 Scope

This thesis was conducted in the context of Software Center Project 66, 'Compliance as Code'. The project investigates how compliance-related knowledge, evidence and checks can be represented and maintained within software-intensive development environments. This context provided access to existing TReqs-based tooling, previous work on continuous compliance, and feedback from industrial partners involved in regulated software development.

This thesis builds on previous work [4] using TReqs to support traceability and automated compliance checking in the development of safety-critical systems. Earlier work focused on enhancing TReqs with the BOMI metamodel to improve coordination, represent boundary objects, assign responsibilities and support automated compliance checking for ISO 26262 work products in large-scale agile development projects. The scope of this thesis is different. Rather than focusing on team coordination or role assignment through BOMI, this thesis addresses continuous compliance through an artifact-based and traceability-supported approach.

The main objective is to develop a reference model that illustrates the continuous compliance lifecycle. The practical implementation in this thesis is limited to treqs-compliance, a plugin of TReqs. The prototype uses ISO 26262 Part 8 Clauses 7 and 8 as an example and focuses on repository-level traceability, rule-based structural checks, lifecycle status, change review, and compliance report generation. Since the demonstration checks the compliance of artifacts managed within the plugin itself, it should be understood as a software compliance example rather than a full industrial certification scenario.

The thesis does not address how standards are interpreted or transformed into compliance requirements. This step is considered an input to the proposed approach, regardless of whether it is carried out manually, by experts, using AI-based tools, or by other methods. The study assumes that the relevant clauses and compliance requirements have already been identified.

Based on this, the focus of the implementation is on how these requirements can be represented, traced, maintained and verified in a product repository. The implemented checks primarily address structural properties, such as artifact existence, traceability links, the presence of required fields, and the review of changed artifacts. More complex tasks, such as semantic content evaluation, risk-based judgment, runtime context checks and full compliance argument generation, fall outside the scope of the implementation.

Therefore, the thesis makes two contributions. First, it provides a reference model to help understand continuous compliance across activities and artifacts, with a particular focus on DevOps-related evidence sources. Secondly, it shows how certain elements of the model can be put into practice using a PRITM tool, such as TReqs.

2

Background

2.1 Continuous Compliance (CC)

2.1.1 Definition

The need for continuous compliance has been recognized in recent research [1], [2], particularly as continuous software engineering practices are adopted in safety- and security-regulated contexts. However, for a long time, there has been neither a clear definition of continuous compliance nor comprehensive solutions that cover the full lifecycle of software development.

Santilli et al. [1] conducted a systematic literature review and identified 13 primary studies that focus on compliance with safety and security standards. They defined continuous compliance as guaranteeing a system's compliance with the standard through its lifecycle, which may involve different techniques and approaches. These techniques are expected to support traceability, versioning, and automated documentation generation and maintenance. In addition, they highlight the importance of organizational factors such as security culture awareness and continuous cross-team communication.

Angermeir et al. [2] conducted a study specifically targeting Continuous Security Compliance (CSC). They defined CSC as a set of practices that ensure both product and process adherence to security-related regulatory requirements, integrated holistically into the software lifecycle while following continuous engineering principles. They identified a total of 21 relevant challenges that hinder automation and efficiency in this domain through literature study, and presented a long-term research roadmap designed to address these challenges.

2.1.2 Continuous Compliance in Agile Development

Continuous software engineering (CSE), which leverages lean and agile principles, has been widely adopted due to its iterative, adaptive, and efficient nature. However, an inherent tension exists between agile methodologies and traditional compliance processes. Traditional compliance process typically follows linear and plan-driven workflows, culminating in "big-bang" audits near release milestones. The incremental and frequent changes in agile development would complicate key assurance activities such as traceability maintenance, risk analysis, and audit preparation [5]. Moreover,

compliance assessment is often document-intensive and relies on manual review, making it difficult to keep pace with rapid agile development [6].

Therefore, while the holistic vision of CSE aims to address both agile and compliance concerns, existing practices still tend to address them in isolation [7]. Prior research has explored approaches to integrate compliance workflows into agile environments [5], [8]. Nevertheless, achieving such an integration remains a challenge. The stakeholders involved in development and compliance form different methodological islands, relying on different frameworks, tools, and knowledge domains, which hinders effective collaboration and alignment [8]. For example, standards are typically difficult for developers to interpret in context while compliance experts may lack sufficient visibility into technical details.

Despite growing interest in continuous compliance, there remains a lack of automated solutions that can simultaneously support the integration of assurance activities into agile workflow while enabling systematic management, versioning, and traceability of compliance-relevant artifacts.

2.2 TReqs

TReqs[1] is a lightweight approach to requirements management that was developed at Chalmers University of Technology and University of Gothenburg in collaboration with industrial partners, including Ericsson, as part of the Software Center research project. It was created to address the coordination and traceability challenges commonly encountered in large-scale agile development environments, and to enable parallel work on shared artifacts.

Rather than relying on traditional database-based requirements tools, TReqs stores requirements as structured text files within a Git version control repository. This allows development teams to handle requirement updates using familiar version control practices, supporting concurrent modifications, transparent change tracking and closer integration between requirements, implementation and testing activities. This approach has been developed through empirical studies conducted with multiple industrial organizations from various sectors, including telecommunications, automotive and manufacturing systems [9], [10].

2.2.1 The Compliance-Relevant Plugin

Earlier master thesis work [4] has explored compliance support in TReqs, and a prototype extension called treqs-compliance[2] was developed as an add-on to the TReqs ecosystem. This extension provides additional functionality to help teams manage compliance-related information alongside requirements artifacts. The tool supports traceability between compliance elements and development artifacts by leveraging the same version control mechanisms. It also allows compliance checks to be incorporated into development practices.

[1] https://gitlab.com/treqs-on-git/treqs-ng
[2] https://gitlab.com/treqs-on-git/treqs-compliance

Building upon this prototype, the present research investigates how TReqs can further support continuous compliance practices in an evolving regulatory context.

This thesis rebuilds the treqs-compliance extension. The goal is to investigate how a PRITM tool can support the collection, maintenance, and review of compliance evidence over time. Therefore, treqs-compliance is used as a lightweight demonstration environment for operationalizing selected theoretical concepts related to continuous compliance.

2.2.2 The TReqs Type and Trace Information Model

Within the TReqs ecosystem, the Type and Trace Information Model (TTIM) provides a structured mechanism for defining requirement artifact types and managing traceability between them. Through the explicit specification of types and trace links, TTIM facilitates interoperability between artifacts and contributes to improved maintainability of requirements information over time. An example TTIM instance of TReqs illustrating how types and traces can be defined and applied in practice is shown in Figure 2.1.

2.2.3 The BOMI Model

In large-scale system development, there's often a challenge in knowledge coordination between different organizational teams. The BOMI (Boundary Objects and Methodological Islands) modeling framework was developed to address this issue and guide knowledge management in practice [11]. "Boundary objects (BOs)" is originally a concept in sociology [12]. It has been appropriated in software and system engineering, referring to artifacts that serve as interfaces and create a common understanding between multidisciplinary teams. These teams are often surrounded by other organizational parts that use different methods, and thus become Methodological Islands (MI) [13]. MIs can be at different abstraction levels from individual teams to the entire organization. In TReqs, BOMI can be used to support coordination among requirement engineers, developers, and standard specialists, who are essential for achieving continuous compliance. The artifacts that can be used for coordination are also various, examples include models, high-level requirements and code.

Previous research has proposed the BOMI metamodel and evaluated different instance models[11], [14]. In the development of treqs-compliance, a BOMI model was also instantiated for this extension. This model is defined using the TTIM, which is specified in the treqs-compliance repository, and integrated into the original TTIM of TReqs.

Such coordination concerns have been identified as critical organizational factors influencing the successful adoption of continuous compliance practices in industry [1]. While the earlier treqs-compliance work has focused on modeling communication structures using BOMI, this research will shift the emphasis towards investigating tool-supported mechanisms that enable continuous compliance in evolving development environments.

ttim.yaml

```yaml
---
name: Sample T-Reqs Type and Trace Information Model (TTIM)
version: 0.0.2
description: An example how types and traces can be defined in T-Reqs.
author: Grischa Liebel, Eric Knauss
types:
# ###
# Requirements are in fact system requirements: A condition or capability that must be
# met by treqs to satisfy its intended use (which we aim to express through stakeholder
# requirements, see below).
- name: requirement
  outlinks:
#   ##
#   Requirements may relate to any other treqs element and this can be expressed through
#   the relatesTo trace link.
#   They can also be a refinement of another requirements, which can be expressed
#   through the hasParent linktype
#   Finally, they should directly or indirectly (e.g. through a parent) be motivated by
#   a stakeholder-requirement
  - type: relatesTo
  - type: hasParent
    target: requirement
  - type: addresses
    target: stakeholder-requirement
# ###
# Stakeholder needs describe a challenge that key stakeholders of treqs have.
- name: stakeholder-need
# We do not anticipate the need to trace from stakeholder needs to any elements.
  outlinks:
  - type: relatesTo
    target: stakeholder-need
# ##
# Stakeholder requirements are requirements from a stakeholder's point of view and
# capture our understanding on why somebody would like to use a tool such as treqs
- name: stakeholder-requirement
  outlinks:
#   ##
#   Stakeholder requirements are often motivated by stakeholder needs; in fact, they are
#   our interpretation of what is needed to solve the stakeholder's problem
#   Stakeholder requirements can also relate to other stakeholder requirements.
```

Figure 2.1: TTIM of TReqs

2.3 ISO 26262

This research adopts ISO 26262:2018 [15] as the primary reference standard for investigating compliance practices in the development of safety-critical software. The rapid integration of complex electronic systems and software in modern vehicles has significantly increased the risk of malfunctions. To address these challenges, the ISO 26262 standard, titled "Road vehicles Functional safety", was established as the definitive international standard for the automotive industry. ISO 26262 is widely recognized as one of the most comprehensive industrial safety standards, providing an end-to-end framework for managing functional safety throughout the system and software lifecycle. It defines structured processes for hazard analysis, safety requirement specification, verification planning, traceability management, and documentation of safety evidence [16].

2.3.1 ISO 26262 Part 8

The ISO 26262 standard comprises a comprehensive suite of twelve parts, consisting of ten normative components and two informative guidelines. Among these components, this research focuses specifically on Part 8, which defines supporting processes for functional safety management. Unlike earlier parts of the standard that primarily address concept-level safety analysis or system design activities, Part 8 translates safety compliance into everyday engineering practices. This includes processes related to configuration and change management, documentation, distributed development coordination, traceability maintenance, and tool qualification.

These supporting processes interact directly with requirements engineering activities and digital tool infrastructures. In industrial environments, compliance is not achieved solely through high-level safety analyzes but also through the continuous management of safety-relevant artifacts and their relationships. Tools such as TReqs play a central role in requirements management by enabling traceability, impact analysis, structured safety argumentation, and collaboration across organizational boundaries. Therefore, focusing on Part 8 allows us to investigate how continuous compliance obligations can be embedded in routine development workflows and supported by PRITM tools.

3

Related Work

3.1 Continuous Compliance Approaches

Continuous compliance has been interpreted and implemented in different ways in both research and industry practices. Existing approaches vary in many different aspects. We did a literature review and summarized several representative solution strategies.

3.1.1 End-to-End Compliance Process Automation

Traditionally, a systematic compliance process can be divided into several distinct stages: eliciting compliance requirements, enforcing controls during development, collecting evidence before audit, and presenting audit documentation to reviewers [6]. Therefore, to achieve continuous compliance, a logical strategy is to automate these activities across the entire lifecycle. For example, Ghaisas et al. [17] propose a comprehensive framework that automates all four stages, including regulation interpretation via Natural Language Processing (NLP), traceability mapping of the requirements to security controls using topic modeling, identification and implementation of security control, as well as verification and reporting. However, this holistic approach focuses primarily on the static alignment of requirements and controls, overlooking the need for real-time adjustments to account for changes.

In line with this perspective, Rouland et al. [18] present a systematic process for deriving a minimal set of architecture requirements from multiple security standards. They clearly specify re-entry points when different types of change occur,providing a solid basis for maintaining continuous compliance. But this framework remains preliminary, the core steps are manual and there's no mechanism for the enforcement of security controls.

3.1.1.1 Prerequisites: Compliance Requirements Extraction

The first stage, eliciting requirements from regulatory texts, has been extensively researched as a foundational prerequisite. Breaux et al.'s early seminal work [19] established the semantic basis for this field by extracting 'rights and obligations' to align software requirements with regulations. Since then, the focus has shifted toward creating machine readable standards through advanced extraction techniques.

Recent systematic evaluations of these techniques highlight a transition from traditional rule-based methods to learning-based approaches, including using supervised models (e.g. CNN, SVM) and unsupervised Transformer-based architectures [20]. For instance, tools such as COREQQA leverage QA systems to help engineers query legal documents in natural language [21], while other frameworks utilize NLP to extract requirements from raw PDF content [17], [22].

As these foundational extraction and modeling techniques have reached a degree of maturity, our research does not focus on the linguistic challenges of requirement elicitation, but rather addresses the downstream challenge of maintaining traceability, evidence, and compliance argumentation throughout the dynamic evolution of software systems.

3.1.2 Process-Integrated Continuous Compliance

One approach to continuous compliance is process-oriented, embedding requirements directly into the development lifecycle. For example, Moyon et al. [5] propose extending agile process models by mapping security standard requirements to development roles, activities, and artifacts. The method is evaluated by extending Scaled Agile Framework (SAFe) according to requirements of IEC 62443-4-1. Similarly, Shenouda et al. [8] also attempted to integrate compliance processes with agile workflows. They introduced boundary objects as intermediaries between safety assurance practices and agile tasks. These persistent and shared references aim to improve traceability and facilitate coordination across teams. While process-oriented approaches provide a logical foundation for continuous compliance, they are often highly context-dependent (limited to specific domains or standards). And these methods tend to neglect automated verification of development artifacts, such as code and documentation.

3.1.3 Automated Verification-Based Compliance

By contrast, some other researchers adopt an artifact-centric perspective, focusing on the automated verification of technical products. These methods generally follow two primary paradigms: rule-based and ontology-based approaches [6].

3.1.3.1 Rule-Based Methods

Rule-based methods typically require data to be normalized before it can be matched against predefined compliance rules. For example, Filepp et al. [23] propose a solution that targets the continuous compliance of services. The solution includes a framework for managed IT services, which integrates heterogeneous compliance enforcement scripts under a unified interface. In this approach, the state of endpoints is matched against the rules specified in the policy map in order to determine which policies should be enforced. The framework ensures automation by handling heterogeneous assets consistently, in the meantime offers high customizability through user-defined configurations and exception management. However, this method does not address software compliance.

Some products are naturally suitable for formal verification, such as code. Taking advantage of this feature, Kellogg et al. [24] focused on data-security-related compliance requirements specifically for code. They developed a lightweight tool that verifies code whenever a change is made, generating warnings and building audit trails. This tool could facilitate the automated collection of evidence during compliance reviews. Since this form of verification mirrors traditional testing, it could significantly reduce manual effort and increase automation. However, while code is well-suited for such formal verification, other development artifacts (such as high-level design documents) are less structured, limiting the broader applicability of such rule-based approach.

3.1.3.2 Ontology-Based Methods

Ontology-based approaches, on the other hand, preserve the original structure of the data and could generalize across the semantic information they contain. For instance, Fenz et al. [25] used the security ontology to store data of existing facts, and proposed an ontological mapping of the ISO/IEC 27001 standard. They introduced a generic OntoWorks framework to access, visualize and reason on the ontology database. The framework employs SPARQL queries to check compliance and could generate reports that allow human auditors to trace the decision-making process, ensuring transparency. The required controls specified in ISO/IEC 27001 facilitate unified mapping, but this method may not apply to other standards with different types of compliance requirements. Besides, from the perspective of continuous compliance, Fenz's approach faces challenges in real-time synchronization.

Building on similar semantic principles, Schubert et al. [6] propose an ontology-based automation framework for continuous compliance of airborne software. By creating knowledge graphs to integrate diverse lifecycle artifacts, their system can automatically check compliance and generate reports for human review. Although this method seems more applicable across different artifact types, it still does not fully transcend the limitations of formal verification. The authors admit that the solution remains semi-automatic, as certain objectives, particularly those involving traceability and coverage, remain difficult to automate.

4

Research Methods

4.1 Design Science Research

This thesis takes a Design Science Research (DSR) approach. This approach is well suited to this study, as the aim is to not only describe continuous compliance, but also to design and refine artifacts that can support it in practice. The main artifacts developed in this thesis are a reference model for continuous compliance and a TReqs plugin prototype demonstration that operationalizes selected parts of the model.

The research was conducted through three design iterations. Each iteration included problem investigation, solution design, implementation of the solution, and evaluation of the solution. These activities were not treated as strictly separate phases. Instead, findings from literature reviews, workshops, discussions with supervisors, and prototype development were used iteratively to refine both the conceptual model and the repository-level demonstration.

In this thesis, 'problem investigation' refers to identifying limitations in the current understanding or implementation of continuous compliance. 'Solution design' refers to the process of defining or extending the conceptual structure of an artifact, such as a reference model, taxonomy, rule map or traceability model. Solution implementation involves modifying the design as either a refined diagram or prototype development. Solution evaluation involves assessing the artifact through expert feedback, workshop discussions, comparisons with the reference model and reflections on its coverage and limitations.

4.2 Data Collection

4.2.1 Literature Exploration

In the initial phase of this study, a broad literature exploration was conducted to establish a foundational understanding of continuous compliance and related research areas.

The primary data sources were publications from major software engineering conferences and journals, including ICSE, FSE, RE, TSE and JSS. A ten-year span of publications (up to March 2025) was considered to ensure both relevance and coverage of recent research.

Rather than starting with a narrowly defined query, this phase took an exploratory approach. Based on expert insights, a preliminary concept framework of the continuous compliance process was proposed, comprising four phases:

- Standards Identification
- Compliance Requirements Extraction
- Standards Interpretation
- Compliance Argumentation

Papers were then selected through title screening to focus on those offering solutions to these phases, as well as other relevant content that had not been anticipated.

Based on this framework, papers related not only to compliance, but also to requirements extraction and knowledge representation, were included in the initial pool.

As a result, the initial dataset covered a wide range of topics. In addition to compliance and regulatory analysis, papers also related to the following:

- Requirements extraction and modeling
- Natural language processing (NLP) for requirements engineering
- Knowledge representation (e.g. ontology-based approaches)
- Emerging techniques such as large language models (LLMs) for compliance support

This broad scope led to the compilation of a diverse set of candidate papers across venues. The purpose of this step was to capture potentially relevant approaches before refining the research boundary.

4.2.2 Literature Review

The literature was collected through a structured search in IEEE Xplore, which was selected as the primary database due to its strong coverage in software engineering. The search was conducted using the exact phrase "continuous compliance" within the title and abstract fields in order to focus on studies that explicitly address the concept as a central research topic. An eight-year publication range was applied. This search strategy initially returned 33 papers. A preliminary screening was then performed based on titles and abstracts to exclude papers that mentioned continuous compliance only marginally or used the term in domains other than software engineering. This resulted in a subset of 16 papers, which are the green entries in form 4.1.

The objective of the literature review was to identify differences in the interpretation and implementation of continuous compliance in different studies. Particular attention was given to papers proposing concrete frameworks or practices, as these offer insight into the continuous compliance workflow.

	A	B	C	D	E	F
1	Document Title	Authors	Author Aff	Publicatio	Date Add	Publicatio
2	Continuous Compliance: Experiences, Challenges, and Opportunities	R. Filepp; (	IBM T.J. W	2018 IEEE	#######	2018
3	Continuous Compliance model for Hybrid Multi-Cloud through Self-Service Orchestrator	R. Rompic	School of	2020 Inter	#######	2020
4	Continuous Compliance	M. Kellogg	University	2020 35th	#######	2020
5	A Secure Framework for Continuous Compliance across Heterogeneous Policy Validation Points	T. Yanaga	IBM Resea	2024 IEEE	#######	2024
6	Characterizing Software Architectural Metrics for Continuous Compliance in the Automotive Domain	D. Amalfit	DIETI, Uni	2024 IEEE	24-Jul-24	2024
7	Continuous Compliance in the Automotive Industry	T. Santilli;	Gran Sass	IEEE Softw	4-Jun-24	2024
8	Continuous compliance to ensure strong cybersecurity posture within digital transformation in smart cities	A. Alroma	College of	6th Smart	#######	2022
9	AI-Enhanced Intelligent Internal Audit Automation Systems for Continuous Compliance Monitoring and Multi-Layer Risk Mitigation	L. A. Mutt	College of	2025 3rd I	#######	2025
10	Using Boundary Objects for Continuous Compliance in Automotive Development	A. Shenou	McMaster	2024 IEEE	#######	2024
11	An Ontology-Based Automation Framework for Continuous Compliance of Airborne Software	T. Schube	German A	2025 AIAA	#######	2025
12	Continuous Regulatory Compliance Through Real-Time Data Lineage Analysis and Security Policy Orchestration	U. Pandya	, Chicago,	2026 IEEE	#######	2026
13	Towards Compliance Management Automation thru Ontology mapping of Requirements to Activities and Controls	D. C. Chen	College of	2018 Cybe	#######	2018
14	A Network Access Control Scheme for IoT Terminals Based on Active Scanning	W. Xie; J. Y	Informatic	2022 Inter	#######	2022
15	AI-Assisted Data Governance with Data Mesh Manager	A. Wider;	Hochschul	2025 IEEE	#######	2025
16	Neuro-Symbolic Compliance-as-Code: Toward Explainable, Adaptive, and Self-Evolving Governance Systems	S. Nigam;	Expert Ap	2025 IEEE	#######	2025
17	AI-Driven Framework for Automating Infrastructure Provisioning and Compliance Validation in Cloud-Native Environments	H. A. Igwe	BellaTech	2025 IEEE	#######	2025
18	Continuous Verification of Network Security Compliance	C. Lorenz;	Departme	IEEE Trans	9-Jun-22	2022
19	Compliance-as-Code for Cybersecurity Automation in Hybrid Cloud	V. Agarwa	IBM Resea	2022 IEEE	#######	2022
20	Object-Process Model-Based Operational Viewpoint Specification for Aerospace Architectures	Y. Mordec	Departme	2020 IEEE	#######	2020
21	Compliance Monitoring on Process Event Streams from Multiple Sources	P. Koenig;	Faculty of	2019 Inter	#######	2019
22	RANVDS: Client-Side Detection of Insecure RAN Configurations in Operational 2G鈥?G Networks	L. L. da Ro	Departme	IEEE Oper	#######	2026
23	Growlithe: A Developer-Centric Compliance Tool for Serverless Applications	P. Gupta; A	The Unive	2025 IEEE	#######	2025
24	AI - Integrated DevSecOps for Scalable, Secure, and Modern Cloud Architectures	R. K. Mahi	Solutions A	2026 Inter	#######	2026
25	Quantum-Driven Cryptographic Framework for Securing Healthcare Data	N. H; G. N.	Computer	2025 3rd I	#######	2025
26	Cloud Governance, Risk, and Compliance (GRC)	J. Edwards	Birkbeck, U	Cloud Security	Funda	2026
27	Kubernetes-Driven Network Security for Distributed ACL Management	T. Gupta	NA	2024 8th C	#######	2024
28	Dynamic Automated Red Team Tool (DARTT): A Comprehensive Framework for System, Network and Data Security Auditing	S. Choudh	Faculty of	2025 IEEE	#######	2025
29	Security Automation and DevSecOps	J. Edwards	Birkbeck, U	Cloud Security	Funda	2026
30	Security Architecture as a Service (SAaaS) for Visual Modeling and Trade-off Analysis of Security, Availability, and Maintainability	R. K. Gaut	School of	2025 Cybe	#######	2025
31	Drift Detection and Self-Healing Infrastructure with Ansible and GitOps	N. Seshag	NA; NA; N	2025 Inter	#######	2025
32	Intelligent Cloud Security Posture Optimization Using AI-Driven Risk Analytics	A. R. R; B.	CSE Depa	2026 5th I	#######	2026
33	Importance Of Gap Analysis: A Case Study of Webtrust Audit In Government Body Certificate Authority	M. I. Fathu	Computer	2025 13th	#######	2025
34	Trustless Compliance: On-Device LLM Orchestration for Zero-Data-Exposure Security	S. Salisu; C	CTO, Bilic,	2025 12th	#######	2025

Figure 4.1: Literature search and screening results

Before the literature search, we have created a diagram of the conceptual structure of the continuous compliance process. This was used as a reference and lens during the literature review, supporting comparison across studies. This enabled us to systematically identify relevant aspects of each paper, such as compliance targets, compliance check triggers and enforcement mechanisms, and summarize these aspects for further analysis.

The collected literature therefore forms a targeted dataset for analyzing variations in continuous compliance practices and supporting the evolvement of the reference model.

4.2.3 Workshops

4.2.3.1 1st Software Center Project Workshop

After the literature review, a workshop was conducted as part of the second design iteration. The workshop was conducted as part of Software Center Project 66, *Compliance as Code*. The project involves academic researchers and industrial partners interested in continuous compliance, compliance automation, software engineering, and regulated software-intensive systems. Therefore, the workshop participants were not recruited as a general sample, but were members or stakeholders of the project with relevant expertise and practical experience.

The purpose of the workshop was to discuss the preliminary taxonomy and reference model developed from the literature review, and to collect feedback on their relevance for industrial continuous compliance settings.

The discussion followed a semi-structured format. First, the preliminary taxonomy and reference model were presented to establish a shared basis for discussion. The

subsequent discussion focused on three guiding questions: what level of automation is meaningful for continuous compliance, which artifacts and evidence should be prioritized, and how TReqs could support continuous compliance through repository-level traceability.

Table 4.1: Workshop participants

	Role / expertise	Domain
P1	Requirements engineering researcher	University
P2	Software/system architect	Telecom
P3	Senior technical leader in automotive safety/DevOps	Automotive
P4	DevOps / software platform /service operation	Automotive / digital services
P5	Cybersecurity compliance for embedded products	Embedded system / cybersecurity
P6	Software/compliance practitioner	Defense / dual use

The workshop included participants from academia and industry sectors related to the Software Center Project 66, including telecom, automotive, digital services, embedded systems, and cybersecurity. All participants have over 20 years' working experience. These domains are relevant to the thesis because they involve software-intensive systems, regulated development contexts, and practical compliance-related challenges. Table 4.1 summarizes the participants in anonymized form.

The workshop was recorded and automatically transcribed. The transcript was used as qualitative data for the analysis. The data captures both direct feedback on the preliminary model and broader discussion about challenges, opportunities, and practical constraints of continuous compliance.

4.2.3.2 2nd Software Center Project Workshop

A second workshop was conducted in Iteration 3 within Software Center Project 66, *Compliance as Code*. The purpose is to review the outcomes of Iteration 2, including the refined reference model and the initial demonstration. The workshop was organized as an artifact review session rather than as an exploratory data collection activity.

During the workshop, a summary of the previous workshop and the current reference model were presented, as well as an initial rule-based checking design. The discussion focused on whether the model sufficiently support continuous compliance pre-merge. Participants discussed the need to extend the model towards DevOps workflows.

The workshop was recorded and transcribed. However, as the discussion took place offline and individual speakers could not always be distinguished in the transcript, it was not analyzed using the same detailed thematic coding process as in Iteration 2. Instead, it was used to provide design feedback for refining the reference model.

4.3 Data Analysis

4.3.1 Literature Exploration

The analysis in this phase followed an iterative narrowing approach.

Firstly, the analysis began with a screening of the titles and abstracts of the initially collected papers. During this step, it became evident that some of the literature was not directly relevant to software compliance.

The analysis was guided by the preliminary conceptual framework created prior to the exploration. In particular, ontology-based and model-driven approaches were considered as initial categories of potential solutions. However, when this classification was applied to the selected papers, it became clear that many did not fit neatly into these categories, indicating the limitations of the initial taxonomy, which was expected to be addressed during the development of the reference model.

As the analysis progressed, recurring methodological themes emerged, including NLP-based techniques, ontology-based approaches, and the use of LLMs. To balance coverage and feasibility, a representative sampling strategy was adopted, so that only one or two representative papers per category were selected for in-depth analysis. This decision was supported by the observation of thematic saturation, where similar approaches did not significantly improve understanding.

A key outcome of this phase was a shift in analytical focus. While the initial intention had been to identify different technical solutions, such as ontology-based or model-driven methods, a different and more constructive perspective emerged from the literature. Some papers provide explicit definitions of continuous compliance, while many others describe compliance processes with clearly defined phases or steps, some of which were illustrated by visual models. These results are directly relevant to the development of the reference model and form the primary basis for creating the preliminary continuous compliance process diagram. The details are discussed in section 5.1.1

4.3.2 Literature Review

The aim of analyzing these literature was to derive a taxonomy of existing continuous compliance solutions, and to finally create a reference model that addresses all the variants and thereby helping people to better understand continuous compliance approaches.

The literature review was guided by a set of anticipated dimensions derived from our knowledge of continuous compliance, with a particular focus on the differences between the proposed solutions. Initial dimensions included aspects such as the scope of compliance, the level of automation and the types of method (e.g. process-oriented, model-based or ontology-based), as well as integration with the software lifecycle.

During the reading process, observations were recorded in an open and exploratory

manner. These include both concepts and implementation concerns. As more papers were examined, recurring patterns across studies began to emerge. Elements that appeared in multiple papers were gradually consolidated into higher-level categories. Many of these are not part of the initial analytical dimensions. Through this iterative process, the initial loosely defined dimensions evolved into a more structured and complete taxonomy.

4.3.3 Thematic Analysis of the 1st Workshop

The workshop transcript was analyzed using thematic analysis following the approach proposed by Braun and Clarke [26].

Table 4.2: Representative quotes from the workshop thematic analysis

Theme	**Representative quote**	**Source**	**Code**	**Type**
Distributed compliance contexts	"We are designing monitors that look into runtime data in the cars to check compliance."	P3	Runtime Data as Compliance Evidence	Industry Practice
Level of automation	"One of the completely obvious things to do is to automate what can be automated."	P5	Automation Support	Solution Suggestion
Compliance as development context	"We are used to thinking about compliance as something you do and check afterwards, but it almost comes as a starting point."	P4	Compliance From the Start	Context
Heterogeneous Artifacts	"The artifacts needed are decided by what we want as evidence."	P3	Artifacts Diversity	Problem
Exemptions and lifecycle governance	"Those exemptions should of course also be version controlled and managed as part of your product."	P2	Traceable Exemption Lists as Audit Materials	Solution Suggestion
Risk and confidence	"The confidence level we need is of course associated with the risk."	P2	Confidence Level	Solution Suggestion

The transcript was first segmented into meaning units based on semantic coherence. Each meaning unit represented a distinct idea, concern, example, or suggestion

expressed by a participant. These units, referred to as quotes in the analysis spreadsheet, were used as the basic units of analysis.

The analysis followed an inductive coding approach, allowing themes to emerge from the data rather than applying a fixed coding scheme in advance. Each quote was reviewed and assigned a short descriptive phrase summarizing its central meaning. These phrases functioned as initial codes and captured the main point expressed in the quote.

In addition to the inductive codes, each quote was categorized according to its functional role in the discussion, such as context, challenge, problem, opportunity, industry practice, or solution suggestion. This additional label was used to support later interpretation of how each quote could inform the development of the reference model.

Table 4.2 provides representative quotes for each theme identified in the workshop analysis. The coding process was conducted in Microsoft Excel. For each quote, the spreadsheet recorded the transcript segment, the speaker, the initial code, the functional type, and notes on potential implications for the reference model. Quotes with similar codes were then compared and grouped into broader themes. Through this iterative process, recurring patterns across participants' statements were identified and used to refine the taxonomy and reference model.

4.3.4 Supervisor-Guided Synthesis

Following the thematic analysis of the workshop transcript, a follow-up synthesis meeting was held with the supervisor. This meeting aimed to interpret the workshop themes from a requirements engineering perspective and translate them into concrete implications for the reference model.

This synthesis was based on the coded workshop data, the thematic analysis spreadsheet and the preliminary reference model. During the meeting, the identified themes were reviewed and discussed in relation to the existing model elements. Particular attention was given to whether the themes indicated missing concepts, unclear relationships or parts of the continuous compliance process that were not sufficiently represented.

We examined how the themes were related to existing model concepts such as compliance requirements, compliance checks, compliance evidence, compliance arguments and DevOps activities. We then discussed which workshop themes should be reflected as new or revised elements in the reference model.

The outcome of the synthesis meeting was a set of refinement decisions for the reference model. These included making compliance checks more central, distinguishing between different types and degrees of checks, and representing exemption lists and lifecycle governance. These decisions guided the redesign of the reference model in the second iteration.

4.4 Design Science Iterations

4.4.1 Iteration 1

The first iteration aimed to investigate how continuous compliance is described and supported in existing literature, and to derive an initial conceptual artifact from these findings. Relevant studies were analyzed with particular attention to the activities, workflows, and artifacts used to support continuous compliance across different contexts. The initial design artifact was first represented as a UML activity diagram. It was then refined through supervisor-guided discussions from a requirements engineering perspective.

The outcome of this iteration was a refined process diagram that served as the first conceptual artifact of the thesis. The model provided an initial answer to RQ1 by identifying the main activities and artifacts involved in collecting and maintaining compliance evidence. Based on the model, we also created an initial traceability design for the later TReqs-based demonstration.

4.4.2 Iteration 2

The second iteration addressed both the conceptual development of the reference model and the initial implementation of the solution. While the first iteration established an initial reference model and a preliminary TTIM, the second iteration aimed to build on this work by analyzing variations in continuous compliance approaches and exploring how these concepts could be represented in the prototype.

Firstly, a structured literature analysis was conducted to derive a taxonomy of continuous compliance approaches. This taxonomy was then used to extend the reference model.

Secondly, the preliminary taxonomy and reference model were discussed at a workshop. The discussion focused on automation levels, relevant compliance artifacts and evidence, and the potential role of the TReqs plugin in supporting continuous compliance.

Thirdly, thematic analysis was used to analyze the workshop transcript. The resulting themes were discussed further and translated into implications for the reference model. In parallel, the prototype was further developed by instantiating the TTIM with ISO 26262 Part 8 Clauses 7 and 8, defining initial rule-based checks and generating an initial compliance report. The feedback from the workshop and the prototype development provide answers to RQ2.

4.4.3 Iteration 3

The third iteration focused on refining the reference model to allow for stronger DevOps integration and to complete the prototype plugin. Although Iteration 2 extended the model to include runtime data, it still did not adequately explain the relationship between continuous compliance and the broader DevOps lifecycle.

Based on feedback from the 2nd Software Center workshop, the reference model was refined to create a DevOps-integrated version combining the activity-centered view from Iteration 1 and the artifact-centered view from Iteration 2. In parallel, the prototype was extended to include more advanced compliance checking. This included the implementation of an extended rule map, initial lifecycle governance, and change review.

4.5 Threats to Validity

Construct Validity

One potential threat to construct validity is the definition of continuous compliance itself. This term is interpreted differently throughout the literature and there is no single, established reference model. To mitigate this issue, this thesis employs multiple approaches combining literature analysis, workshop feedback, supervisor-guided discussions and prototype development. Nevertheless, the resulting model still reflects the interpretation developed within this study and may not encompass all possible perspectives on continuous compliance.

Internal Validity

Some of the findings are based on qualitative feedback from workshops, rather than on objective measurements. While this provides useful insights into practical relevance, it may also introduce subjectivity. In particular, evaluation of the reference model depends on expert interpretation and design reasoning rather than a controlled empirical comparison.

The prototype evaluation is also limited. It demonstrates selected concepts, such as traceability, rule-based checks, lifecycle status, change reviews and report generation, but does not measure how much effort is actually reduced in a real project.

External Validity

The prototype is demonstrated using selected clauses from ISO 26262, Part 8. This restricts the applicability of the results to other standards, domains and compliance processes.

Furthermore, implementation depends on the current capabilities of TReqs and git version control. Different organizations may use different tools, repository structures and document formats, which could impact the feasibility of applying the same approach.

Reliability

The literature analysis, workshop interpretation, and rule extraction involved researcher judgment. Although the analysis was documented through taxonomies, spreadsheets and diagrams, another researcher might classify some concepts differently.

The prototype rules were also manually created. This means that the correctness of the checking results depends on how accurately the rules reflect the interpreted

compliance requirements. Future work should evaluate the approach with additional standards, independent reviewers, and industrial projects to improve reliability.

5

Findings

5.1 Iteration 1

The first iteration aimed to develop an initial reference model for continuous compliance based on domain knowledge and selected literature. The purpose was to transform fragmented and context-specific descriptions of continuous compliance into a more structured process representation.

This iteration began with an investigation of existing studies and their workflow descriptions. Based on recurring activities identified in the literature, an initial UML activity diagram was created as the first design artifact. The diagram was then refined through supervisor-guided discussions from a requirements engineering perspective.

The resulting process diagram represents the main artifact of this iteration. It provides a clearer conceptual basis for the following iteration, where the reference model is further extended and evaluated through broader workshop activities.

5.1.1 Problem Investigation

The aim of the problem investigation phase is to understand how continuous compliance is currently interpreted and implemented in existing studies.

Based on domain knowledge, an initial conceptual framework of the continuous compliance process was introduced as the starting point. This initial structure included four core activities:

- Standards Identification
- Compliance Requirements Extraction
- Standards Interpretation
- Compliance Argumentation

While this early framework does not capture all critical aspects of continuous compliance, it does guide literature exploration. A key observation from the literature analysis is that many works construct their own workflow to support continuous compliance. In particular, studies focusing on process integration (e.g. within agile development) often include explicit workflow diagrams, while other approaches, such

as those artifact-oriented solutions, also implicitly or explicitly describe compliance activities in a staged manner.

However, these representations are either incomplete or too context-specific. Process models are often tightly coupled to a particular approach, which may involve a specific toolchain, standard or domain (e.g. automotive systems). Consequently, while there are clear similarities across studies, such as recurring phases of requirement extraction, verification, and evidence collection, these models are not directly comparable and lack generalizability.

This fragmentation makes it difficult to systematically understand how continuous compliance is implemented across different contexts. It also highlights the absence of a unified reference model that abstracts from solution-specific details while preserving essential process characteristics. Therefore, we derived insights from the workflow models in [17], [18] and created the first version of the reference model as a UML diagram.

5.1.2 Solution Design

Based on the findings of the problem investigation phase, the first version of the reference model was designed as a UML activity diagram. This initial model was not intended to provide a complete or final representation of continuous compliance, but to make the process explicit and available for extension.

As shown in Figure 5.1, the model starts with a sequence of compliance preparation activities. The process begins with identifying relevant standards, followed by identifying relevant statements within those standards and extracting compliance requirements from them. These requirements are then mapped to the system description. Artifacts involved in this flow are treated boundary objects because it connects regulatory information with system-specific design. Automated checks can be defined on the basis of the mapped requirements using KPIs and traceability information.

The lower part of the diagram illustrates the continuous development loop. Once the checks have been defined, developers can modify the code and create a merge request. Automated checks are then performed to determine whether the proposed change complies with the defined checks. If the checks are successful, the change can be merged into the main branch. If the checks fail, the process model distinguishes between several possible causes. A failure may indicate that the change violates an already defined policy, in which case the code needs to be revised. Alternatively, the failure may reveal that the current checks do not sufficiently cover the relevant policy, requiring a return to the activity of defining or refining automated checks. Finally, if the policy itself has not been properly mapped to the system description, the process returns to the mapping activity.

Continuous compliance solutions need to respond to changes in the environment. In [18], they considered system and standard changes and designed loops for each. Using the branches and jump-backs in the UML diagram, we also attempt to handle continuous compliance maintenance as the context evolves.

The initial UML model provides an initial abstraction of continuous compliance as an iterative process. It combines standards interpretation, requirement elicitation, automated compliance checking and continuous feedback into a single process. This model was then used as the basis for the subsequent development and refinement phase, where its structure, terminology, and process coverage were reviewed.

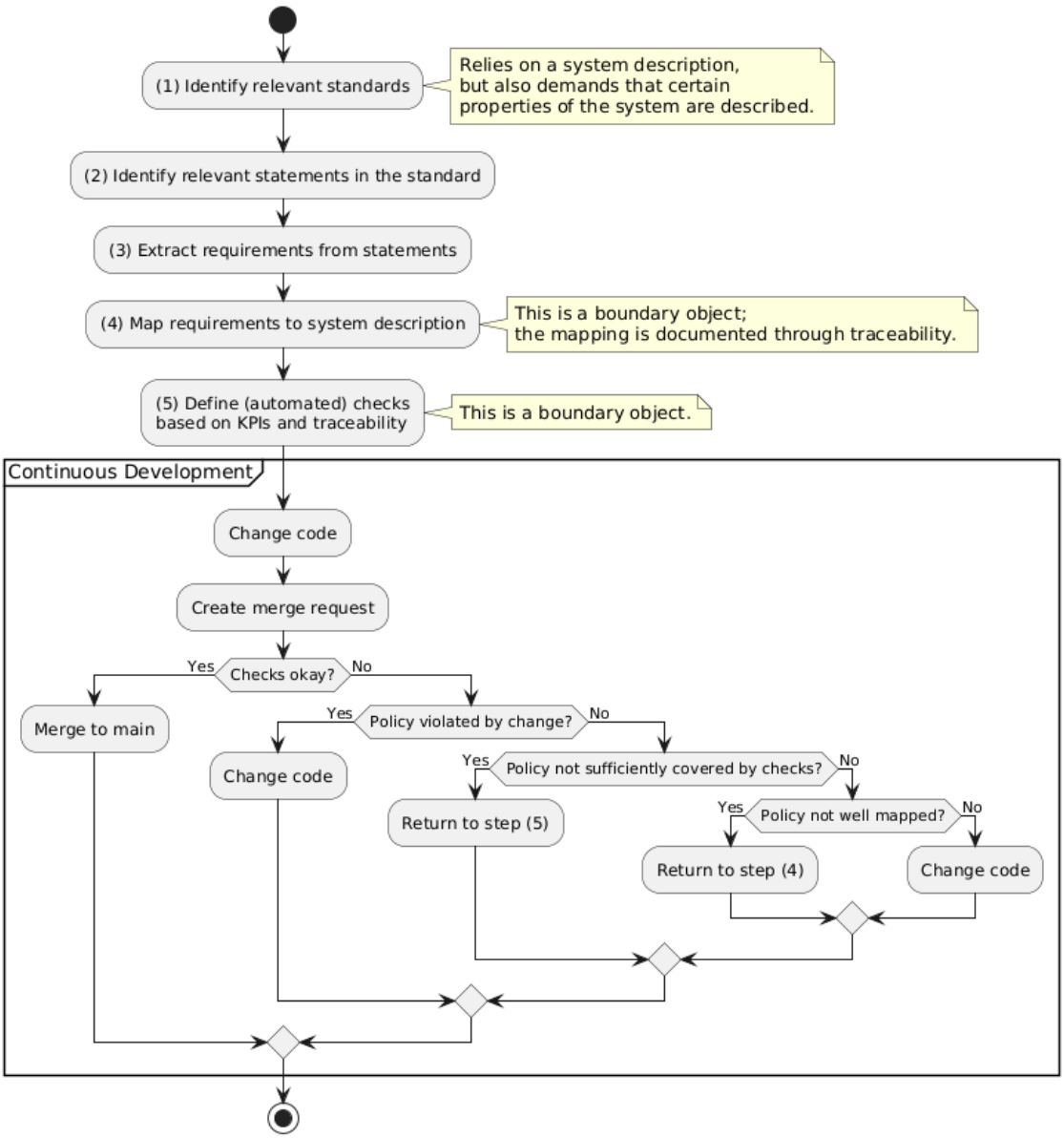


Figure 5.1: The initial UML diagram of the CC workflow

5.1.3 Solution Implementation

Reference Model

The process model was refined based on the initial UML diagram and the opinion of requirement engineering experts. This refinement aimed to improve the clarity,

structure and process coverage of the model without changing its overall direction. In particular, the refinement focused on making the artifacts involved and the different types of activity more explicit.

The initial diagram captured several key elements of the process, including standards identification, requirement extraction, mapping to the system description, automated compliance checks and feedback from continuous development. However, the representation was still limited in terms of clarity and structure. The limitations of the UML diagram restrict the expression of loops. The relationships and divisions between regulatory work, development activities, and assessment activities were not sufficiently explicit. Furthermore, the artifacts involved in the workflow could also be expressed more explicitly.

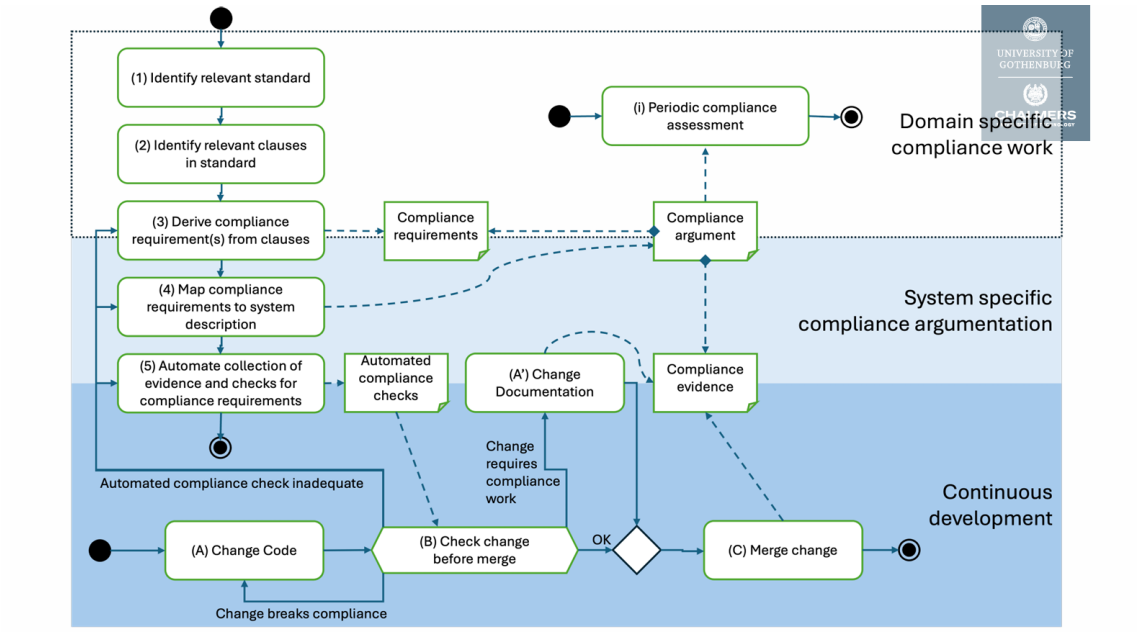


Figure 5.2: A process diagram of the CC workflow

As a result, the diagram was refined into a more structured version, as shown in Figure 5.2. Compared with the initial UML model, the refined diagram makes the different areas of continuous compliance more explicit. The left part of the model represents compliance work, including the identification of relevant standards and clauses, the derivation of compliance requirements, the mapping of these requirements to the system description, and the automation of evidence collection and checks. The lower part represents the continuous development process, where code changes will trigger compliance checks. The upper-right part represents periodic compliance assessment, which is supported by compliance arguments and evidence collected from other activities. This whole model is also divided into three levels: domain-specific work, system-specific compliance argumentation and continuous development.

The refined model also makes the role of artifacts clearer. Compliance requirements, automated compliance checks, compliance evidence, and compliance arguments are

represented as explicit artifacts that connect different activities. This distinction is important because continuous compliance does not only involve a sequence of activities, but also depends on maintaining traceable relationships between these boundary objects.

TReqs-Compliance Demo

An initial TTIM for treqs-compliance was also designed based on the artifact relationships made explicit in the refined process diagram, as shown in 5.3. This model translates selected concepts from the reference model into traceability elements at the repository level that can be used in the demonstration.

```yaml
---
name: T-Reqs Compliance Type and Trace Information Model (TTIM)
version: 0.0.1
description: Defines element types and links that can be used in the treqs-compliance project. S
author: Jinlu Yu, Eric Knauss
types:
  # A specific governing document for our system under development
  - name: standard
  outlinks: []
  # A thing required by a standard
  - name: clause
  outlinks:
  - type: from
    target: standard
    required: 'true'
  - name: compl.requirement
  outlinks:
  - type: derivedFrom
    target: clause
  # An artifact required by a standard
  - name: abstractBoundaryObj
  outlinks:
  - type: derivedFrom
    target: clause
  # An artifact in our system development repo that corresponds to an abstract boundary object
  - name: systemLevelBoundaryObj
  outlinks:
    - type: relatesTo
      target: abstractBoundaryObj
    - type: address
      target: systemLevelBoundaryObj
```

Figure 5.3: TTIM of TReqs Compliance Plugin Demo

The initial TTIM focuses on the relationship between standards, clauses, compliance requirements, abstract boundary objects, and system-level boundary objects. A `standard` represents a governing document, while a `clause` represents a relevant statement within that document. A `compl.requirement` is derived from a clause and represents a compliance requirement that needs to be addressed by the system. The model further distinguishes between `abstractBoundaryObj`, which represents an artifact expected or required by the standard, and `systemLevelBoundaryObj`, which represents a concrete development artifact in the repository that corresponds to such an abstract boundary object.

This initial traceability design reflects the evidence-oriented view of compliance used in this thesis. By connecting standards, clauses, requirements, and work products, the TTIM provides a basic structure for collecting and maintaining compliance evidence through traceability management.

5.1.4 Solution Evaluation

This design of the diagram reflects important assumptions about continuous compliance. First, continuous compliance requires both preparation and continuous maintenance. The early activities, such as identifying standards, extracting requirements, and mapping them to the system description, provide the basis for compliance checking. However, continuous compliance cannot be ensured only once during the initial design phase. It also needs to be maintained during continuous development, for example through automated checks that are executed when code changes are introduced.

Second, as discussed in previous subsection, continuous compliance needs to account for different sources of change. The feedback structure of the initial UML diagram reflects an early attempt to model these sources of change. In the design phase, the model does not only consider code changes during continuous development, but also allows for changes related to policies, standards, or their interpretation.

In the implementation phase, however, the scope of process model was narrowed to be more solution-oriented. The refined diagram therefore represents the feedback loop mainly through code changes. This reflects the practical scope of the solution we want to present, which focused on integrating automated compliance checks into software lifecycle rather than modeling the automated interpretation of standards or requirements elicitation. However, since our goal is to develop a reference model for continuous compliance, rather than to explain one single solution, more comprehensive consideration is needed and the diagram still needs more refinements and extension.

Overall, the evaluation showed that the UML model was useful as a starting point, and the implementation improves in structure and readability. To be specific, the refined diagram provided a clearer separation between regulatory activities, software development and audit activities. However, this refined model also revealed a limitation in scope. This informs the next iteration, in which the model should be developed as a more general reference model rather than a workflow specific to a particular solution.

5.2 Iteration 2

The second iteration focused on extending the initial reference model and the prototype plugin treqs-compliance. While the first iteration established a basic process view of continuous compliance, it did not yet fully capture the different ways in which continuous compliance is interpreted and implemented across studies and practice.

To address this limitation, a structured literature review was conducted to derive a

taxonomy of continuous compliance approaches. The taxonomy was used as an intermediate design artifact. It helped identify which aspects of continuous compliance should be represented in the reference model.

The taxonomy was then discussed and refined through a workshop with industrial participants and a subsequent supervisor-guided analysis of the workshop results. These activities informed the redesign of the reference model in this iteration.

5.2.1 Problem Investigation

The first iteration resulted in an initial reference model that described continuous compliance as a process connecting standards interpretation, requirement mapping, automated checks, compliance assessment, and continuous development. However, the evaluation of the first iteration also showed that the model was still limited in scope. It provided a useful process structure, but did not yet systematically capture the variation of continuous compliance approaches found in the literature.

In particular, existing studies differ in what they treat as the target of compliance, when compliance activities are triggered, which mechanisms are used to enable compliance checking and many other different aspects. This variation is important for RQ1, because the collection and maintenance of compliance evidence depends on the type of artifacts involved, the phase of the software lifecycle in which compliance is checked, and the degree to which compliance activities are automated. Therefore, the second iteration aimed to extend the initial reference model by systematically analyzing these differences across the literature.

To support this goal, a structured literature review was conducted. The review focused on identifying recurring dimensions of continuous compliance solutions. The result of this investigation was a taxonomy of continuous compliance approaches, which served as an intermediate artifact for extending the reference model and preparing the subsequent workshop discussion.

5.2.2 Solution Design

Reference Model

Based on the reviewed literature, a taxonomy of continuous compliance approaches was developed. The purpose of the taxonomy was to identify recurring dimensions that distinguish different interpretations and implementations of continuous compliance. Rather than treating continuous compliance as a single fixed workflow, the taxonomy shows that existing approaches differ in various aspects, as summarized in Table 5.1.

The first dimension relates to the compliance target. Some approaches focus on process-level compliance; typical solutions involve integration with agile model and other model-driven methods. Many papers of this kind emphasize the importance of organizational factors, such as team coordination and compliance awareness. Other approaches focus on software lifecycle artifacts, such as source code, Git metadata, configurations and CI pipeline outputs. Involved artifacts can be heterogeneous and

Table 5.1: Taxonomy dimensions of continuous compliance approaches

Dimension	Categories
Compliance target	process; software lifecycle artifacts
Compliance phase	during development; before audit; review and certification
Enablement solution	process modelling; rule-based approaches; ontology-based approaches
Trigger	commit/PR/compiler; CI pipeline step; iteration/planning milestone; artifact update
Requirement extraction	manual interpretation; linguistic techniques; ML-based methods

scattered, making automation more difficult. This distinction between the two types of method is important because they lead to completely different implementation directions.

The second dimension concerns compliance phases. In different phases, there are different compliance activities. During development, continuous compliance approaches typically focus on enforcing compliance controls, maintaining traceability, and updating artifacts. Traceability management usually concerns the relationships among artifacts, such as tracing code back to requirements. Typical solutions involve using code comments, ontological graph or boundary objects. Before audit, the focus shifts towards preparing compliance evidence, such as verification results or CI outputs. During the review and certification process, the focus is on supporting human review, for example through compliance reports.

The third dimension concerns compliance enablement solutions. The literature includes process modeling approaches, rule-based approaches, and ontology-based approaches. These approaches differ in their degree of automation. Process modeling approaches are often more manual, rule-based approaches can be automatically executed through verification scripts or tools, and ontology-based approaches support semi-automatic reasoning by querying an ontological graph or database. Although formal methods are easy to automate, they are limited to check formally specified data. Existence checks, traditional testing, traceability coverage and other structural properties can be easily verified using formal methods. However, due to artifact heterogeneity, the scalability of such automation is limited.

The taxonomy also captures different triggers for compliance check, including commits, pull requests, compiler events, CI pipeline steps, iteration, planning milestones, and artifact updates. Finally, the literature shows different approaches to compliance requirement extraction, ranging from manual interpretation to linguistic techniques and machine-learning-based methods.

TReqs-Compliance Demo

Compliance Target The taxonomy showed that continuous compliance approaches may target either development processes or software lifecycle artifacts. In this the-

sis, the solution focuses on software lifecycle artifacts because we intend to use a PRITM tool which is closely integrated with the system and supports compliance through its traceability management capabilities. Developing this demo could help us answer RQ2.

Compliance Check Types As the aim was not to automate all forms of compliance reasoning, the design concentrated on compliance requirements that could be converted into relatively explicit, verifiable rules. Two check types were selected in particular: existence checks and content checks. These were considered suitable first steps because they address structural aspects of compliance evidence, which can be checked with limited semantic interpretation.

The existence check was designed as a traceability check. It verifies whether an artifact required by a standard is represented in the repository and linked to the corresponding element.

The traceability design distinguishes between two types of relationship. The first connects abstract boundary objects to system-level boundary objects. This relationship represents the mapping from a standard-level artifact expectation to its concrete realization. For instance, the requirement for configuration management plan could correspond to a specific plan document. The second connects system-level boundary objects to other system-level boundary objects. This relationship illustrates dependencies between concrete work products, for example a change request should have a corresponding analysis.

The content check is a lightweight structural check of the contents of a work product. In its simplest form, the content check can be reduced to the existence check of required fields or sections. For instance, a certain ISO 26262 Part 8 requirement specify that the document must contain specific information, including configuration, date, reason, and description. Rather than attempting to automatically interpret the full semantic correctness of the artifact, this iteration checks whether the expected fields are present.

ISO 26262 offers a clear and readable structure where the requirements for content are normally outlined in the *Requirements and recommendations* section, while the abstract boundary objects are specified in the *Work Products* section.

Compliance Enablement Mechanisms The taxonomy also distinguished between different compliance enablement mechanisms. Among these, the rule-based approach was selected. This was motivated by the goal of supporting lightweight automation over repository artifacts.

To express these checks, we decide to introduce a compliance rule map. The rule map defines checkable rules for each boundary object type. Each rule may specify required fields, traceability constraints, required child artifacts, among other things. In this way, requirements extracted from the standard are translated into structured rules that can later be checked against repository artifacts.

Overall, this design starts by focusing on a limited subset of compliance requirements

that can be easily automated. In particular, it targets structural requirements that can be checked through traceability links or the existence of required fields. More complex checks are therefore left for later refinement.

5.2.3 Solution Implementation

Reference Model

The preliminary taxonomy and reference model were used as input for the workshop. During the workshop, the taxonomy was first presented to participants to summarize the main variations identified in the literature, including compliance targets, compliance phases, enablement mechanisms, triggers, and requirement extraction methods. The participants were then invited to discuss the practical relevance and limitations of these dimensions.

The discussion focused on three guiding questions. First, participants discussed what level of automation is appropriate for continuous compliance and where human judgment remains necessary. Second, they discussed which artifacts and evidence are most important for compliance support. Third, they discussed how TReqs could support continuous compliance through traceability between standards, requirements, work products, checks, and evidence.

After the workshop, with the consent of the participants, the transcript was analyzed using thematic analysis. The resulting themes were summarized in a spreadsheet and then discussed in a follow-up meeting with the supervisor. In this meeting, the workshop themes were mapped to possible changes in the reference model. This led to the redesign of the model.

TReqs-Compliance Demo

TReqs Elements Annotation In the previous iteration, an initial TTIM was designed to incorporate selected concepts from the reference model. In the second iteration, this traceability design was instantiated in a lightweight demonstration.

As a first step, ISO 26262 Part 8 was converted into a markdown representation to make the standard easier to process within a repository-based environment. The demonstration focused on Clauses 7 and 8, which deal with configuration and change management. Based on the TTIM, the relevant elements were then tagged and linked in TReqs.

These included the standard itself, clauses, and the abstract boundary objects and corresponding system-level boundary objects required by the standard.

To visualize the resulting traceability structure, the command `treqs list -plantuml -outlinks` was used to generate a UML graph. The resulting graph illustrates how the tagged elements are connected through trace links.

To see the results more clearly, Fig. 5.4 crops part of the UML graph. It shows the generated traceability graph for the instantiated example based on ISO 26262 Part 8, Clauses 8. The figure illustrates the connection between standards and clauses,

the derivation of the need for artifacts from clauses, and the link between abstract boundary objects and the corresponding work products represented in the repository.

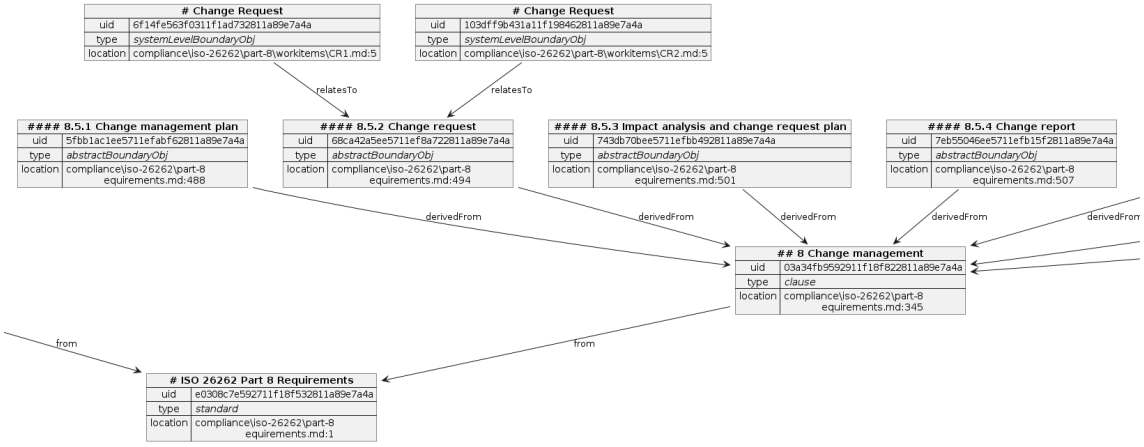


Figure 5.4: The UML graph of the elements traceability structure

Compliance Rule Map A compliance rule map, as shown in Fig. 5.5, was created to define the initial check rules. The rule map included selected boundary object types, such as `configManagementPlan`. For each document type, the rule map defined required fields and, where relevant, additional constraints such as required child artifacts.

```
COMPLIANCE_RULES = {
    "configManagementPlan": {
        "required_fields": ["baseline", "configuration_management_strategy"],
        "description": "ISO 26262-8 Clause 7.5.1"
    },
    "changeRequest": {
        "required_fields": ["date", "reason", "description", "config"],
        "min_links": 0,
        "description": "ISO 26262-8 Clause 8.5.2",
        "required_children": ["changeRequestAnalysis"]
    },
    "changeRequestAnalysis": {
        "requestReview": True,
        "required_fields": ["type", "work_products","responsibility", "impact","schedule"],
        "description": "ISO 26262-8 Clause 8.5.3",
    }
}
```

Figure 5.5: The compliance rule map in iteration 2

The rule map separates different kinds of compliance constraints. For example, the field `required_fields` represents content that must be present in a system-level boundary object. In this iteration, this is checked by matching the titles and YAML metadata in the markdown file. The field `min_links` represents whether an abstract boundary object must be linked to concrete system-level boundary objects. A value

of `0` means that a concrete instance is not required in every case. This is different from `required_children`, which represents dependencies between system-level boundary objects, such as the relation between a change request and its corresponding change request analysis. Finally, `requestReview` is used for cases where a requirement cannot be fully checked automatically and should instead trigger human inspection.

Check Logic Implementation In addition to defining the rule map, this iteration implemented initial content check logic. The implementation focused on the `required_fields` part of the rule map. For each system-level boundary object selected, the checker reads the corresponding markdown file to see if the required fields are present. Currently, fields can be detected in either the YAML metadata block or the headings of the markdown file. The limitation is one document may contain multiple boundary objects. Therefore, we could improve the logic by checking the YAML code block inside the TReqs elements instead.

The rule map is not directly connected to system-level boundary objects through explicit TReqs trace links. Instead, the implementation uses TReqs element attributes to associate artifacts with rules. For Treqs elements we can define a `document` attribute, such as `document="changeRequest"`, which identifies the artifact type. The checker uses this attribute to find the corresponding rule entry in the rule map, after which it applies the checks to the file. An example of change request file is shown in Fig. 5.6.

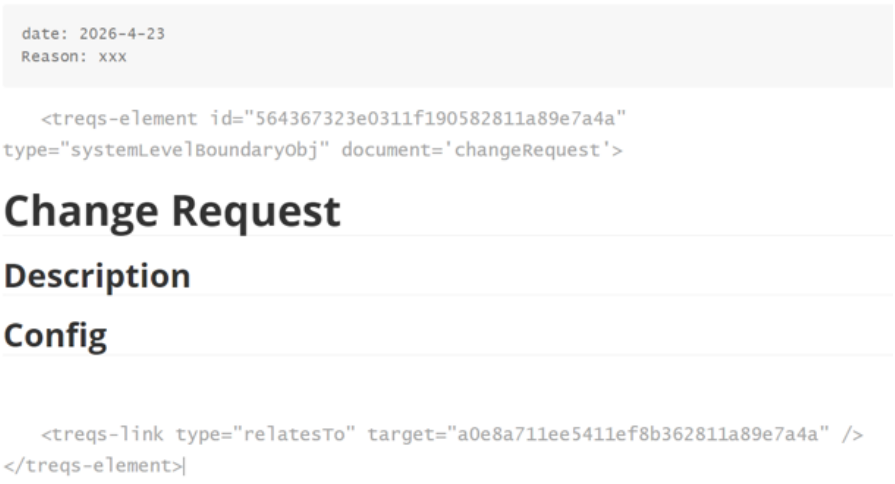


Figure 5.6: An example of annotated work product

The check results were then summarized in an initial compliance report. This report lists each compliance requirement, the relevant work products, the status of the check, and a brief explanation of the result. For example, a requirement is marked as green if the expected artifact is linked and the required fields are found. It is marked as red when there is no corresponding system implementation or when required fields are missing. Fig. 5.7 shows an excerpt of the generated report.

Requirement ID	Requirement	Linked System BOs	Status	Notice
a0e8a711ee5411ef8b362811a89e7a4a	#### 7.5.1 Configuration management plan	564367323e0311f190582811a89e7a4a	GREEN	OK
5fbb1ac1ee5711efabf62811a89e7a4a	#### 8.5.1 Change management plan	-	RED	No system implementation
68ca42a5ee5711ef8a722811a89e7a4a	#### 8.5.2 Change request	6f14fe563f0311f1ad732811a89e7a4a, 103dff9b431a11f198462811a89e7a4a	RED	OK | Missing fields: config

Figure 5.7: The initial compliance report

5.2.4 Solution Evaluation

Reference Model

The second iteration was evaluated through two steps. First, the workshop transcript was analyzed using thematic analysis to identify recurring concerns, challenges, and suggestions from industrial participants. Second, the workshop results were discussed in a follow-up meeting with the supervisor, where the themes were further synthesized and translated into implications for the reference model.

Workshop feedback. The thematic analysis showed that the literature-based taxonomy was useful for structuring the discussion, but also revealed several aspects that were not sufficiently represented in the previous reference model. Table 5.2 summarizes the main themes identified from the workshop and their implications for the model.

Supervisor-guided synthesis. After the thematic analysis, the identified themes were discussed in a supervisor-guided synthesis meeting. The purpose of this meeting was to translate the workshop feedback into concrete changes to the reference model. The discussion focused on how the model should be extended so that it could better represent both the conceptual structure of continuous compliance and the practical concerns raised by industrial participants.

Several refinement points were identified:

- **Compliance checks should be modeled as lifecycle objects.** The workshop discussion showed that checks are not static. They can be introduced, refined, become gating or non-gating, and be retired over time. Therefore, the model should represent compliance checks as artifacts that require lifecycle governance.
- **Exemptions should be explicitly represented.** The discussion highlighted that non-compliance does not always result in an immediate halt to the development process. Any temporary exemptions, accepted risks or justified deviations must be recorded and version-controlled for consideration during later audits.
- **Checks should be differentiated by type and automation level.** The synthesis distinguished between four types of check: existence checks, KPI-based checks, content checks, and context checks. These checks differ in how easily they can be automated. While existence checks can be fully automated, context checks may require manual inspection or AI-supported assistance.

Table 5.2: Main themes identified from the workshop analysis

Theme	Main observation	Implication for the reference model
Distributed compliance contexts	Compliance activities do not only happen during development. Participants discussed development-time, release-time, deployment-time, and runtime compliance. In some cases, products are operated by third parties, which limits the developer's control over operational data.	The model should not restrict compliance evidence to repository or CI artifacts. It should also allow evidence to originate from deployment and runtime contexts.
Levels of automation	Automation can support simple checks, such as artifact existence, KPI thresholds, or traceability coverage. However, semantic and contextual checks often still require human judgment or AI-supported interpretation.	The model should distinguish different types of checks, including existence checks, KPI-based checks, content checks, and context checks. It should also represent different degrees of automation.
Heterogeneous artifacts	Compliance-relevant work products can include source code, configurations, CI results, runtime data, use-case executions, statistics, documentation, and manually produced analyses.	Compliance model should consider and handle a heterogeneous artifact category, rather than as a single fixed artifact type.
Compliance as development context	Compliance is not only something checked after development. Especially in AI-assisted or specification-driven development, compliance constraints may become part of the input context for development activities.	The model should include compliance-by-design aspects, where compliance requirements and checks guide system changes before or during development.
Exemptions and lifecycle governance	A failed compliance check does not always stop development. Organizations may temporarily accept a violation, document an exemption, or make a risk-based decision. These decisions must be maintained and audited.	The model should include exemption lists, lifecycle governance, and mechanisms for maintaining exceptions over time.
Risk and confidence	Compliance checks differ from ordinary software tests because failures may not be detected through later technical feedback, but by auditors or regulators. Therefore, the required confidence level depends on risk and consequence.	The model should represent that the degree of automation should be interpreted together with confidence and prioritization.

- **Compliance evidence should include runtime data.** Feedback from the workshop showed that compliance evidence may come not only from repository artifacts or CI pipelines, but also from runtime monitoring, operational data, triggering conditions, use cases, and behavioral statistics.
- **Compliance reports and assessment reports should be separated.** The discussion clarified that collected evidence may be organized into compliance reports, while assessment activities may produce assessment reports. This distinction helps separate and link evidence collection from compliance assessment.
- **Standard changes should be connected to impact analysis.** As continuous compliance must react to changes in standards or regulations, the refined model should incorporate standard changes and change impact analysis to inform updates to requirements.

Refinement of the reference model. Based on the feedback from the workshop and the supervisor-guided synthesis, the reference model was redrawn, as shown in Figure 5.8. To improve readability and clarity, this version shifts the focus from activity-centered to artifact-centered. Compared with the previous version, the refined model provides a more detailed representation of the connections between standards, requirements, the system, checks, evidence and audit documents.

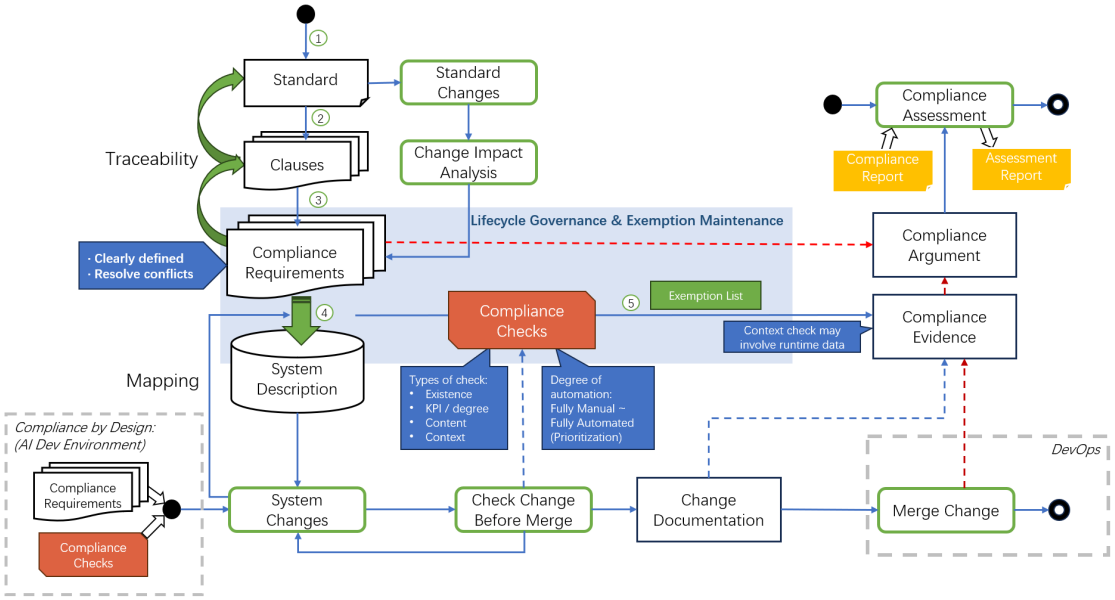


Figure 5.8: An artifact-flow diagram of the CC workflow

The left side of the model illustrates the flow of standards to requirements. It introduces standard changes and change impact analysis. This addresses the limitation of the previous model, where continuous compliance was mainly intervened by code changes. In the refined model, changes in standards can trigger impact analysis and lead to updates of compliance requirements and subsequent steps.

Table 5.3: Mapping from taxonomy dimensions to the reference model

Taxonomy dimension	**Representation in the reference model**
Compliance target	The model contains both activity flow and artifact flow. Process-oriented targets are reflected through compliance and development activities, while artifact-oriented targets are reflected through requirements, system description and other compliance-relevant artifacts.
Compliance phase	The model covers the main phases of a complete continuous compliance workflow, and clarifies what compliance-related activities are expected in each phase. For example, before audit, it requires evidence collection and the preparation of compliance reports.
Enablement solution	The model does not specify a particular solution type. Instead, it provides placeholders that can support different enablement mechanisms. For example, compliance checks can be implemented through rule-based scripts, ontology-based queries, or other mechanisms.
Trigger	The model shows various triggers for compliance work, including standard or system changes, merges, and runtime data. These triggers can lead to activities such as change impact analysis or evidence updates. The model is further refined in the next iteration to include more triggers.
Requirement extraction	The model represents requirement extraction as an upstream activity that generates compliance requirements from regulatory documents. Different extraction methods can be used for this step.

The bottom part still represents the software development lifecycle. It illustrates the relationship between system descriptions, system changes and compliance checks. While the system changes, the modified parts need to be traced back to compliance requirements. System changes can also trigger checks before merging. In an AI development environment, compliance requirements and compliance checks should be input to the development to ensure compliance throughout the lifecycle, which is titled as 'compliance by design'.

The refined model also distinguishes between different types of compliance check. These include existence checks, KPI-based checks, content checks, and context checks. The model further represents the degree of automation as a spectrum from fully manual to fully automated.

Another important refinement is the introduction of lifecycle governance and exemption maintenance. Compliance checks and exemptions are not treated as static elements. Instead, they need to be introduced, maintained, versioned, and reviewed over time. The exemption list represents cases where a compliance issue can be tem-

porarily accepted or justified. This enables the model to reflect situations in which a failed check does not automatically halt development, but must be documented and considered in the future.

The right side of the model still represents assessment process. Compliance checks and development activities produce compliance evidence, which supports the compliance argument. The compliance argument is then used in the compliance assessment. The model also distinguishes between compliance reports and assessment reports as inputs and outputs of the assessment process.

As discussed, the themes identified from the workshop have been incorporated into the reference model in different ways. To further trace this refinement back to the results of the literature review, Table 5.3 summarizes how the original taxonomy dimensions in Table 5.1 are represented in the reference model.

Overall, the second iteration expanded the reference model, shifting to a broader, evidence-based, risk-aware process. It introduced standard changes, impact analysis, check types, automation levels, runtime evidence, exemption maintenance and assessment-related documents. However, the evaluation revealed that DevOps integration remained under-specified, so this became the central focus of the next iteration.

TReqs-Compliance Demo

The demo showed that the initial TTIM and the rule map can be used to instantiate selected parts of the reference model at the repository level. The prototype can represent standards, clauses, compliance requirements, system description in TReqs.

The implementation provides initial proof of feasibility for artifact-oriented continuous compliance solution. In particular, it shows that some compliance requirements can be translated into structural rules over repository artifacts. Existence checks can be handled as traceability checks, while simple content checks can be implemented as checks for required fields. This supports the idea that parts of compliance evidence collection can be automated when the relevant requirements are sufficiently explicit and the artifacts are structured.

The rule map also has potential for extensibility. By separating rule definitions from checking logic, the prototype can be extended with additional rule types and checking scripts. This is important because continuous compliance involves heterogeneous artifacts and different degrees of automation.

However, the demonstration remains limited. Formal methods focus only on structural properties, such as the existence of links and required fields. They do not evaluate the semantic correctness and the quality of the artifact content. Therefore, further work is needed to support more complex checks, DevOps integration, and automated evidence collection.

5.3 Iteration 3

The third iteration focused on refining the reference model to allow for stronger DevOps integration and to complete the prototype plugin. The Iteration 2 reference model and prototype were discussed in a second Software Center workshop. The feedback emphasized that continuous compliance should be connected more clearly to the DevOps lifecycle.

Based on this feedback, the reference model was redesigned to integrate a DevOps-oriented view of development and operation. In parallel, the treqs-compliance plugin was extended from simple traceability and required-field checks toward a more complete rule-based workflow and to include many other advanced functions.

5.3.1 Problem Investigation

The third iteration began with an evaluation of the results from Iteration 2 in another Software Center workshop. Participants discussed the refined reference model and the initial demo.

The demo was used to illustrate a repository-level realization of the reference model, showing how traceability links could connect standards, compliance requirements, and BOs, and how rule-based methods could be used for existence check and content check.

The feedback on the demo was primarily used as a feasibility check. Although the participants did not provide extensive comments on the implementation details, the discussion indicated that repository-level traceability, explicit evidence (such as the generated report), and lightweight automated checks were all relevant and helpful for continuous compliance.

The main piece of feedback is on the model. Many participants pointed out that the model still focused too much on repository-level activities and those that occur before merging. In particular, it was highlighted in the discussion that the model should not stop at code changes or merge decisions. Continuous compliance also needs to cover later DevOps activities such as building, testing, releasing, deploying, operation and monitoring. Participants also emphasized that runtime data and operational statistics are not only additional sources of evidence, but are also necessary for verifying whether the assumptions made during development remain valid during operation.

This revealed a limitation of the previous model. Although it mentioned runtime data and context checks, it did not adequately represent how compliance checks and evidence collection are integrated across the DevOps lifecycle. Consequently, Iteration 3 aimed to refine the reference model, establishing a stronger connection between continuous compliance and DevOps activities, and finalize the prototype implementation.

5.3.2 Solution Design

Based on the feedback from the third workshop, the reference model was refined to more accurately reflect the relationship between continuous compliance and DevOps. This is realized by explicitly linking compliance activities to the DevOps lifecycle.

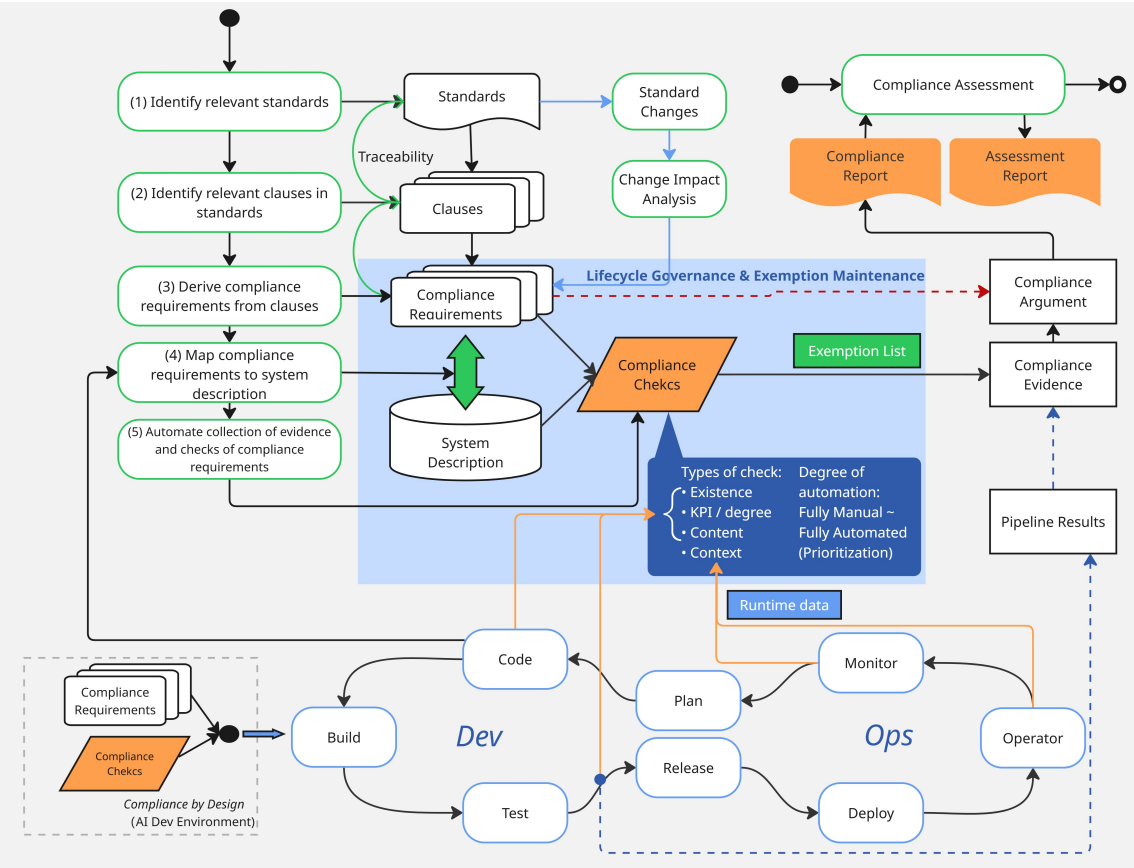


Figure 5.9: DevOps-integrated continuous compliance reference model

As shown in Figure 5.9, the left side retains the flow of standards to requirements. Compared with the previous versions, this model combines both workflow and artifact-flow of continuous compliance, completely and explicitly demonstrating the continuous compliance lifecycle.

The main refinement in this iteration is the explicit representation of the DevOps lifecycle. The model now includes development and operation activities such as code, build, test, plan, release, deploy, monitor, and operator activities. Compliance checks can be connected to different steps in this lifecycle. For example, existence check, KPI check, content check may be executed during development and in CI pipeline, while context check can be executed in runtime, collecting operational statistics or monitoring outputs.

The refined model therefore extends the scope of continuous compliance from repository-level and pre-merge workflows to encompass a more comprehensive DevOps-incorporated process. It demonstrates that compliance evidence can be generated throughout the development process, including pipeline execution, deployment and runtime mon-

itoring. At the same time, the model preserves earlier concepts such as lifecycle governance and exemption maintenance.

5.3.3 Solution Implementation

The prototype implementation developed in this thesis is available in a public project repository.[1]

The Rule Map

In Iteration 3, the demo was extended from simple required-field checking to a more complete rule-based compliance checking solution. The rule map was refined so that each rule is tagged as a compliance requirement, as shown in Fig. 5.10. This better aligns the reference model, where compliance requirements are derived from clauses and serve as input to checks.

```
COMPLIANCE_RULES = {
    #<treqs-element id="81c89f38592911f193da2811a89e7a4a" type="compl.requirement">
    ### Requirements for configManagementPlan (systemBO)
    "configManagementPlan": {
        "required_fields": [
            "baseline",
            "configuration_management_strategy"
        ],
        "description": "ISO 26262-8 Clause 7.5.1"
    },
    #<treqs-link type="derivedFrom" target="a133ff18592811f1a0382811a89e7a4a" />
    #</treqs-element>

    #<treqs-element id="e98e3c29592911f1abbc2811a89e7a4a" type="compl.requirement">
    ### Requirements for changeRequest (systemBO)
    "changeRequest": {
        "required_fields": [
            "date",
            "reason",
            "description",
            "config"
        ],
        "required_children": ["changeRequestAnalysis"],
        "description": "ISO 26262-8 Clause 8.5.2"
    },
    #<treqs-link type="derivedFrom" target="03a34fb9592911f18f822811a89e7a4a" />
    #</treqs-element>
```

Figure 5.10: The compliance rule map in iteration 3

The updated rule map defines rules derived from ISO 26262 Part 8 Clause 7 and Clause 8. They are requirements for several system-level boundary object types, including configuration management plans, change requests, change request analyses, and change reports. Each rule specifies the required fields for the relevant document.

[1] https://github.com/inco-yjl/treqs-compliance-public-

Some rules also define the necessary child documents. For instance, a change request requires a linked change request analysis, while a change request analysis requires documentation in the change report. In this way, the rule map supports content and structural dependency checks between system-level boundary objects.

The previous `min_links` field was removed from the rule map because it represents a constraint on abstract boundary objects rather than on system-level boundary objects. However, there are no direct links between rules and abstract boundary objects, so this design approach is not feasible. Instead, this constraint is now represented as an attribute of the abstract boundary object. For example, in Fig. 5.12, the second Treqs element has an attribute called `requireImplementation` which is set to false. This attribute indicates whether an abstract boundary object must have a corresponding system-level implementation.

```
### 8.5 Work products
<treqs-element id="5fbb1ac1ee5711efabf62811a89e7a4a" type="abstractBoundaryObj">

#### 8.5.1 Change management plan
Resulting from requirements 8.4.1.
<treqs-link type="derivedFrom" target="03a34fb9592911f18f822811a89e7a4a" />
</treqs-element>
<treqs-element id="68ca42a5ee5711ef8a722811a89e7a4a" type="abstractBoundaryObj" requireImplementation="false">

#### 8.5.2 Change request
Resulting from requirements 8.4.2.
<treqs-link type="derivedFrom" target="03a34fb9592911f18f822811a89e7a4a" />
</treqs-element>
```

Figure 5.11: Abstract boundary objects in clause 8

To improve extensibility, the rule-verification logic has been moved into a separate module. The main compliance checker is now responsible for loading elements, building traceability maps, applying rules, merging results and exporting reports. Individual rule checkers are only responsible for evaluating specific rule fields. The system uses a checker map to allocate rule fields to their respective checking methods. This design enables new compliance rule types to be added without altering the main checking workflow.

The `required_fields` checker has been refined to support files that contain multiple boundary objects. Originally, the checker searched the entire Markdown file for YAML metadata and headings. This could result in false positives.

The updated approach now only extracts the content inside the relevant `<treqs-element>` block for the current system boundary object. The checker then searches within that block for YAML code blocks and Markdown headings. Consequently, required fields are validated at the boundary-object level rather than the file level, resulting in a more precise content check.

Lifecycle Governance

Lifecycle management was also added for system-level boundary objects. Each `systemLevelBoundaryObj` can define a lifecycle status using the attribute `status`. For example:

```
<treqs-element id="6e1a9cb5596511f19f342811a89e7a4a"
type="systemLevelBoundaryObj"
document='changeRequestAnalysis' status="inactive">

# Change Request Analysis

```yaml
change type: error resolution
```
```

Figure 5.12: Inactive System Boundary Object

If a system boundary object is marked as inactive, the compliance checker will skip all rule checks for that object. The report includes a note to this effect. The individual system status is left blank and an inactive object does not have a negative impact on the overall compliance result. This supports cases where an artifact exists in the repository but should not currently participate in compliance evaluation.

Change Detection and Review

The review mechanism was also refined. Rather than defining review requirements directly in the rule map, a Git-based review gate was introduced, whereby the checker now compares the current artifact content with the previously committed version. It detects whether a file containing a system boundary object has changed compared to the base branch or the local working tree. If an active system boundary object is located in a file that has been modified, the object must include review evidence before it can pass compliance checking. The review evidence can be stored inside a YAML codeblock in the TReqs element, for example:

```
review_status: approved
reviewed_by: reviewer
reviewed_at: 2026-05-27
```

If the file has changed but no valid review evidence is found, the system boundary object fails and a note indicating that a review is required before the check can pass. If review evidence is present, the checker continues with the normal compliance rules. This creates a lightweight lifecycle gate: compliance-relevant artifacts that have been changed cannot return to a passing state until they have been reviewed, and the change and review decisions must be recorded to ensure compliance.

Implementation Architecture

The treqs-compliance prototype consists of two different parts: the target software system and the compliance tool. More specifically, these two parts are, respectively, repository-level inputs and additional checking code. Repository-level inputs in-

clude TReqs-annotated artifacts and rule definitions. The added checking code reads these inputs, resolves TReqs elements and links, applies the rules, and generates a compliance report. In this thesis, compliance is evaluated on the treqs-compliance repository itself. Therefore, the tool implementation and the target system artifacts are located in the same repository, but they play different conceptual roles. Table 5.4 summarizes the main components in the prototype, their formats, roles, and expected update frequency.

Table 5.4: Main components of the prototype

Component	**Format**	**Role**	**Update frequency**
TReqs annotations	`<treqs-element>` and `<treqs-link>` embedded in markdown or comments in code files	Annotate standards, clauses, requirements, abstract BOs, system-level BOs, with attributes and trace links	Updated when the status or mapping of artifacts change.
System files	Text-based documents (markdown) and source code (the prototype uses Python)	Provide the concrete system-level BOs	Updated frequently during development.
Rule map	Structured rule definitions used by the checker (the prototype defines a Python dictionary)	Defines rules for each artifact type, such as required fields, required child artifacts, and lifecycle-related constraints.	Updated less frequently, mainly when standard interpretation changes.
Checker code	Python implementation	Parses TReqs elements, resolves trace links, applies rules, handles lifecycle status, detects changed files, and generates reports	Updated when new check types or reporting functions are added.
Compliance report	Generated PDF document	Shows the status as pass, fail, or warning, along with the warning message.	Generated automatically whenever the checker is run.

The TReqs language is used mainly through annotations. These annotations are embedded in repository artifacts. In markdown files, they can be written directly in the document. While in source code, they can be placed inside comments. The annotations define compliance-related elements, such as standards, clauses, compliance requirements, abstract BOs, and system-level BOs, as well as the traceability links between them.

The rule map defines the checks that should be applied to different artifact types. Since the rule map reflects the interpretation of compliance requirements, it is not

expected to change frequently. In contrast, the system artifacts are expected to change more frequently during development, because they will be updated as the system evolves.

The components mentioned above, the TReqs annotations, system files, and the rule map, belong to the repository input side. In contrast, on the tool side, the added checking logic reads the repository input, resolves the TReqs elements and links, applies the rule map, checks lifecycle and review information, and generates a compliance report.

In a real setting, creating and maintaining the system input would probably require collaboration between compliance experts, requirements engineers, and system developers. Since treqs-compliance is designed to be extensible, project developers may also contribute project-specific checking logic when needed. This collaboration between different stakeholders aligns with the BOMI design, in which boundary objects facilitate coordination across different methodological areas.

5.3.4 Solution Evaluation

The final evaluation compares the prototype that has been implemented with the final reference model. The aim is not to claim that the prototype implements the entire reference model, but rather to determine which elements of the model have been instantiated and which remain conceptual.

The final reference model describes continuous compliance through several connected concepts: standards, clauses, compliance requirements, system descriptions, compliance checks, exemptions, compliance evidence, compliance arguments, assessment activities, and DevOps-related artifacts. The prototype primarily implements the artifact-oriented and repository-level components of this model. Standards, clauses, compliance requirements, work products are tagged as TReq elements. Trace links connect standards to clauses, clauses to requirements, and abstract boundary objects to concrete system artifacts.

The rule-based checker implements some of the compliance check types defined in the reference model. It supports structural checks, including existence and initial content checks. Lifecycle governance is partially implemented through a status attribute on system-level boundary objects. When an artifact is marked as 'inactive', this becomes an exemption for now.

The prototype also implements a simple review mechanism. If a file has changed since it was committed, the checker requires evidence of the review before the item can pass. This implements the idea in the reference model that changes may require compliance work and that review decisions should form part of the maintained evidence.

Fig. 5.13 shows the generated compliance report. The compliance report has also been improved to better represent the new checking logic. It merges abstract boundary object existence results with system-level rule results. For each abstract requirement, the report shows linked system boundary objects, their individual statuses,

Table 5.5: Comparison between the final reference model and the demonstration

Reference model concept	Implemented	Notes
Standards, clauses, requirements	Yes	Represented as TReqs elements and trace links.
System Description	Yes	Modeled as system-level boundary objects and mapped to abstract boundary objects in standard.
Compliance checks	Partly	Automated existence check and basic Content check; rule-based methods
Exemptions and lifecycle governance	Partly	Implemented through status attributes of system boundary objects.
Change triggers	Partly	Changed files trigger warnings until review evidence is recorded.
Compliance evidence	Partly	Represented through artifacts, links, check results, and report output.
Compliance argument	No	The tool creates reports but does not construct a full argument.
DevOps/runtime evidence	No	Included in the reference model but not implemented in the prototype.

and notes explaining any issues. The report uses text-based status labels such as 'PASS', 'WARN' and 'FAIL', combined with background colors, to improve readability.

While the report does not represent a complete compliance argument, it provides a lightweight view of the evidence. It shows which requirements have linked documents, which checks pass or fail, which artifacts are inactive, which artifacts require a relevant document and which changes require review.

Requirement	Overall	System BO Location	Status	Notice
#### 7.5.1 Configuration management plan	FAIL	compliance\iso-26262\part-8\workitems\CM_plan.md:4	FAIL	ISO 26262-8 Clause 7.5.1: Missing required fields: configurationmanagementstrategy
#### 8.5.1 Change management plan	FAIL	-	-	No linked system implementation
#### 8.5.2 Change request	FAIL	compliance\iso-26262\part-8\workitems\CR1.md:1	PASS	ISO 26262-8 Clause 8.5.2
		compliance\iso-26262\part-8\workitems\CR2.md:1	FAIL	ISO 26262-8 Clause 8.5.2: Missing required child documents linked by address: changeRequestAnalysis
#### 8.5.3 Impact analysis and change request plan	PASS	compliance\iso-26262\part-8\workitems\CR1_analysis.md:1	PASS	ISO 26262-8 Clause 8.5.3
		compliance\iso-26262\part-8\workitems\CR2_analysis.md:1	-	Inactive system BO: rule checks skipped
#### 8.5.4 Change report	WARNING	compliance\iso-26262\part-8\workitems\ChangeDoc.md:2	WARNING	File changed; review evidence required before pass

Figure 5.13: Generated compliance report from the final prototype

Overall, the evaluation shows that the prototype operationalizes selected parts of the final reference model: repository-level traceability, structural compliance checks,

lifecycle status, review warnings, and report generation. However, several parts of the reference model remain outside the implementation scope. The prototype does not implement runtime monitoring, DevOps pipeline evidence, KPI checks, context checks, full exemption governance, or automatic construction of compliance arguments. Therefore, the implementation should be understood as a partial demonstration of how PRITM tooling can support continuous compliance, rather than as a complete continuous compliance platform.

6

Discussion and Conclusion

6.1 Answering the Research Questions

RQ1: How do software development activities and their resulting artifacts support the collection and maintenance of compliance evidence for continuous compliance?

The results show that achieving continuous compliance requires both activities and artifacts. Development activities such as system changes, change reviews, testing, and CI pipeline execution can all generate compliance-relevant evidence. However, this evidence is only useful if it can be traced back to compliance requirements and to the clauses and standards from which these requirements were derived.

This finding confirms previous work that emphasizes traceability, versioning, and automated documentation as key mechanisms for continuous compliance [1]. It also aligns with process-integrated approaches, which argue that compliance activities need to be integrated with development workflows rather than treated as isolated audit preparation tasks [5], [8]. However, this thesis complements these studies by explicitly and comprehensively connecting development activities, compliance workflow, compliance relevant-artifacts, and assessment activity in one reference model.

The reference model developed in this thesis illustrates this relationship by linking standards, clauses, compliance requirements, system descriptions, compliance checks, evidence, and compliance arguments. A key finding is that compliance evidence must be maintained over time. Since both standards and systems can change, evidence can become outdated if relevant artifacts or trace links are not updated. This complements work by Rouland et al., who emphasize the need to handle both software evolution and regulatory change in continuous compliance processes [3].

Therefore, the answer to RQ1 is that software development activities support continuous compliance by producing and updating artifacts that can serve as evidence of compliance. However, these artifacts only become useful evidence when they are traceable, maintained and connected to compliance requirements, checks and arguments.

RQ2: How can a PRITM tool such as TReqs support continuous compliance?

The TReqs plugin prototype, treqs-compliance, demonstrates that PRITM tools can support continuous compliance by representing compliance-related artifacts directly within the product repository and by making traceability relationships explicit.

In this case, treqs-compliance is used to represent standards, clauses, compliance requirements, abstract boundary objects, and system-level boundary objects. This enables requirements derived from ISO 26262 Part 8 to be linked to specific work products. The generated traceability graph, as shown in Fig. 5.4, demonstrates how TReqs can provide a repository-level view of compliance relationships.

This finding complements previous work on TReqs and BOMI-based compliance support [4]. While earlier work focused more using boundary objects to facilitate coordination, this thesis shifts the focus toward continuous compliance and evidence maintenance over time. It also confirms prior research on boundary objects as useful interfaces between different methodological islands and stakeholder groups [8], [13], [14].

The prototype also confirms findings from rule-based compliance approaches. Similar to previous work that uses policy maps, scripts, or verification tools to automate parts of compliance checking [23], [24], treqs-compliance shows that lightweight structural checks can be automated. Existence checks are solved as traceability checks between boundary objects. Simple content checks are implemented by verifying the presence of required fields in Markdown files. The rule map associates artifact types with the relevant rule entries.

However, the prototype reinforces a limitation identified in previous research: formal methods alone are insufficient for achieving perfect continuous compliance. More complex compliance issues could still require semantic interpretation, human review or advanced AI solutions [6], [25]. Artifact-oriented methods also tend to neglect process compliance. Therefore, although PRITM tools can provide an important integration point for continuous compliance, they need to be combined with requirements interpretation techniques, CI/CD pipelines and potentially runtime monitoring to cover the full lifecycle.

RQ3: To what extent can PRITM tools reduce the effort required to achieve continuous compliance compared with traditional approaches?

Referring to the finalized reference model, the results suggest that PRITM tools can primarily reduce the effort required for structural and compliance work. It aligns with prior work on automated verification-based compliance, which shows that automation is most feasible when compliance concerns can be expressed as explicit rules or checks over structured artifacts [23], [24].

Traditional compliance activities are often labor-intensive when performed manually, especially if evidence collection and traceability construction have to be repeated before every audit. This solution can significantly reduce the effort by providing traceability management and evidence maintenance. In this respect, the solution also complements previous compliance-as-code and rule-based approaches.

However, the reduction in effort is limited for compliance concerns that require

interpretation, justification, or domain expertise. For example, determining whether a risk analysis is complete, whether an exemption is reasonable, or whether a runtime behavior aligns with a context-specific safety requirement cannot be accomplished solely through traceability alone. This confirms the limitations identified in semi-automatic and ontology-based compliance approaches, where human judgment is still required for complex semantic or contextual checks [6], [25].

Therefore, PRITM tools should not be viewed as a replacement for traditional compliance work. Instead, they can reduce manual effort by automating low-level structural checks, maintaining traceability, and preparing evidence for review. This complements previous research by positioning PRITM tools as a means of integrating development artifacts, automated checks and compliance evidence rather than as a fully automated compliance assessment solution.

6.2 Conclusion

Through three design iterations, a reference model for continuous compliance was developed and progressively refined. The model evolved from an activity-centered process view, to an artifact-centered view, and finally to a DevOps-oriented view that connects compliance activities and artifacts across development and operation.

The thesis also explored how selected concepts from the reference model could be realized using a TReqs plugin. The resulting prototype demonstrates how standards, clauses, compliance requirements, and system artifacts can be represented in a product repository. It also demonstrates simple, rule-based checks and an initial compliance report.

Overall, the work produces both a conceptual structure and a practical tool for continuous compliance. A reference model clarifies what needs to be connected, and a PRITM tool such as TReqs can manage these connections in practice. This approach is most effective for structural compliance tasks such as traceability maintenance, artifact existence checks and simple content checks. However, more complex compliance reasoning still requires human judgment and further tool support.

6.3 Future Work

First, future work should evaluate the prototype in a more realistic industrial setting. While the current work demonstrates the feasibility of the approach on selected ISO 26262 examples, it does not yet measure the actual effort reduction achieved in concrete industrial use cases.

Secondly, the approach should be applied to a broader set of standards and domains. This thesis primarily uses ISO 26262 Part 8 for demonstration purposes. Applying the model to other standards, such as those relating to cybersecurity, AI governance or cloud compliance, would help to determine whether the proposed artifact types, check categories and traceability relations are sufficiently general.

Thirdly, future work should investigate more advanced compliance checks. While the prototype mainly focuses on structural checks, more semantic and context-dependent checks remain difficult. Future work could explore how natural language processing, large language models and domain-specific analysis tools can support content and context checks while maintaining human review.

Fourthly, the relationship between continuous compliance and runtime operation requires further study. While the final reference model incorporates DevOps and runtime evidence, the demonstration does not yet implement runtime monitoring or operational feedback. Future work could examine how runtime data, monitors, logs and operational assumptions can be linked to compliance requirements and arguments.

Finally, future work should examine how compliance arguments can be generated or maintained from repository-level evidence. While the current prototype produces simple reports, it does not construct a full compliance argument. Further work could investigate how evidence can be transformed into structured compliance arguments for auditors.

Bibliography

[1] T. Santilli, P. Pelliccione, R. Wohlrab, and A. Shahrokni, "Continuous compliance in the automotive industry," *IEEE Software*, vol. 41, no. 4, pp. 134–142, 2024. DOI: 10.1109/MS.2023.3342974.

[2] F. Angermeir, J. Fischbach, F. Moyón, and D. Mendez, "Towards automated continuous security compliance," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '24, Barcelona, Spain: Association for Computing Machinery, 2024, pp. 440–446, ISBN: 9798400710476. DOI: 10.1145/3674805.3690748. [Online]. Available: https://doi.org/10.1145/3674805.3690748.

[3] Q. Rouland, S. Gjorcheski, and J. Jaskolka, "Eliciting a security architecture requirements baseline from standards and regulations," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, 2023, pp. 224–229. DOI: 10.1109/REW57809.2023.00045.

[4] M. O. Aduamah and M. H. Araya, "Enhancing coordination, traceability and compliance in systems development," Master's thesis, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden, 2024. [Online]. Available: https://gupea.ub.gu.se/items/59524d29-7563-4aef-8356-37835b749b26.

[5] F. Moyon, K. Beckers, S. Klepper, P. Lachberger, and B. Bruegge, "Towards continuous security compliance in agile software development at scale," in *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE '18, New York, NY, USA: Association for Computing Machinery, 2018, pp. 31–34, ISBN: 9781450357456. DOI: 10.1145/3194760.3194767. [Online]. Available: https://doi.org/10.1145/3194760.3194767.

[6] T. Schubert, M. Ibrahim, N. Breitmoser-Widdecke, and U. Durak, "An ontology-based automation framework for continuous compliance of airborne software," in *2025 AIAA DATC/IEEE 44th Digital Avionics Systems Conference (DASC)*, 2025, pp. 1–8. DOI: 10.1109/DASC66011.2025.11257376.

[7] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.06.063. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121215001430.

[8] A. Shenouda et al., "Using boundary objects for continuous compliance in automotive development," in *2024 IEEE 35th International Symposium on Soft-*

ware Reliability Engineering Workshops (ISSREW), 2024, pp. 224–231. DOI: 10.1109/ISSREW63542.2024.00083.

[9] E. Knauss et al., "T-reqs: Tool support for managing requirements in large-scale agile system development," *arXiv preprint arXiv:1805.02769*, 2018.

[10] R. Kasauli, G. Liebel, E. Knauss, S. Gopakumar, and B. Kanagwa, "Requirements engineering challenges in large-scale agile system development," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 352–361. DOI: 10.1109/RE.2017.60.

[11] J. Holtmann et al., "Using boundary objects and methodological island (BOMI) modeling in large-scale agile systems development," *Software and Systems Modeling*, vol. 24, no. 1, pp. 183–207, 2025.

[12] S. L. Star and J. R. Griesemer, "Institutional ecology,translations' and boundary objects: Amateurs and professionals in berkeley's museum of vertebrate zoology, 1907-39," *Social studies of science*, vol. 19, no. 3, pp. 387–420, 1989.

[13] R. Wohlrab, P. Pelliccione, E. Knauss, and M. Larsson, "Boundary objects and their use in agile systems engineering," *Journal of Software: Evolution and Process*, vol. 31, no. 5, e2166, 2019.

[14] R. Wohlrab, J. Horkoff, R. Kasauli, S. Maro, J.-P. Steghöfer, and E. Knauss, "Modeling and analysis of boundary objects and methodological islands in large-scale systems development," in *International Conference on Conceptual Modeling*, Springer, 2020, pp. 575–589.

[15] *Iso 26262:2018 road vehicles – functional safety*, Geneva, Switzerland: International Organization for Standardization, 2018.

[16] R. Debouk, "Overview of the second edition of iso 26262: Functional safety road vehicles," *Journal of System Safety*, vol. 55, no. 1, pp. 13–21, Mar. 2019. DOI: 10.56094/jss.v55i1.55. [Online]. Available: https://jsystemsafety.com/index.php/jss/article/view/55.

[17] S. Ghaisas, M. Motwani, B. Balasubramaniam, A. Gajendragadkar, R. Kelkar, and H. Vin, "Towards automating the security compliance value chain," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015, Bergamo, Italy: Association for Computing Machinery, 2015, pp. 1014–1017, ISBN: 9781450336758. DOI: 10.1145/2786805.2804435. [Online]. Available: https://doi.org/10.1145/2786805.2804435.

[18] Q. Rouland, S. Gjorcheski, and J. Jaskolka, "Eliciting a security architecture requirements baseline from standards and regulations," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, 2023, pp. 224–229. DOI: 10.1109/REW57809.2023.00045.

[19] T. D. Breaux, M. W. Vail, and A. I. Anton, "Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations," in *14th IEEE International Requirements Engineering Conference (RE'06)*, 2006, pp. 49–58. DOI: 10.1109/RE.2006.68.

[20] J. Luttmer, V. Prihodko, D. Ehring, and A. Nagarajah, "Requirements extraction from engineering standards systematic evaluation of extraction techniques," *Procedia CIRP*, vol. 119, pp. 794–799, 2023, The 33rd CIRP Design Conference, ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.

2023.03.125. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827123005802.

[21] S. Abualhaija, C. Arora, and L. C. Briand, "Coreqqa: A compliance requirements understanding using question answering tool," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022, Singapore, Singapore: Association for Computing Machinery, 2022, pp. 1682–1686, ISBN: 9781450394130. DOI: 10.1145/3540250.3558926. [Online]. Available: https://doi.org/10.1145/3540250.3558926.

[22] O. A. Cejas, M. I. Azeem, S. Abualhaija, and L. C. Briand, "Nlp-based automated compliance checking of data processing agreements against gdpr," *IEEE Transactions on Software Engineering*, vol. 49, no. 9, pp. 4282–4303, 2023. DOI: 10.1109/TSE.2023.3288901.

[23] R. Filepp, C. Adam, M. Hernandez, M. Vukovic, N. Anerousis, and G. Q. Zhang, "Continuous compliance: Experiences, challenges, and opportunities," in *2018 IEEE World Congress on Services (SERVICES)*, 2018, pp. 31–32. DOI: 10.1109/SERVICES.2018.00029.

[24] M. Kellogg, M. Schäf, S. Tasiran, and M. D. Ernst, "Continuous compliance," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, pp. 511–523.

[25] S. Fenz, G. Goluch, A. Ekelhart, B. Riedl, and E. Weippl, "Information security fortification by ontological mapping of the iso/iec 27001 standard," in *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*, ser. PRDC '07, USA: IEEE Computer Society, 2007, pp. 381–388, ISBN: 0769530540. DOI: 10.1109/PRDC.2007.45. [Online]. Available: https://doi.org/10.1109/PRDC.2007.45.

[26] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006. DOI: 10.1191/1478088706qp063oa.

A

Implementation Screenshots

```
def check_existence(self):
    self.existence_results = []

    for abstract_bo in self.abstract_bos:
        systems = self.trace_map.get(abstract_bo.uid, [])

        doc_type = abstract_bo.attributes.get("document")

        # default: every abstract BO requires implementation
        require_implementation = abstract_bo.attributes.get(
            "requireImplementation",
            "true"
        )
        if require_implementation=='true' and not systems:
            fulfilled = "NO"
            status = "FAIL"
            message = "No linked system implementation"
        elif require_implementation =='false' and not systems:
            fulfilled = "N/A"
            status = "PASS"
            message = "System implementation not required"
        else:
            fulfilled = "YES"
            status = "PASS"
            message = "Linked system implementation exists"

        result = {
            "abstract_id": abstract_bo.uid,
            "abstract_label": abstract_bo.label,
            "abstract_location": f"{abstract_bo.file_name}:{abstract_bo.placement}",
            "abstract_doc_type": doc_type,
            "fulfilled": fulfilled,
            "abstract_status": status,
            "abstract_message": message,
            "linked_systems": [s.uid for s in systems]
        }

        self.existence_results.append(result)
```

```python
def get_changed_files(self, base_ref="origin/main"):
    changed = set()
    commands = [
        ["git", "diff", "--name-only"],           # unstaged changes
        ["git", "diff", "--cached", "--name-only"], # staged changes
        ["git", "diff", "--name-only", f"{base_ref}...HEAD"], # committed branch changes
    ]
    for cmd in commands:
        result = subprocess.run(cmd,capture_output=True,text=True)
        if result.returncode == 0:
            changed.update(
                line.strip().replace("\\", "/")
                for line in result.stdout.splitlines()
                if line.strip()
            )

    return changed

def is_system_bo_file_changed(self, system_bo, changed_files):
    file_name = system_bo.file_name.replace("\\", "/")
    return any(
        file_name.endswith(changed_file)
        or changed_file.endswith(file_name)
        for changed_file in changed_files
    )
def extract_yaml_data(self, content):
    yaml_blocks = re.findall(
        r"```(?:yaml|yml)\s*\n(.*?)\n```",
        content,
        re.DOTALL | re.IGNORECASE
    )
    data = {}
    for yaml_content in yaml_blocks:
        try:
            parsed = yaml.safe_load(yaml_content)
            if isinstance(parsed, dict):
                data.update(parsed)
        except:
            continue

    return data
```

```
  def has_review_evidence(self, system_bo):
      file_content = self.load_content(system_bo)

      element_content = self.extract_treqs_element_block(
          file_content,
          system_bo.uid
      )

      yaml_data = self.extract_yaml_data(element_content)

      review_status = str(
          yaml_data.get("review_status", "")
      ).strip().lower()

      reviewed_by = yaml_data.get("reviewed_by")
      reviewed_at = yaml_data.get("reviewed_at")

      return (
          review_status in ["approved", "reviewed", "pass"]
          and reviewed_by
          and reviewed_at
      )
  def extract_treqs_element_block(self, content, uid):
      pattern = r"<treqs-element\b[^>]*>.*?</treqs-element>"

      for match in re.finditer(pattern, content, re.DOTALL):
          block = match.group(0)

          if uid in block:
              return block

      return ""
```

```
def load_rules(self, part_path):
    import importlib.util

    rules_file = os.path.join(part_path, "rules.py")

    spec = importlib.util.spec_from_file_location("rules", rules_file)
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)
```

```python
def get_changed_files(self, base_ref="origin/main"):

    changed = set()

    commands = [
        ["git", "diff", "--name-only"],           # unstaged changes
        ["git", "diff", "--cached", "--name-only"], # staged changes
        ["git", "diff", "--name-only", f"{base_ref}...HEAD"], # committed branch changes
    ]

    for cmd in commands:
        result = subprocess.run(
            cmd,
            capture_output=True,
            text=True
        )

        if result.returncode == 0:
            changed.update(
                line.strip().replace("\\", "/")
                for line in result.stdout.splitlines()
                if line.strip()
            )

    return changed
```

```
def apply_rules(self):
    self.system_rule_map = {}

    for system_bo in self.system_bos:
        lifecycle_status = self.get_lifecycle_status(system_bo)

        if lifecycle_status == "inactive":
            self.system_rule_map[system_bo.uid] = {
                "status": "-",
                "display_status": "",
                "message": "Inactive system BO: rule checks skipped",
                "notes": ["Inactive system BO: rule checks skipped"],
                "skip_for_overall": True,
            }
            continue
        file_changed = self.is_system_bo_file_changed(
            system_bo,
            self.changed_files
        )

        if file_changed and not self.has_review_evidence(system_bo):
            self.system_rule_map[system_bo.uid] = {
                "status": "WARNING",
                "message": "File changed; review evidence required before pass",
                "notes": [
                    "File changed since base branch",
                    "Missing review_status=approved, reviewed_by, or reviewed_at"
                ],
                "skip_for_overall": False,
            }
            continue
        doc_type = system_bo.attributes.get("document")

        if not doc_type:
            self.system_rule_map[system_bo.uid] = {
                "status": "FAIL",
                "message": "Missing document attribute",
                "notes": ["Missing document attribute"]
            }
            continue
```

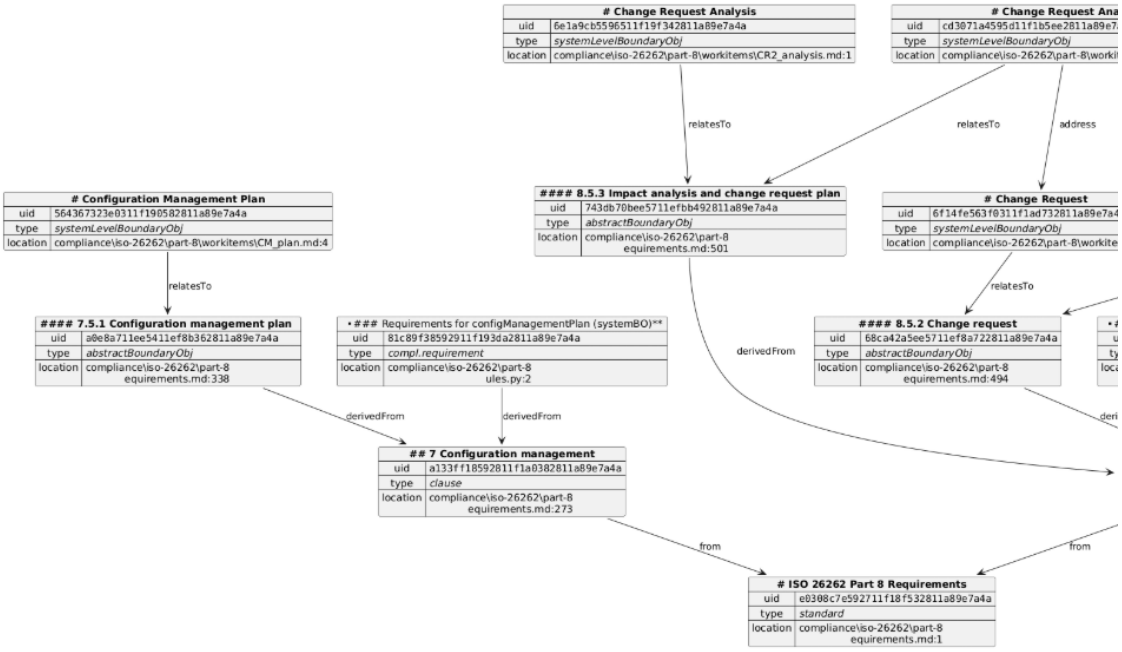
```
        if doc_type not in self.rules:
            self.system_rule_map[system_bo.uid] = {
                "status": "WARNING",
                "message": f"No system rule defined for document type: {doc_type}",
                "notes": [f"No system rule defined for document type: {doc_type}"]
            }
            continue

        rule = self.rules[doc_type]
        context = self.build_rule_context(system_bo)

        check_results = []

        for rule_field, rule_value in rule.items():
            if rule_field == "description":
                continue

            checker = self.rule_checkers.system_rule_checker_map.get(rule_field)

            if checker is None:
                check_results.append({
                    "status": "WARNING",
                    "note": f"No checker implemented for rule field: {rule_field}"
                })
                continue

            check_results.append(
                checker(context, rule_value, rule)
            )

        self.system_rule_map[system_bo.uid] = combine_rule_results(
            check_results,
            rule
        )
```

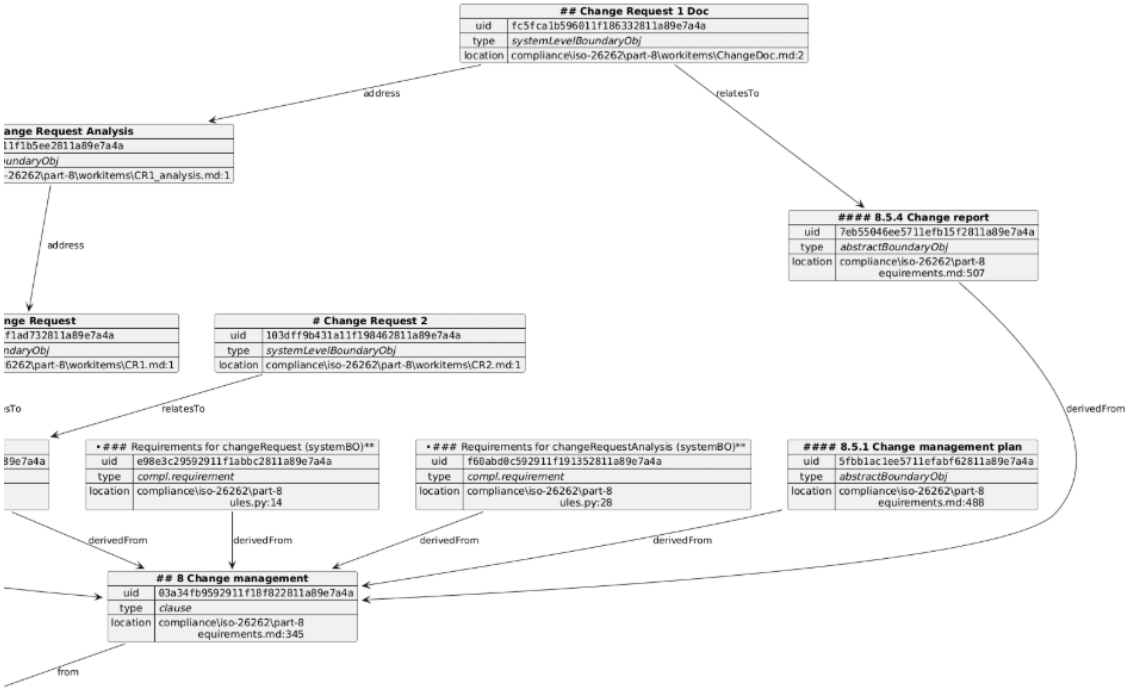
```python
def check_required_fields(self, context, rule_value, rule):
    content = context["content"]

    yaml_fields, heading_fields = self.extract_all_fields(content)
    all_fields = yaml_fields.union(heading_fields)

    required = {
        self.normalize(field)
        for field in rule_value
    }

    missing = required - all_fields

    if missing:
        return {
            "status": "FAIL",
            "note": f"Missing required fields: {', '.join(sorted(missing))}"
        }

    return {
        "status": "PASS",
        "note": ""
    }
```

```python
def check_required_children(self, context, rule_value, rule):
    parent_bo = context["system_bo"]
    all_system_bos = context["all_system_bos"]

    required_child_types = set(rule_value)
    found_child_types = set()

    for candidate_child in all_system_bos:
        child_doc_type = candidate_child.attributes.get("document")

        if child_doc_type not in required_child_types:
            continue

        for link in candidate_child.outlinks:
            link_type = getattr(link, "tlt", None)

            if link.target == parent_bo.uid and link_type == "address":
                found_child_types.add(child_doc_type)

    missing = required_child_types - found_child_types

    if missing:
        return {
            "status": "FAIL",
            "note": (
                "Missing required child documents linked by address: "
                + ", ".join(sorted(missing))
            )
        }

    return {
        "status": "PASS",
        "note": ""
    }
```

B

Traceability Graph




Change Request 1 Doc
uid fc5fca1b596011f186332811a89e7a4a
type systemLevelBoundaryObj
location compliance\iso-26262\part-8\workitems\ChangeDoc.md:2
address
relatesTo
ange Request Analysis
l1f1b5ee2811a89e7a4a
undaryObj
-26262\part-8\workitems\CR1_analysis.md:1
address
8.5.4 Change report
uid 7eb55046ee5711efb15f2811a89e7a4a
type abstractBoundaryObj
location compliance\iso-26262\part-8 equirements.md:507
nge Request
f1ad732811a89e7a4a
ndaryObj
6262\part-8\workitems\CR1.md:1
Change Request 2
uid 103dff9b431a11f198462811a89e7a4a
type systemLevelBoundaryObj
location compliance\iso-26262\part-8\workitems\CR2.md:1
sTo
relatesTo
derivedFrom
89e7a4a
•### Requirements for changeRequest (systemBO)**
uid e98e3c29592911f1abbc2811a89e7a4a
type compl.requirement
location compliance\iso-26262\part-8 ules.py:14
•### Requirements for changeRequestAnalysis (systemBO)**
uid f60abd0c592911f191352811a89e7a4a
type compl.requirement
location compliance\iso-26262\part-8 ules.py:28
8.5.1 Change management plan
uid 5fbb1ac1ee5711efabf62811a89e7a4a
type abstractBoundaryObj
location compliance\iso-26262\part-8 equirements.md:488
derivedFrom
derivedFrom
derivedFrom
derivedFrom
8 Change management
uid 03a34fb9592911f18f822811a89e7a4a
type clause
location compliance\iso-26262\part-8 equirements.md:345
from