Security Findings Management in Modern Industrial Software Development

Dissertation von Markus Ludwig Voggenreiter

München 2025

Security Findings Management in Modern Industrial Software Development

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

eingereicht von
Markus Ludwig Voggenreiter

05.02.2024

1. Gutachter/in:	Dr. Ulrich Schöpp
2. Gutachter/in:	Prof. Dr. Daniel Mendez
3. Gutachter/in:	Prof. Dr. Stefan Wagner
Tag der mündlichen Prüfung:	14.10.2024

Eidesstattliche Erklärung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, 01.02.2024

.....
(Voggenreiter, Markus Ludwig)

Abstract

The development of software has turned into one of the central activities for industrial companies over the last decades. With almost every industrial product across all industry sectors containing or entirely consisting of software, its secure and efficient development became crucial in practice. In particular, the assessment of software products for security shortcomings or vulnerabilities, plays a vital role during the secure software development lifecycle in industry. Similar to these checks, the management of security findings resulting from them is equally indispensable and required by multiple standards, guidelines and norms. With new trends and processes in the software engineering domain, including concepts like Agile Software Development or DevOps, industrial software engineering evolved from traditional concepts to modern software development approaches. However, this not only affects the software engineering itself, but also all security activities performed as part of the software development lifecycle. While areas like security testing already adapted to this shift by applying, e.g., automated security checks during all lifecycle stages, the management of security findings still lacks the transformation to modern software development principles. This is problematic for practitioners in industry, as it not only diminishes the efficiency of the software development process but infringes the security of products as well.

This thesis addresses this gap by researching and designing a methodology for the management of security findings in modern industrial software development projects. The methodology is based on the requirements arising from the state-of-practice security findings management and modern software development principles. Employing a three-step approach, the data quality of security findings is improved, reactions to each finding decided and the resulting information communicated to stakeholders. To measure the impact of the methodology, it is implemented as platform for the management of security findings and, in collaboration with our industry partner Siemens AG, evaluated in ongoing industrial software development projects. The results indicate the importance of a modernized security findings management process and confirm the relevance of our methodology for industrial practice.

The main contribution of this thesis is the methodology for the management of security findings in modern industrial software development projects. With its implementation as platform and evaluation in real-world projects, it contributes to the software engineering domain and industrial practice alike. Moreover, it yields several advancements in the areas of Knowledge Engineering, Software Security, and Natural Language Processing.

Zusammenfassung

In den letzten Jahrzehnten ist die Entwicklung von Software zu einer der zentralen Aktivitäten von Industrieunternehmen geworden. Da fast jedes Produkt unabhängig vom Industriezweig Software enthält oder vollständig aus Software besteht, ist eine sichere und effiziente Entwicklung in der Praxis von entscheidender Bedeutung. Insbesondere die Analyse von Softwareprodukten auf Sicherheitsmängel spielt für die Industrie eine zentrale Rolle im Lebenszyklus der sicheren Softwareentwicklung. Ähnlich wie die Untersuchungen selbst, ist auch die Handhabung und Lösung der daraus resultierenden Sicherheitsbefunde unverzichtbar und durch zahlreiche Standards, Richtlinien und Normen vorgeschrieben. Diese Weiterentwicklung betrifft nicht nur den Bereich der Softwaresicherheit, sondern auch die Softwareentwicklungsansätze selbst. So spielen moderne Konzepte wie agile Softwareentwicklung oder DevOps eine immer größere Rolle für die Industrie und beeinflussen auch alle Sicherheitsaktivitäten im Projekt. Während sich Bereiche wie das Testen bereits an diesen Wandel angepasst haben, z.B. durch automatisierte Sicherheitsprüfungen, fehlt diese Adaptation beim Management von Sicherheitsbefunden weitgehend noch. Dies stellt eine Herausforderung für die industrielle Praxis dar, nachdem es nicht nur die Effizienz der Softwareentwicklungs mindert, sondern auch die Produktsicherheit beeinträchtigt.

Die vorliegende Arbeit adressiert diese Lücke, indem sie eine Methodik für das Management von Sicherheitsbefunden in modernen industriellen Softwareentwicklungsprojekten erforscht und konzipiert. Basierend auf den Anforderungen aus dem Management von Sicherheitsbefunden in der industriellen Praxis sowie den Prinzipien der modernen Softwareentwicklung wird mithilfe eines dreistufigen Ansatzes die Datenqualität von Sicherheitsbefunden verbessert, über Konsequenzen für jeden Befund entschieden und die daraus resultierenden Informationen an die Projektstakeholder weitergegeben. Um die Auswirkungen der Methodik zu messen, wird sie als Plattform für das Management von Sicherheitsbefunden implementiert und in Zusammenarbeit mit unserem Industriepartner Siemens AG in Softwareentwicklungsprojekten evaluiert. Die Ergebnisse weisen auf die Relevanz eines modernisierten Prozesses für das Management von Sicherheitsbefunden hin und bestätigen den Stellenwert unserer Methodik für die industrielle Praxis. Der wissenschaftliche Hauptbeitrag dieser Arbeit ist die Methodik für das Management von Sicherheitsbefunden in modernen industriellen Softwareentwicklungsprojekten. Mit ihrer Implementierung und Evaluierung wird gleichermaßen ein Beitrag zum Bereich des Software-Engineerings sowie der industriellen Praxis geleistet. Darüber hinaus entstehen kleinere Beiträge zu den Bereichen Knowledge Engineering, Software Security und Natural Language Processing.

Acknowledgements

I would like to express my sincere and heartfelt gratitude to all those who have supported me throughout the incredible journey of this PhD. This thesis would not have been possible without the guidance, encouragement, and expertise of numerous individuals and groups, whose contributions have been essential at every step.

First and foremost, I would like to extend my deepest thanks to my supervisor, Dr. Ulrich Schöpp, for being more than just a supervisor – you have been a mentor, a guide, and a constant source of support. Your availability, patience, and willingness to engage in weekly discussions over the years, especially in times of uncertainty, have been indispensable.

I am also deeply grateful to the 'Fun with 4-1' group – Fabiola, Florian, and Pierre. The initial ideas for this research emerged from our collaboration, and I am still amazed at how our brainstorming sessions in 2019 sparked such a journey for all of us - it was gorgeous. Thank you for your continuous professional and emotional support throughout this project. You have become far more than just colleagues to me.

A large part of the insights presented in this thesis relies on the collaboration with Siemens AG, where I had the privilege of working alongside highly knowledgeable colleagues whose expertise in scientific research and innovation greatly enriched this work. From the very first day, I was welcomed into an environment of intellectual curiosity and excellence. I am especially grateful to Dr. Holger Dreger, whose support began long before the research on this topic even started. Your encouragement and assistance, extending well beyond the completion of the research, have meant a lot to me. I would also like to extend my gratitude to Berk, Colin, Ozan, Philipp, and Sandip, the working students who have supported the development of the Security Flama, both past and present.

Additionally, I would like to thank all of my co-authors for their invaluable collaboration. My sincere thanks also go to Prof. Dr. Daniel Mendez and Prof. Dr. Stefan Wagner for agreeing to act as second and third reviewers for this thesis. Your comments and remarks have deepened my understanding of research methodologies and broadened my scientific perspective.

Finally, I owe a special debt of gratitude to those who have supported me on an emotional level. To my wife Angelina, my parents, my dear friend Sejid, and the entirety of Team Six – thank you for always believing in me and for listening, even when my thoughts were cryptic or disjointed. Your unwavering support made this journey not only possible but truly enjoyable.

Table of Contents

Abstract **iv**

Zusammenfassung **vi**

Acknowledgements **viii**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Statement 2
1.3 Research Methodology 4
1.4 Contribution 6
1.5 Thesis Structure 8

2 Security Findings Management Problem Conceptualization **11**
2.1 Background, Terminology, and Literature 11
2.2 Requirements Towards the Industrial Security Findings Management Process 16
2.3 Problem Description 27
2.4 Objectives of the Methodology 35

3 Platform for Security Findings Management **43**
3.1 Problem Description 43
3.2 Related Work 44
3.3 Semantic Knowledge Base 46
3.4 Inference Logic 52
3.5 Implementation 57
3.6 Preliminary Evaluation 61
3.7 Conclusion 65

4 Improving Data Quality of Security Findings **67**
4.1 Problem Description 67
4.2 Related Work 69
4.3 Security Findings Collection and Parsing 71
4.4 Security Findings Aggregation and Correlation 80
4.5 Finding Information Enrichment 99

4.6 Implementation . . . 103
4.7 Preliminary Evaluation . . . 109
4.8 Conclusion . . . 112

5 Security Finding Analysis and Tracking 113
5.1 Problem Description . . . 113
5.2 Related Work . . . 116
5.3 Documentation and Tracking of Security Findings . . . 119
5.4 Analysis of Security Findings . . . 126
5.5 Prioritizing and Deciding on Security Findings . . . 130
5.6 Implementation . . . 142
5.7 Preliminary Evaluation . . . 148
5.8 Conclusion . . . 154

6 Security Findings Feedback Communication 155
6.1 Problem Description . . . 155
6.2 Related Work . . . 158
6.3 Communication Strategy . . . 159
6.4 Implementation . . . 163
6.5 Conclusion . . . 165

7 Instantiation and Evaluation in the Industry 167
7.1 Implementation and Integration of the Methodology . . . 167
7.2 Evaluation Planning . . . 171
7.3 Evaluation Design . . . 177
7.4 Evaluation Results . . . 187
7.5 Discussion . . . 195

8 Conclusion and Future Work 203

A Analyzed Documents for State of Industrial Practice 207

B List of DevOps Principles 215

C Visual Communication Interface 217

D Changelog of the Security Flama during Evaluation 219

E Quantitative Evaluation Results 221

List of Figures

1.1 Visualization of the Design Science Research Approach 5
1.2 Design Science Iterations 6

2.1 Core Terminology 14
2.2 Review Protocol 19
2.3 Blackbox Data Flow Diagram 33
2.4 Security Findings Process - Data Flow Diagram 34
2.5 Problem Conceptualization Process 41

3.1 Conceptualized Components of the Knowledge base 46
3.2 Relation between Belief Classes and their Instances 48
3.3 Orchestration of the Inference Engine 50
3.4 Relationship between Instances of Belief 54
3.5 Flow Chart for Maintaining Soundness after an Add and Delete Operation 55
3.6 Component Diagram of the Semantic Knowledge Base 58
3.7 Example Process for Adding Belief 60
3.8 Example Process for Reading Belief 60
3.9 Example Process for Deleting Belief 61

4.1 Proactive and Reactive Collection Strategies 72
4.2 SeFiLa Webinterface 85
4.3 Results of KG-Based, LSI, and SBERT Clustering of the SAST Dataset . . 90
4.4 Results of KG-Based, LSI, and SBERT Clustering of the DAST Dataset . . 90
4.5 Clustering Process for Industrial Security Findings 96
4.6 Diagram of the Data Quality Improvement Process 103

5.1 Occurrence of Security Findings across multiple Dimensions 120
5.2 Deterministic State Diagram of the Security Findings Lifecycle 124
5.3 Flow Chart of the Security Findings Validation Process 129
5.4 Base Score Calculation Process 132
5.5 Refined Score Calculation Process 135
5.6 Finding Response Planning Strategy as Flow Chart 138

6.1 Recommendation System Design for Security Findings Management 163

6.2 Implementation of the Communication Strategy 165

7.1 Belief Classes, Rules and Queries implemented in the Platform 168
7.2 Simplified ABC Framework 172
7.3 Finding Statistics of both Projects 189
7.4 Findings Status of Project A 190
7.5 Findings Status of Project B 191
7.6 Severity of Findings in Project A 192
7.7 Severity of Findings in Project B 193

C.1 Webinterface Overview Page 217
C.2 Webinterface Findings List Page 218
C.3 Webinterface Finding Page 218

List of Tables

2.1 Requirements towards the Industrial Security Findings Management 25
2.2 Topic List of Modern Software Development Principles 30
2.3 Clusters of Requirements 34
2.4 Thesis Objectives 40

3.1 Knowledge Base Evaluation Question 62
3.2 Manual Effort Times for the Initialization of the Knowledge Base 64
3.3 Rule Execution Times 64

4.1 Security Activity Categories and their respective Tools 74
4.2 Initial List of Common Data Fields 75
4.3 Common Data Model derived from Security Activity Reports 78
4.4 Selection of Activities for the Golden Dataset 84
4.5 Distribution of the Original Dataset across Activities 86
4.6 Performance Indicators of KG-Based, LSI, and SBERT Clustering 91
4.7 Reasons for the Incorrect Clustering, given by the Domain Experts 92
4.8 JSON Parameters in a Security Report Upload Operation 104
4.9 Research Questions for the Preliminary Quality Improvement process . . . 110

5.1 Conclusions based on Findings History 122
5.2 Orientation Scale for Prioritization Score 134
5.3 Glossary of ISO 27000 Terms in the Security Findings Management Context 140
5.4 Answers to the Interview Questions 1-3 of the Prioritization Evaluation . . 152
5.5 Development Principles Fulfillment for the Analysis and Tracking 152

6.1 List of Presentation in Visual and Text-based Form 161

7.2 Subject Project Properties 179
7.1 Recruitment Requirements for Projects 179
7.3 Interview Characteristics 182
7.4 Reoccurring Questionnaire 184
7.5 Development Indicators 185
7.6 Final Questionnaire 186
7.7 Quantitative Statistics per Week 190

A.1 Collected Documents 213

B.1 Aggregated DevOps Principles 216

D.1 List of Changes to the Security Flama during the Evaluation Period 219

E.1 Project A - Quantitative Data 221
E.2 Project B - Quantitative Data 222

Chapter 1

Introduction

1.1 Motivation

With the recently proposed Cyber Resilience Act, the European Union introduced yet another motivation for industrial enterprises and organizations to build products with a high level of security [45]. This regulation enqueues amongst multiple other standards, regulations, and norms that demand specific security properties from industrial products. In contrast to other voluntary norms, the EU CRA prevents organizations from entering the European market if they fail to fulfill the given cybersecurity demands. This approach is not distinctive to the European market but has been applied worldwide, affecting specific industry sectors, contractual partners, or locations [81, 185]. Hence, the security of industrial products evolved from being a quality criterion with limited business impact to the decisive factor for the success of a product and its ability to be commercialized. The continuously evolving threat landscape [169] and regular industrial security incidents with global media coverage like the recent SolarWinds breach [31] further reinforces the importance of security in industrial products.

The security properties of an industrial product are rooted in the activities conducted during development and its subsequent operation. Therefore, demands apply to the resulting product and the process employed during its development and operation. Standards like the IEC-62443-4-1 [70] state activities that must be conducted during the software development process to achieve compliance. Typical demands across all process-oriented standards include tests for specific security properties of the software product, including, e.g., publicly known vulnerabilities or insecure coding patterns. The resulting test reports are further required to prove compliance with the requirements of product-oriented standards like the EU CRA. However, the identification of these findings itself is insufficient to improve the security of a product. Instead, they must be looped back to actors that refine the respective aspect of the product [167] and treat the risk inherent to the issue. This so-called Feedback Loop and the subsequent management of security findings are process requirements similar to the security testing itself. Recommendations and demands towards the activities conducted during the security findings management can be found in

process-oriented standards, best practices, or industrial guidelines [26, 140, 158, 70, 134].

Not only has the status of security for industrial enterprises evolved over the last decades, but also the models and methodologies employed to develop software in the industry have changed [28, 18]. With a continuously increasing number of organizations and enterprises relying on modern software engineering models like DevOps or Agile, the propositions of software engineering have matured. Cross-functional collaboration, communication, and continuous delivery of results have become common principles in industrial software development projects [104, 34, 145, 65, 202, 51, 16]. Amongst others, these principles also affect the security activities conducted in each project. With test automation being a central consequence of modern software engineering [152], the integration of automated security tests is a crucial step for industrial enterprises [121]. As a consequence, the management of reports from automated security tests must also adhere to the fundamental principles of modern software development and its constraints. This introduces another layer of complexity to the existing challenge of managing security findings in industrial practice.

We believe that security activities, particularly the management of security findings, should be conducted with the same efficiency as other practices in modern industrial software development. Traditional, manual approaches do not conform with these modern software development principles, necessitating research in the domain of security findings management to implement and close the Feedback Loop for security findings. We further believe that solely a methodology minding the inter-discipline constraints of the software development and security domain can address these challenges. Consequently, this thesis explores how a methodology for managing security findings in modern industrial software development projects should be realized.

1.2 Problem Statement

A methodology for managing security findings in modern industrial software development projects has to cope with various challenges. Fundamentally, all challenges that apply to security findings management in general also apply when conducted within a modern software development setting. However, several additional problems are introduced by following modern software engineering principles. While the automation of repetitive tasks, including tests in particular, has multiple advantages [152, 121], it also introduces challenges to the subsequent management of reported security findings. The frequent execution of different tools results in large datasets with low data quality and various formats, claims, and values. Since the reported information potentially reveals security risks to the software product, they must be treated seriously, regardless of issues like False Positives, duplicates, or a lack of semantically rich information. Another problem is integrating such a methodology into modern software development projects. Minding the shortening of release cycles and work packages, not every security finding can be addressed. Moreover, the recipients of information from this process are not exclusively developers. Cross-functional teams necessitate a communication strategy that can be comprehended by security experts and

novices alike while providing each role with the information relevant to complete tasks. Finally, the methodology and its design must not only cope with the challenges introduced by modern software development but must adhere to its principles itself. Otherwise, the management of security findings infringes the efficiency of the overall software engineering process. Therefore, security findings management processes focusing on traditional processes like waterfall-oriented models can not support the demands of modern industrial software development projects. Instead, only a methodology solving the abovementioned challenges can bridge the gap between security and software engineering in this domain.

Dealing with the security shortcomings of a software product is a widely researched domain with various propositions ranging from methodologies and frameworks to tools and applications. However, most of the academic literature focuses on particular aspects of the software development lifecycle or a subset of all shortcomings that might affect the security of a product. Examples include frameworks focusing explicitly on the management of publicly known vulnerabilities [199, 48] or tools managing findings from static code analysis tools [170]. Consequently, methodologies that address security findings management, particularly in modern industrial software development projects, represent a gap in the current academic literature. In addition to literature, multiple tools and applications are proposed in practice, addressing similar challenges [37, 47, 170]. However, all proposed approaches lack one of three properties.

1. Security Findings are only considered from a fraction of the entire software development lifecycle

2. The principles of modern software engineering are disregarded in its design

3. The target audience only includes a fraction of the project team

While the first property is an issue, also commonly applicable to academic proposals, the second and third properties address the particularities of modern software development. Either the approaches focus on a single aspect of the entire software engineering process, hence lacking to interconnect the subsequent steps into a joint methodology that acknowledges the big picture of the security findings management process, or it is deficient in minding the surrounding development process and consequently failing to perform the security findings management with the same efficiency as other development practices. This affects, in particular, the target audience of the security findings management process, which is not restricted solely to developers or security experts. Consequently, we identified a gap in the existing State-of-the-Art and, hence, the necessity to investigate this problem to contribute a methodology for managing security findings in modern industrial software development.

To achieve this goal, the scope of this problem must be defined. First, we focus on the management of security findings in software development projects. This disregards, e.g., organizational-wide risk management or enterprise vulnerability management. Instead, the methodology shall focus exclusively on the security findings reported within one project. Aligned with the ISO standard on vulnerability analysis [73], we consider security findings

as any weakness to the security of a software product. In contrast to security issues, findings have been identified but are not yet confirmed or further processed. This implies that our methodology addresses security findings reported throughout the entire lifecycle of a software product. Moreover, we primarily consider projects employing modern software development principles and models. While the resulting methodology may also be usable for traditional development models like waterfall-oriented projects, its main focus is on models and their principles commonly considered modern in the industry. Finally, our methodology may not only consist of a theoretical concept but must be usable in industrial practice. This implies that industrial development projects following the principles of modern software development must be able to employ the methodology in practice to manage security findings.

Based on this scope, we derive five research questions for this thesis. Each consists of multiple sub-problems, discussed in their respective chapters. Fundamentally, the underlying problem must first be conceptualized and described. Hence, the requirements of security findings management in industrial software development projects must be identified and refined with the principles of modern software development projects. Based on the results, we identified four additional research questions. Initially, the methodology requires a platform that orchestrates and automates the security findings management process. Second, issues with the data quality must be corrected. Next, the findings must be managed by tracking, analyzing, and responding to them. Finally, the methodology must be integrated into ongoing industrial software development projects to evaluate its impact on industrial practice. In summary, the five research questions are:

- **RQ1** - Which requirements exist for the management of security findings in modern industrial software development projects?
- **RQ2** - How does a platform for the management of security findings in modern industrial software development projects look like?
- **RQ3** - How can flaws in the data quality of security findings reports be resolved?
- **RQ4** - How should security findings be tracked and analyzed in modern industrial software development projects?
- **RQ5** - How does the usage of the methodology impact modern industrial software development projects?

1.3 Research Methodology

In this thesis, we follow the fundamental target of software engineering research: Improving real-world practice [154]. Consequently, we research solutions that can address the challenges of security findings management in modern industrial software development, considering the security domain and the software engineering domain alike. This methodology represents the core contribution of this thesis as a solution to a challenge existing

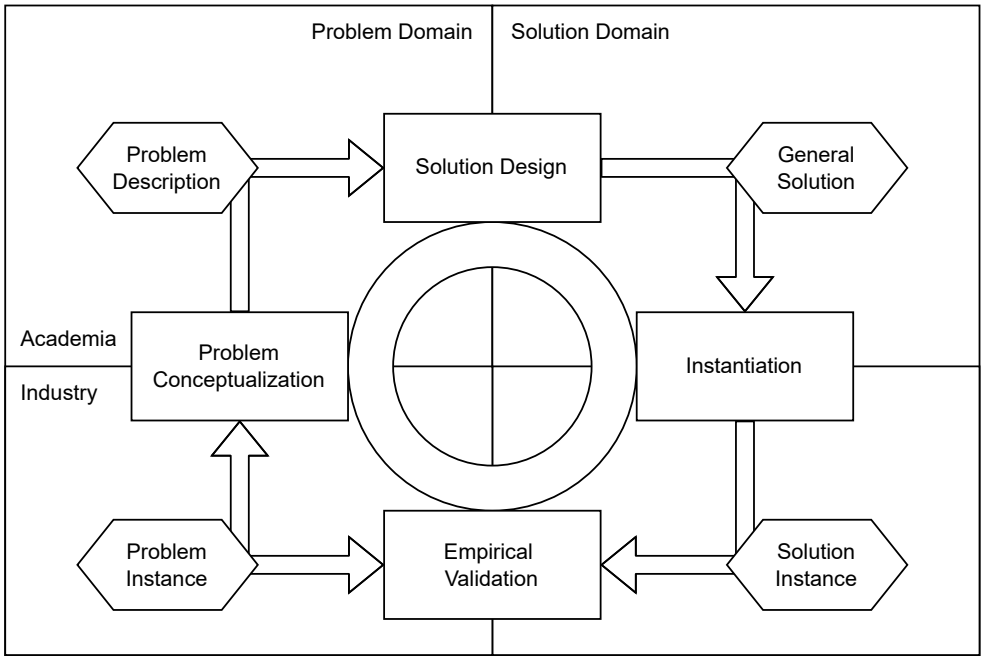

Figure 1.1: Visualization of the Design Science Research Approach

in practice. Our research aims to improve the existing, human-made processes of security findings management and seeks a solution to the current challenges in the problem domain, indicating the necessity of Design Science Research [172, 154]. Following this research strategy provides a framework for conducting empirical software development. Since our challenge arose from an industrial domain, we collaborated with Siemens AG, a multinational industrial enterprise, on this problem. With more than 300.000 employees and software development activities in industrial sectors, including industry automation, healthcare, mobility, and energy, this strategic partner provides insights into industrial practice and its challenges.

This thesis follows the Design Science research described by Runeson et al. [154]. Figure 1.1 depicts an abstract version of their Design Science concept. Research following this concept is motivated by *Problem Instance* existing in the industrial context. This instance is abstracted by *Problem Conceptualization* as a collaboration between practitioners and researchers. The resulting constructs describe the problem and the solution constraints. These constructs represent the baseline of information used during the *Solution Design* to transform the knowledge of the problem domain to the solution domain. This results in a theoretical *General Solution*. To validate the feasibility of the solution for the problem instance, it is instantiated (*Instantiation*) into a *Solution Instance*. Using *Empirical Validation*, this instance is evaluated against the original *Problem Instance*.

Following the Design Science approach by Runesone et al., we conceptualize the problem of security findings management in modern industrial software development approaches to identify problem constraints. Based on these problems, we design a general solution in

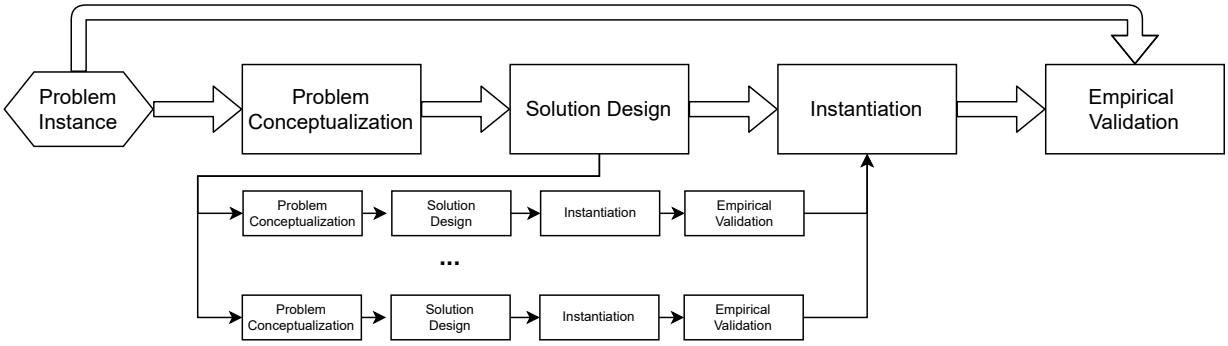

Figure 1.2: Design Science Iterations

the form of a methodology and instantiate it in the environment of our industry partner. Finally, we validate the feasibility of our solution for the problem instance. Due to the amount of challenges and the complexity of problems in this domain, we perform these stages iteratively for every problem constraint identified. This implies that the overall thesis approach uses Design Science principles. At the same time, each identified problem constraint is designed by a separate iteration, including a preliminary empirical evaluation depending on the addressed problem constraint. The general solution of each problem constraint is aggregated, and the resulting methodology is instantiated. Finally, this solution instance for the overall problem of security findings management in modern industrial software development is empirically validated with our industry partner. This approach is visualized in Figure 1.2.

The selected research approach necessitates a close collaboration with our industrial partner. For each challenge, we collaborate with the respective practitioners on the industry side to conceptualize the problems and evaluate our solution instance in their environment. Therefore, a collaboration over three years is established, including access to the industry partner's resources, as Wohlin recommended for successful software engineering research with industry [204].

In addition to the overall Design Science Research approach, we utilize different strategies for each constituent of the Design Science paradigm [154]. The research methodology used for each constituent is explained in the respective chapter of the thesis.

1.4 Contribution

This thesis contributes to two areas of research: the software engineering (SE) domain and the security (SEC) domain. Its main contribution is a methodology for managing security findings, minding the principles of modern industrial software engineering. The relevance of this contribution is represented by its impact on industrial practice, shown by an empirical evaluation. Each chapter of this thesis contains a separate contribution to the current state-of-the-art research by analyzing and solving domain-specific problems in the respective research area. The following paragraphs present this thesis's four most

significant contributions and summarize additional ones. If not explicitly stated, all authors of the mentioned publications conceived the initial idea, discussed the results, and contributed to the manuscript. However, the author of this thesis took care of the problem statement, design, implementation, evaluation, and formulation of the manuscript for each publication.

Semantic Knowledge Base for the Management of Industrial Security Findings

The first major contribution of this thesis is the design and implementation of a semantic knowledge base intended as a platform for managing security findings in industrial projects. Using the well-established concept of semantic knowledge bases and refining it to our use case by considering information as belief instead of facts, this customizable platform has proven to be a valid solution in our final evaluation. The fundamental concept of this contribution has further been published in [195].

Semantic Similarity-Based Clustering of Industrial Security Findings

The second major contribution of this thesis is the design, implementation, and evaluation of a semantic similarity-based clustering and aggregation approach to identify and eliminate duplicate security findings. Multiple language-based similarity algorithms have been assessed as part of this contribution, and Latent Semantic Indexing has been implemented as part of the solution approach. Integrated into an overarching process of data-quality improvement, this solution approach has proven to reduce the number of duplicate security findings and positively impact industrial software development projects. The comparison of different similarity algorithms was published in [161]. The contributions of all authors for this publication can be found in the respective chapter (Section 4.4). The evaluation in an industrial context as part of the overall solution approach has been published in [194].

Automated Prioritization of Industrial Security Findings

The third major contribution is an automated prioritization process for industrial security findings. Considering the importance of cross-functional collaboration, this solution approach unifies severity ratings from multiple sources and calculates a priority score depending on formalized inputs from stakeholders and environmental factors. This solution approach extends traditional methods for vulnerability scoring or prioritization by considering security findings from the entire lifecycle and computing a priority on a universal scale. Using expert interviews and industrial software development projects, the solution approach was evaluated and showed its correctness and relevance for industrial practice. This solution approach and its evaluation have also been published in [196].

Methodology for the Management of Security Findings in Modern Industrial Software Development

The fourth and last major contribution of this thesis is the methodology for the management of security findings in modern industrial software development projects, its implemen-

tation and evaluation. The methodology is based on the combination of requirements for the management of security findings and the principles of modern software development. It consists of all previous contributions and is implemented as a platform for managing security findings with a dedicated communication strategy towards the project. This instance of the methodology has been evaluated in two industrial software development projects and has indicated its usefulness and positive impact on industrial practice. The methodology and the fundamental results of the evaluation have been published in [193].

Additional Contributions
In addition to the above-mentioned major contributions, several further ones have advanced the State-of-the-Art. These include a common data model for security findings, the concept of Findings Prioritization Policies for formalizing actor input on the prioritization of findings, a state model for the security finding lifecycle, and a risk-based guidance for responding to security findings.

1.5 Thesis Structure

The format of Design Science Research is reflected in the outline of this thesis. Each chapter deals with at least one constituent of the Design Science approach. Those chapters focusing on the solution design depict the entire Design Science spectrum in a reduced format, by conceptualizing the respective problems (Problem Description) of the chapter, presenting related work and background in the domain (Related Work), designing a solution toward it, instantiating the solution approach (Implementation), and present a preliminary validation designed according to the problem domain. In the following paragraphs, each chapter of the thesis is described briefly.

Chapter 2 Chapter 2 represents the problem conceptualization for the overall thesis. This chapter contains the background and related work applicable to every chapter of the thesis. Furthermore, it systematically conceptualizes the problems of security findings management in modern industrial software development by studying the state of practice for industrial security findings management and mapping it to principles of modern software development. Finally, it formulates objectives for the remaining thesis following the Design Science Research approach.

Chapter 3 The first chapter, focusing on the solution design, is Chapter 3. In this chapter, we present the challenges and constraints for a platform, supporting the security findings management in modern industrial software development projects. Based on these challenges, we utilize the well-established concept of semantic knowledge bases and customize it to the given constraints. This novel design is implemented into practice and preliminarily evaluated using a dummy-project. The results of this evaluation indicate that our solution approach provides a flexible, sound, and fast platform that fulfills our requirements.

Chapter 4 In Chapter 4, we present the problems necessitating the improvement of data quality during the security findings management and the constraints affecting potential solution approaches. A potential solution approach is designed, including the collection of security report, their parsing to a common data model, the deduplication of security findings, and the enrichment of information transported by the security findings. With a focus on automating these procedures, the proposed approach is implemented into the security findings management platform. Employing a preliminary evaluation, we conclude that the solution approach covers all requirements while adhering to modern industrial software development principles.

Chapter 5 Chapter 5 proposes solutions to the predominantly manually conducted activities of tracking and analyzing security findings. Derived from the challenges in industrial practice, the approaches include a history and state tracking of findings, support for the analysis and documentation of security findings, and a formalized prioritization and response strategy aligned with common risk management standards. The solution approaches are instantiated as part of the security findings management platform and preliminarily evaluated. The results suggest that the proposed approaches support all initial requirements while respecting most modern software development principles.

Chapter 6 The last iteration of the Design Science Research cycle is presented in Chapter 6. The chapter categorizes and analyses the problems related to communicating security findings data and proposes a visual and text-based strategy for presenting security findings data. The strategy is implemented as a web interface for the security findings management platform and integrated with the existing platform implementation of previous chapters. In contrast to the other solution approaches, this chapter does not present a preliminary evaluation since the subsequent empirical evaluation covers all proposed solution approaches equally.

Chapter 7 Finally, Chapter 7 presents the implementation of all preceding solution approaches into one system and its evaluation in two software development projects. The implementation of the system and the preconditions for its integration into projects are described. Afterward, strategies for the evaluation are examined, and the final evaluation protocol is presented. The results of evaluating the methodology in two ongoing software development projects at our industry partner are shown and discussed. This empirical evaluation concludes our Design Science Research approach.

Chapter 2

Security Findings Management Problem Conceptualization

In this chapter, we present the background and the problem conceptualization for the remaining thesis. First, we present the related work for the entire thesis, providing an overview of the topic and its necessary background. Second, we analyze requirements for the management of security findings that currently exist in the industry. This represents the first half of the problem instance impacting practitioners in industry. This list is mapped in the next section with the principles for modern software engineering representing the second half of the problem instance, resulting in the problem description. Based on this problem description, the objectives of the thesis are constructed. The summary of the chapter and its results are discussed in Subsection 2.4.3.

2.1 Background, Terminology, and Literature

This section presents the background and terminology used throughout the remaining thesis. This excludes any background related exclusively to one particular chapter. The background or domain-specific language necessary for individual chapters is elaborated in the respective chapter. Furthermore, this section introduces literature relevant to security findings management in modern industrial software development.

2.1.1 Software Engineering Practices

The advancements of software engineering practices throughout the last decades define the fundamental knowledge necessary to understand the challenges of security findings management when conducted in modern industrial software development.

The term ”Software Engineering” is disputed and can roughly be traced to the second half of the 20th century [22, 203]. Even though the concept of writing programs existed previously, the emergence of more powerful computers and their availability to institutions made programming a practice for many [203]. Similar to the growth of computing capac-

ity, the need for programmers and the complexity of tasks they intended to solve grew. Amongst other developments, the principles suggested during this time resulted in the construction of software development models. One of the first and most influential was the waterfall-oriented model and its adaptions presented by Winston Royce [112]. This model followed a sequential approach in which the development practices are ordered according to their sequential execution in projects. A vital feature of this model states that one practice can only be started if the previous one is finished. However, the major drawback of this methodology is the complexity of grasping the complete problem at the beginning of the development. Hence, new models emerged dealing with this challenge by applying iterative approaches. Examples include the Prototype Model or the Spiral Model [112], both following the underlying idea of iteratively building new versions of software. Combining the incremental approach of the waterfall-oriented model in an iterative fashion introduced various new models [7, 112] and represent the strategy most recent software development models follow. Reaching the 21st century, software engineering models are often divided into classical or traditional methods and modern or agile ones [89, 118].

Agile methodologies, especially, have received significant attention from academia and industry alike. The fundamental propositions for Agile software development were published in 2001 as a manifest [17]. The manifest comprises four propositions, each indicating the relevance of one engineering aspect compared to another one. Moreover, 12 principles exist, providing general guidance for conducting activities during software development [16]. With a strong focus on a close collaboration between customers and developers, practitioners try to overcome the shortcomings of traditional, plan-based approaches by using Agile models [87]. Representatives for Agile methodologies include Scrum, Extreme Programming (XP), Pair Programming, or Lean software development [87], which all follow the same fundamental value propositions. Applying these practices to large-scale software development organizations introduces challenges ranging from communication issues to a lack of coordination [35]. Hence, many organizations turn to scaled versions of Agile methodologies, such as the Scaled Agile Framework (SAFe), Nexus, Scrum at Scale, or Large Scale Scrum (LeSS). However, Agile methods have one major drawback: they solely address software development and lack practices for subsequent software operation.

An approach trying to bridge this gap between development and operations by considering the entire software development lifecycle is DevOps. DevOps is regarded as an evolution of the agile movement [104], as a response to the challenges of large-scale software platforms [34]. In contrast to the Agile methodologies, DevOps is an ambiguous term with no universal definition [51, 145]. Instead, various suggestions exist in academia and industrial practice on the principles of DevOps [68, 93, 80]. Common core elements of DevOps include a culture of collaboration, automation of repetitive tasks, continuous measurement, and information sharing [145]. However, DevOps does not represent an alternative to Agile methodologies but complements it by suggesting solutions to known challenges [4]. The critical aspect for this thesis originating from using DevOps, Agile, or both in combination is the concept of Feedback Loops [174, 4]. The necessity for fast feedback emerging from DevOps practices [117] and communication of feedback to team members from Agile practices [4] reinforces the importance of looping information from its source to the respective

stakeholder. Both, the origin and target of the feedback, can be distributed across various lifecycle stages, necessitating loops between stages as well as between development and operations phases [93].

2.1.2 Threats to the Security of Software

When developing industrial software products, evading properties that might lead to a breach of fundamental security principles is crucial. Security principles refer in this context to the *Confidentiality*, *Integrity* and *Availability* [159] of software products and the data processed by it. The terminology in this domain has precise wording for the lifecycle of these security-critical properties and all implications around it. As different standards, papers, and references use multiple definitions for specific terms, we see the necessity of clarifying the terms for this thesis. In this subsection, we define each of these words to establish a common understanding of terminology.

The first term to define is the *Security Requirement*. A security requirement or security feature represents a requirement in the security domain. Similarly, it defines a specific functionality that ensures the fulfillment of security properties in the software [141]. This functionality can be entirely new or an extension to an existing feature. Security requirements are often derived from project external demands like standards, directives, or laws [127]. Another strategy is engineering new security requirements from the existing threat landscape surrounding the software product.

A *Security Threat* is any event or circumstance resulting in a negative effect on the fulfillment of the security principles [126]. In most cases, these threats only represent the potential cause of a damaging event [71]. Each threat is accompanied by an inherent *Security Risk*. It measures the extent to which an entity is threatened by a damaging event [126]. Typically, it is described as a combination of damage impact and the likelihood of the damage occurring [24, 126, 83, 74].

The actual occurrence of a damaging principle infringement is defined by the term *Security Incident*. Such an incident consists of one or a series of events actually or imminently threatening the security principles of a software product [126, 71]. A security incident is clearly distinguishable from day-to-day disruptions in the operation of a software product [24]. A term closely related to security incidents is the *Risk Incident*, which describes the actual manifestation of a security risk. Even though the term risk incident emphasizes the risk that accompanies it, both are interchangeable in practice.

These incidents mainly occur due to a malicious entity intentionally exploiting errors in the software product. Depending on the type of error, its cause, and its context the terminology is differentiated. The most prominent term for problems in the security of software is *Vulnerability*. A vulnerability is a security-relevant flaw or weakness in an IT system [24]. Whether the flaw can already be exploited or not depends on the source of definition [71, 126]. In practice, however, there is a significant difference between an error that exists in the software but has no effect on the security of the system and an error that can be exploited. Consequently, we favored a more granular distinction between vulnerabilities in these cases. We define the term *Security Finding* as any problem to the

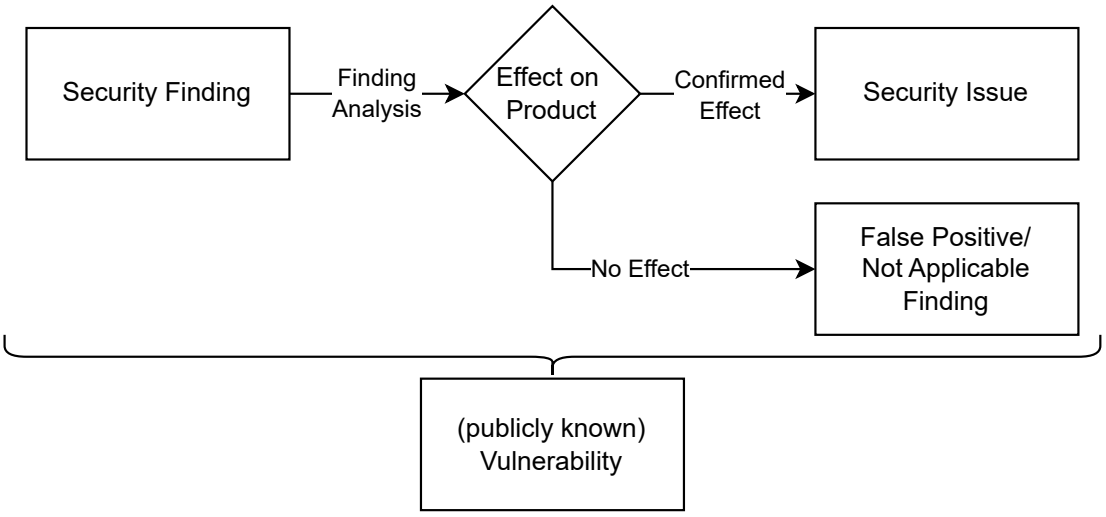

Figure 2.1: Core Terminology

security of a software product that is reported to be in a product. A security finding might be *False Positive* or *Not Applicable* to the product. If it indeed impacts a software product's security, we define it as a *Security Issue* [177]. Furthermore, we must distinguish between publicly known vulnerabilities and undisclosed ones. A publicly known vulnerability is accessible to the general public and typically has an identifier corresponding to it, in most cases a CVE-ID [178]. This *Common Vulnerability and Exposure* identifier allows a unique identification and precise communication about vulnerabilities. Known vulnerabilities are mainly accessible to the public in database-like structures provided by various institutions and organizations [125, 14, 149, 168]. On the other hand, undisclosed vulnerabilities are not known to the public but solely to a distinct group. This is typically the case if an organization identifies the vulnerability in any way (responsible disclosure, internal testing, etc.) but decides to keep it undisclosed for a certain period (see 7.1.3 of [77]. The root cause of a vulnerability can be any error, defect, or bug in the software, like implementation, configuration, or operation [24]. In particular, this includes errors in sub-components of the software product that separate teams or organizations develop. This circumstance results in a transitivity of security errors where suppliers can introduce new vulnerabilities into vendor products. Our explanation further introduces the terms *Security Flaw*, *Security Weakness*, *Security Bug*, and *Security Defect*. As we will not use these terms in the thesis, we are not defining them in this section.

In this thesis, we will be predominantly using the terms *Security Finding* for any error not yet analyzed, *Security Issue* for a finding with confirmed impact on the product, and *False Positive / Not Applicable Finding* for any finding with no impact on the software product. *Vulnerability* will be used as an umbrella term for any security-related error. These relations are depicted in Figure 2.1.

2.1.3 Security Findings Management

The development of software in industrial organizations has to comply with a variety of standards and guidelines. These do not only affect the product itself, like the recently proposed Cyber Resilience Act for the European Union [45] but can also shape the software development process itself. Multiple security standards, maturity models, and best practices guide organizations in designing secure software development lifecycles and include activities in their process that contribute to the security of the product itself [26, 140, 158, 70, 134]. These activities are intended not only to improve the product's security directly but also to inspect the product for shortcomings. These activities test the security-related properties of the product and report them. Minding that the automation of repetitive tasks, including tests in particular, is one of the core principles of modern software engineering [152], multiple tools exist that automate security tests.

Angermeir et al. identified five types of testing tools employed in enterprise-driven Open Source Software [11], while Github notes almost 4000 repositories that are related to the topic of "security tools" [54]. As discussed in the last subsection, the focus of these tools ranges from checks on the code of the software for poor coding patterns over publicly known vulnerabilities to the malicious behavior of the software. To integrate these tools with the existing development workflow, they are often automatically executed as part of a CI/CD pipeline [108, 148, 146]. However, the initially mentioned security standards not only require these testing activities to be conducted but also require management and response to the shortcomings reported by these tools.

Academia and industrial practice provide a variety of strategies for dealing with findings reported by security activities. In particular, the area of vulnerability management has been well-researched and applied in practice for decades. Research ranges from vulnerability management on an enterprise level to the scope of a single project applied in various domains and supported by guidance, frameworks, and methodologies [180, 136, 199, 48]. Concepts like vulnerability identifier [178], databases of known vulnerabilities [125, 14, 149, 168], lists of weakness types [179] or common severity ratings for vulnerabilities [135] support practitioners during their work. However, the management of vulnerabilities and its supporting material mainly relates to publicly known vulnerabilities or vulnerabilities that are found during operations phases instead of the entire lifecycle. Hence, it covers only a fraction of all sources of security findings. Another commonly employed approach is the usage of bug, defect, or technical debt tracking. In these cases, the reported security findings are treated as a bug, issue, defect, or technical debt and managed as such during the regular process [151, 142, 183]. Since security findings still differentiate from the traditional definition of these other artifacts, including their treatment [208] and contain inherent challenges like False Positives that must be sorted out before being considered as actual bugs or defects, the management of security findings must be initialized before any issue tracker can be employed.

Knowing this challenge, suggestions exist in practice on how to manage security findings throughout their entire lifecycle. Following the principles of modern software development, these are primarily automated and provide a visualization of the reported security find-

ings. The most prominent tool in this domain is *DefectDojo* [37]. The tool is defined by the OWASP foundation as an "open source vulnerability management tool" and supports various types of security test reports. As part of their management activities, they support the deduplication of findings and integrate them with downstream systems like Jira. However, OWASP identifies that "the top goal of DefectDojo is to reduce the amount of time security professionals spend logging vulnerabilities" [139], also representing the challenge of using this tool. Its primary user group consists of security professionals instead of cross-functional project teams. Further challenges in the state of practice include tools solely focusing on specific assets like networks or collecting findings exclusively from particular stages like operations. An example would be Faraday, which identifies and collects security vulnerabilities during operations phases and supports the management of all reported findings [47]. Furthermore, many security tools already provide some interface for managing all findings reported by the tool itself. Specific tools, like Sonarqube [170], even support the management of findings that other tools or frameworks have identified. However, the primary objective of these platforms is the management of security findings identified by the tool itself, often resulting in reduced functionality for externally reported security findings. Therefore, multiple approaches for managing security findings exist in practice, which, however, either lack the coverage of the entire software development lifecycle, disregard modern software development principles, or target an incorrect audience for modern industrial software development.

2.2 Requirements Towards the Industrial Security Findings Management Process

The management of security findings is a crucial stage in the secure development of software products in the industry. Consequently, multiple security standards and industrial best practices not only state the necessity of a security findings management process but also elaborate on the different tasks it should cover. To develop a methodology for security findings management in modern industrial software engineering, identifying the fundamental problems and challenges represents our first step. Consequently, we begin our research by investigating common requirements for security findings management in industry. To identify the industry demands, we perform a systematic document review on standards, specifications, and best practices currently in use at the industry partner, as well as standards and similar regulatory documents commonly employed in the industry. Even though we do not perform a traditional literature review, we follow recent publications for the analysis protocol [98].

2.2.1 Document Collection Preparation

In advance to the actual review, we discuss its design and planning in this paragraph. The primary goal of our research is to identify commonly accepted requirements for managing

security findings in industrial software development projects. More precisely, we want to focus on those requirements industrial practitioners would face when trying to implement a security findings management process into their projects. Based on this research goal, we derived the following research questions:

- **RQ1**: Which documents would be classified as relevant by an industrial practitioner trying to find requirements for the security findings management in industrial practice?
- **RQ2**: Is there a category of documents more valuable for practitioners when identifying requirements for security findings management in industrial practice?
- **RQ3**: Which requirements are contributed by the identified documents?
- **RQ4**: Which list of unique requirements can be aggregated over all identified documents?

To answer our research questions, we collect documents with a strategy similar to an industrial practitioner and perform an in-depth analysis of all identified documents. A strategy focusing on reviewing scientific publications would not be coherent with this goal as the collected knowledge would not be in our area of interest. Consequently, we separate our document collection into two streams. First, we request a list of relevant documents from our industry partner, representing documents used in practice and, therefore, certainly in the area of documents a practitioner would be confronted with. Second, we identified documents considered substantial for security in industrial software engineering. This implies that we do not rely on any particular search string or search engine for our document collection. The term document refers, in this case, to any type of framework, model, standard, best practice, or guideline that comprises input on the security of industrial software engineering. Since requirements towards the management of security findings could be part of documents with various scopes, we did not limit the document selection solely to the domain of secure software development but considered any up-to-date document related to security or software development.

However, not all documents are considered relevant for our review. As we aim to achieve a high relevance with our list of requirements (RQ4), any document solely applicable to one specific domain is excluded. This covers, e.g., documents only relevant in one country (South Africa, UK) or for products in a particular domain (Healthcare, Card Payment). Moreover, this restriction eliminates any company internal documents from our industry partner, as it only applies to our partner's corporate environment. Further exclusion criteria comprise any non-English document, documents that are not accessible with reasonable effort, and documents already considered. To ensure all documents still impact the industrial practice, we excluded all documents that do not impact practice yet or anymore. Hence, we consider the latest stable version for each document. This excludes draft versions or documents that have already been withdrawn.

2.2.2 Document Analysis Protocol

To answer research questions three and four, the list of documents collected during the last stage must be analyzed in-depth. For our analysis, we focus predominantly on requirements for the management of findings arising during the software development lifecycle. Consequently, organizational-wide guidelines or security program standards provide no input for practitioners. In particular, this excludes requirements to develop policies for the security findings management ("Security findings must be managed") or the documentation of the security findings management process ("The process how security findings should be managed must be documented and accessible to the project team") as this represents a higher abstraction layer than the actual management process. Furthermore, requirements towards an organisational-wide vulnerability and risk management are omitted as well. While these requirements address findings management, they focus on investigating new vulnerabilities and their applicability to the entire portfolio of an organization instead of the findings management for a particular software product. Hence, they exceed our scope.

A requirement is considered relevant to the security findings management process whenever it demands specific actions after following the identification of a security findings. This excludes the generation of findings by security tests, assessments, or similar as they are considered requirements towards a preceding step. We focus on actionable process steps, as we are interested in the security findings management process in practice. Hence, mainly requirements for the process in terms of its design or its stages are considered relevant. This excludes, e.g., all documents solely consisting of examples for security findings management or definitions on the topic.

Whenever a relevant requirement is identified, its content, the identifier within the document, and a reference to the document are recorded. Since multiple documents might demand the same or similar requirements, our analysis will be challenged by duplicates in the final list. The fourth research question demands a list of unique requirements, excluding duplicates. Consequently, an aggregation and deduplication of every identified requirement is necessary. To avoid these duplicates, we determine the differences between each new requirement and the existing list. Whenever the difference between the content of both requirements is not substantial, they are summarized. Our definition of a "substantial" difference is driven by the scope of this review. We want to investigate the state of practice for the security findings management in industry. Consequently, each requirement should represent a self-contained practice representing a single building block for the overall process. Therefore, a substantial difference is present if the fundamental meaning of the requirement would be changed or the action required by it altered. In cases where no substantial difference can be identified, the respective requirements are summarized. This summary concatenates the identifier within the document with the reference to the document and unionizes the content of both requirements. This ensures that each requirement can be traced to every occurrence in our list of investigated documents.

The entire protocol, starting from the document identification and resulting in the list of aggregated requirements, can be found in Figure 2.2.

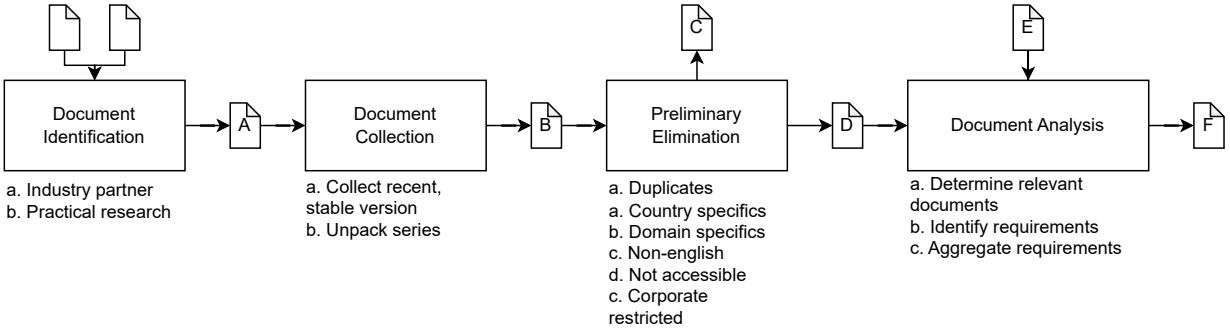

Figure 2.2: Review Protocol

2.2.3 Document Collection

During the first stage of the document collection, we requested documents from our industry partner using the following request:

> Could you provide us with a list of all documents a practitioner would be accessing when looking for requirements towards managing security findings in software development projects in your company

The response included 28 documents or document series. In addition to the documents provided by our partner, the demands arising from guidelines, standards, and best practices commonly accepted in industry were acquired. Since security standards are a crucial component of cybersecurity in industry, we considered them predominantly to establish common practices across all domains of IT systems [175, 182]. The initial selection of documents already confronts practitioners with the first challenge due to the vast number of supporting documents and whitepapers with varying reliability and relevance for the use case. Solely the United States recognizes 48 entire organizations developing standards [10]. Additionally, considering country-specific regulations or directives for projects in specific industry sectors, like software for healthcare, challenges the creation of a comprehensive list of all documents by practitioners. For this selection, we decided to follow two recent publications, focusing on common standards, maturity models, and best practices related to cybersecurity and software development [175, 147]. With this selection, we aim to achieve a high relevance and reliability level while maintaining a manageable amount of documents. Those publications provided us with 18 and 9 document/-series, respectively, resulting in 55 documents or document series (A = 55 in Figure 2.2).

This list references several series of documents, instead of a single document. We considered all documents in the series for our analysis in these cases. For instance, this resulted in nine documents being added to the list as they are covered by *IEC 62443*. Resolving all document series, we derived a list of 539 documents (B = 539 in Figure 2.2). Following our exclusion criteria, we cleared 254 duplicate documents. From the remaining entries, one document was removed, as it could not be accessed. Further 11 documents were scoped to one particular domain, and one solely affected a single country, resulting in

their removal. Our partner included one internal whitepaper and two internal wiki pages, resulting in three additional documents to be removed (C = 270 in Figure 2.2). The final list of collected documents consists of 269 entries (D = 269 in Figure 2.2). We decided to replace the documents with their most recent successor at the time of the analysis instead of the initially referenced and potentially outdated version. This ensures compliance with the current State-of-the-Art. Exemplary for this update is the originally referenced *FIPS 140-2*, which was updated for our use case to the 2019 published *FIPS 140-3*. All considered documents and their origin can be found in the Appendix Chapter A in Table A.

2.2.4 Document Analysis

Beginning with the analysis, we first eliminated all documents unrelated to our topic. This step removed most documents from our initial list since their scope differentiated from our use case. This step excluded, e.g., supplementary documents, like *IEC 61508-4:2010*, presenting definitions and abbreviations for the IEC 61508 series, or guidelines intended for the assessment of standard compliance like *ISO/IEC TS 27008:2019*.

The resulting list of documents potentially contributing to our goal was studied to identify requirements for the security findings management. However, this step already necessitated an extension of the existing list of documents. Some documents cross-referenced additional documents for a more detailed description of particular processes or requirements. Whenever this affected a requirement or process connected to the management of security findings, a gap in the completeness of our analysis arose. Exemplary for this circumstance is the regulation "8.8 Management of technical vulnerabilities" of ISO 27002:

> ISO/IEC 29147 provides detailed information on receiving vulnerability reports and publishing vulnerability advisories. ISO/IEC 30111 provides detailed information about handling and resolving reported vulnerabilities. [79]

This extended our list by *ISO 29147*, which covers the disclosure of vulnerabilities in products and services, and *ISO 30111*, dealing with vulnerability handling. Any cross-referenced document already in scope (e.g. BSI/COBIT to ISO 27001) was not added to avoid the introduction of duplicates. This extension added four additional sources (E = 4 in Figure 2.2). Finally, we derived requirements for managing security findings from this list. The results of this analysis can be found in the next paragraph.

2.2.5 Results

The results of our analysis are twofold. The first contribution is a list of 21 documents that we found to contain requirements for the security findings management. These documents are listed below with their contribution:

1. BSIMM 12 (15) [26]
2. OWASP SAMM (15) [140]
3. Safecode (14) [158]
4. IEC 62443-4-1 (13) [70]

5. NIST SP800-218 (11) [134]
6. NIST SP800-53 (10) [131]
7. ISO 30111 (9) [77]
8. COBIT (8) [82]
9. NIST SP800-160 (7) [133]
10. ISO 15408-3 (6) [78]
11. NIST SP800-181 (6) [132]
12. ISO 27002 (5) [79]
13. NIST CSF (4) [128]
14. BSI Grundschutz (4) [24]
15. ISO 27005 (2) [74]
16. ISO 29147 (2) [75]
17. IEC 62443-2-4 (2) [70]
18. IEC 62443-3-2 (2) [70]
19. ISO 15026-4 (1) [72]
20. NIST SP800-37 (1) [129]
21. IEC 62443-2-1 (1) [70]

Even though Common Criteria also contained requirements, we disregarded it as a source due to its similarity with ISO 15408. Since all documents listed above contain demands to the security findings management, a practitioner would consider them as relevant when searching for such requirements. Hence, the list of documents represents the result for **RQ1**. The documents can be predominantly categorized into security standards, maturity models, and guidelines. Considering the documents with the highest contribution, we conclude that maturity models and software development-focused security standards provide noticeably more input than other document types. Hence, documents of these categories provide more value to practitioners trying to identify requirements towards the security findings management in industrial practice, representing the answer to **RQ2**.

The second contribution is a list of requirements for the security findings management. In summary, we identified 185 requirements aggregated into 35 distinct requirements (F = 35 in Figure 2.2). The exhaustive list of all identified requirements can be found in Table 2.1.

ID	Title	Description	Lifecycle	#M	#R	Standards
1	Response Decision and Planning	The planning for a response includes: 1. Analyse all potential finding reactions including avoid, modify, share, mitigate, transfer and accept. 2. Decide for a response taking e.g. cost effectiveness, schedule, performance, and security risks into consideration 3. Plan the response to the finding 4. Document the response	Finding Response	19	11	COBIT: APO12.02, BSI: IND.1.A12, NIST SP800-218: RV.2.1, NIST SP800-218: RV.2.2, NIST SP800-160: QA-5.4, NIST SP800-181: T0264, NIST SP800-181: T0466, NIST SP800-181: T0579, NIST SP800-181: T0955, NIST SP800-181: T0076, NIST SP800-181: T0550, NIST SP800-181: T1006, NIST CSF: ID.RA-6, NIST CSF: RS.MI-3, IEC 62443-4-1: DM-4, ISO 27002: 8.8, ISO 27005: 9, ISO 30111: 7.1.5, ISO 15408-3: ALC_FLR.1.3C

2	Collection and Management	Security findings must be gathered from the respective sources identifying them and [centrally] managed. The sources include Development as well as Operation, internal sources (testing) as well as external (responsible disclosure, Vulnerability Databases, users),	Collection/ Processing	16	10	NIST SP800-218: RV.1.1, BSIMM 12: CR1.6, BSIMM 12: CR3.2, BSIMM 12: CMVM1.2, BSIMM 12: CMVM3.7, COBIT: EDM03.02, COBIT: APO12.01, COBIT: DSS05.07, NIST SP800-218: PO.4.2, NIST SP800-160: MS-2.2, IEC 62443-2-4: SP.03.03 BR, IEC 62443-4-1: DM-1, ISO 27002: 8.8, ISO 27005: 8.2.5, Safecode: Manage Security Findings, OWASP SAMM: DM1
3	Documentation	Security findings and their respective outcomes must be documented and stored. The data must be stored securely, protected from alteration or deletion and solely accessible by authorized personal. This is also required by multiple standards for security testing, as the results from security testing should be stored, regardless of whether the tests passed or failed.	Collection/ Processing	19	9	COBIT: APO12.01, BSI: APP.3.1.A22, BSI: APP.3.2.A16, BSI: OPS.1.1.6.A14, BSI: OPS.1.1.6.A5, BSI: OPS.1.1.6.A12, NIST SP800-218: PW.7.2, NIST SP800-218: PO.4.2, NIST SP800-53: SA-10, NIST SP800-218: PW.8.2, NIST SP800-53: SA-15, NIST SP800-160: MS-2.2, NIST SP800-160: QA-4.2, NIST CSF: ID.RA-1, IEC 62443-3-2: ZCR 5.2, IEC 62443-3-2: ZCR 5.13, Safecode: Manage Security Findings, OWASP SAMM: SB3, OWASP SAMM: DM1
4	Solution/Action	Solve security findings in a timely matter. The solution does not cover mitigation, but risk acceptance for example as well	Finding Response	12	8	NIST SP800-181: T0485, NIST SP800-181: T0954, NIST SP800-181: T1007, NIST CSF: RS.MI-3, IEC 62443-2-1: 4.3.4.5.10, IEC 62443-4-1: SM-11, IEC 62443-4-1: DM-4, ISO 27002: 8.8, ISO 30111: 7.1.5, ISO 27002: 8.28, ISO 15408-3: ALC_FLR.2.6C, Safecode: Manage Security Findings
5	Process Tracking	Track identified security findings throughout the fix process, to ensure the closure of the loop. Measure performance indicators like time to closure. Track and document beyond mitigation.	Process	10	7	BSIMM 12: CMVM2.2, NIST SP800-218: PO.4.1, NIST SP800-53: SA-10, NIST SP800-53: SI-2, NIST SP800-160: QA-5.7, IEC 62443-4-1: SM-11, IEC 62443-4-1: SI-1, IEC 62443-4-1: DM-1, ISO 15408-3: ALC_FLR.1.2C, Safecode: Manage Security Findings
6	Verificaton & Validation	Verify and Validate findings and security-related data for their applicability. Invalid findings could include: Duplicate, Obsolete Finding, No security relation (e.g. Linter), Not applicable (False Pos.)	Collection/ Processing	6	6	NIST SP800-160: MS-2.2, NIST SP800-181: T0347, IEC 62443-4-1: DM-2, ISO 27002: 8.8, ISO 30111: 7.1.4, OWASP SAMM: DM1
7	Analyze findings for their risk	Analyze findings for their risk	Collection/ Processing	6	6	COBIT: APO12.02, NIST SP800-218: RV.2.1, NIST CSF: ID.RA-5, IEC 62443-4-1: DM-3, Safecode: Manage Security Findings, OWASP SAMM: DM3

8	Feedback to Management/ Overseer/Decision Making	Ensure that information is looped back to those responsible for decision making. This includes in particular ISOs/CISOs. Escalate findings on demand. Ensure that the information contains at least: Consequences/Impact of finding in practice, Worst-Case/Average Case scenarios	Finding Response	9	6	COBIT 2019: EDM01.02, COBIT: APO12.04, BSI: APP.3.1.A22, BSI: APP.3.2.A16, NIST SP800-181: T1006, NIST SP800-37: R-5, NIST SP800-37: M-5, Safecode: Manage Security Findings, OWASP SAMM: DM2
9	Feedback to Stakeholder	Report risks and findings securely to the appropriate party/stakeholder	Side Effect	8	6	COBIT: EDM03.02, COBIT: EDM05.02, COBIT: APO12.04, NIST SP800-160: QA-5.6, NIST SP800-181: T0545, IEC 62443-2-4: SP.03.03 RE(1), IEC 62443-3-2: ZCR 5.13, OWASP SAMM: DM2
10	Analyze for more Occasions	Review the code for all instances/related instances of an identified software bug. Not just fix the identified one, but review all code for it, while not waiting for additional external reports.	Side Effect	6	5	BSIMM 12: CMVM3.1, NIST SP800-218: RV.1.2, NIST SP800-218: RV.3.3, IEC 62443-4-1: DM-3, ISO 30111: 7.1.4, OWASP SAMM: DM2
11	Analyze Findings in General	Analyze and classify findings	Collection/ Processing	6	5	BSI: OPS.1.1.6.A14, NIST SP800-53: RA-5, NIST SP800-160: MS-2.2, NIST SP800-181: T0710, NIST SP800-181: T1007, IEC 62443-4-1: SI-1
12	Vulnerability Disclosure	Loop the results to enable the disclosure of vulnerabilities in software (in operation/at customer): to users of the system, to the public, other stakeholders	Finding Response	6	5	BSIMM 12: CMVM3.6, IEC 62443-4-1: DM-5, ISO 29147: 7, ISO 15408-3: ALC_FLR.3.6.C, Safecode: Manage Security Findings, Safecode: Vulnerability Response and Disclosure
13	Feedback to Finding Response	Enable a correct findings response by providing the necessary information to the respective group. This typically includes development/engineering teams to improve the state of security. This covers security findings from operations as well as development.	Finding Response	6	5	BSIMM 12: PT1.2, BSIMM 12: CMVM1.2, NIST SP800-218: RV.2.2, NIST SP800-53: SA-11, Safecode: Manage Security Findings, OWASP SAMM: DM2
14	Aggregate and Correlate	Aggregate and Correlate the data to provide useful information to the development team. This includes in particular deduplication. Furthermore, identify how findings influence each other organization wide.	Collection/ Processing	5	4	BSIMM 12: CMVM1.2, COBIT: APO12.01, NIST SP800-53: RA-5, OWASP SAMM: DM1, OWASP SAMM: DM3
15	Prioritize	Prioritize findings from security tests and triage them	Collection/ Processing	5	4	NIST SP800-218: PW.7.2, NIST SP800-218: PW.8.2, ISO 30111: 7.1.4, Safecode: Manage Security Findings, OWASP SAMM: DM1
16	Root Cause Analysis	Identify the root cause of vulnerabilities	Collection/ Processing	4	4	NIST SP800-218:RV.3.1, IEC 62443-4-1: DM-3, ISO 30111: 7.1.4, Safecode: Vulnerability Response and Disclosure
17	Feedback to other Projects	Share the security findings with other projects to eliminate similar vulnerabilities in other systems	Side Effect	4	4	NIST SP800-53: RA-5, IEC 62443-4-1: DM-4, Safecode: Vulnerability Response and Disclosure, OWASP SAMM: DM2
18	Feedback to Lifecycle in General	Update elements of the software development lifecycle gained during management	Side Effect	4	4	NIST SP800-218: RV.3.4, ISO 30111: 7.1.7, Safecode: Vulnerability Response and Disclosure, OWASP SAMM: SR2
19	Verify finding closure	Check the finding has been solved using scanning tools	Process	3	3	ISO 27002: 8.8, ISO 30111: 7.1.5, Safecode: Vulnerability Response and Disclosure

20	History Tracking of Finding Occurrence	Track the history of all findings, allowing analysis on historical trends and shortcomings.	Collection/ Processing	3	3	BSIMM 12: CR1.6, NIST SP800-53: SA-10, ISO 29147: 6.2.4
21	Feedback to System Documentation	Document leftover security findings in a system documentation, notifying personel about known vulnerabilities in the system.	Finding Response	4	3	NIST SP800-53: SA-5, ISO 15408-3: ALC_FLR.1.1C, ISO 15408-3: ALC_CMS.4.1C, Safecode: Manage Security Findings
22	Determine finding circumstance	Determine the circumstances under which findings have been identified	Collection/ Processing	2	2	COBIT: APO12.01, IEC 62443-4-1: DM-3
23	Identify Correlations over time	Analyze findings to identify correlations between them, e.g. a coding practice being not followed. Perform lessons learned based on the correlations. Try to identify trends in vulnerabilities and the existence of multi-vulnerability exploits.	Process	2	2	NIST SP800-218: RV.3.2, NIST SP800-53: RA-5
24	Review of Open Findings	Review unsolved findings. Remaining security and compliance risks must be signed off and accepted, before shipping the software.	Process	3	2	IEC 62443-4-1: DM-4, BSIMM 12: SM2.6, BSIMM 12: CP2.2
25	Feedback to KPI	Monitor the success of the process by measuring e.g., speed, completeness and effectiveness	Side Effect	2	2	ISO 30111: 7.2, OWASP SAMM: DM2
26	Feedback to Training	Include the company history and a role specific curriculum to the participants. Includes e.g. data from pentests and noteworthy vulnerabilities.	Side Effect	4	2	BSIMM 12: T2.8, BSIMM 12: T2.9, BSIMM 12: CR1.6, OWASP SAMM: DM2
27	Feedback to Design Patters	Failures in TM, design review or analysis should be looped back to security and engineering teams to develop improved design patterns. Example: Findings on insecure infrastructure code can provide insights on insecure design patterns and indicate better ones	Side Effect	2	2	BSIMM 12: AA3.2, NIST SP800-53: SA-15
28	Parsing	Combine security assessment results into one common format used along the reporting. The consideration of different terminologies in vulnerability information is essential.	Collection/ Processing	1	1	BSIMM 12: CR3.2
29	Finding Enrichment	Each finding must contain at least a certain set of information, including: Effect of Flaw, Nature of Flaw	Collection/ Processing	1	1	ISO 15408-3: ALC_FLR.1.2C
30	Feedback to Assurance Cases	Feed statistics about vulnerabilities to validate assurance cases	Side Effect	1	1	ISO 15026-4
31	Feedback to QA	Share the results of security tests or bugs in Operation with the Q&A team. This provides an up-to-date overview and allows security tests to be tailored to the code/software	Side Effect	2	1	BSIMM 12: ST2.4, BSIMM 12: CMVM3.2
32	Feedback to Pentesters	Loop the results from testing, code review, analysis etc. to the pentesters, so they have all information relevant to analyse the application in depth	Side Effect	1	1	BSIMM 12: PT2.2
33	Feedback to Hotspots	Identify Hotspots that require attention	Side Effect	1	1	OWASP SAMM: DM2
34	Feedback to Policy	Create and refine policies from security lifecycle information. This includes reoccurring vulnerabilities, weak coding patterns, ignored security defects etc.	Side Effect	1	1	BSIMM 12: CP3.3

35	Feedback to Top N Bugs	Maintain a list of the most important bugs, that should be eliminated from the company. This list should incorporate real data from the companies, refined by additonal constraints as solely the number of occurences is insufficient.	Side Effect	1	1	BSIMM 12: CR2.7

Table 2.1: Requirements towards the Industrial Security Findings Management

The table presents each finding with its title (Title) and description (Description), containing information about what to do to fulfill the requirement. Furthermore, each entry is assigned a requirement type, clustering them into classes (Lifecycle). The number of appearances of the entry throughout all documents is listed (#M), as well as the number of documents referencing the requirement (#R). The number of mentions (#M) is always higher than the amount of referenced documents (#R) since one requirement might occur multiple times in the same document. These cases occur when the difference between the two requirements is not substantial. Exemplary are requirements that request a timely reaction to a security risk originating from different sources ("React to findings from Static Testing/Dynamic Testing"). According to our definition, both are considered findings, even though the document differentiates between them. Finally, the table lists all occurrences of each entry in the analyzed documents. This table further represents the result to **RQ3** and **RQ4** as it depicts the identified requirements as a list of unique entries.

2.2.6 Discussion

The results of our document review provide us with a list of requirements to be fulfilled in an industrial security findings management process. In the following paragraphs, we will discuss the results and their applicability.

The answers to research questions one and two did not show any anomalies. As expected, we primarily identified standards that contribute to the security findings management in industry, which can be tracked to our initial selection of documents consisting mainly of standards. Furthermore, those documents that contribute the most are also those that focus on software development. Since most security findings will occur during the development (low maturity of software security), their management is crucial in these documents. This also indicates that these categories of documents present a beneficial source if further information is required throughout the thesis. However, the answers to the third and fourth research questions provided several insights. The first striking aspect in our list of requirements is the frequency of requirements that request feedback to other processes or entities. Fifteen of our requirements request that certain information about the findings or the process status is provided to additional processes or entities within the organization. Consequently, we conclude that information communication is a core aspect of security findings management. Furthermore, our analysis showed that the requirements in documents differ in terms of coverage and precision. While some documents define specific actions ("Use findings to improve security trainings"), others employ vague

formulations ("Update elements of the development lifecycle"). This low level of precision in certain documents poses another challenge for less experienced practitioners. Another result from the analysis is the identification of a security findings lifecycle. The concept of feedback is not the only similarity between multiple requirements. Instead, we found that requirements can be categorized into collection/processing of findings, finding response, side-effects, and demands towards process execution. The core activities of this lifecycle are represented in the most referenced requirements. Both the documentation of findings and their collection and management describe the processing, while response planning and execution refer to the direct finding response. Finally, the tracking of findings is aimed towards the entire process. Solely, the side effects are not represented in this list. The distinctive nature of their demands can explain their limited visibility, as they are often just covered by a single standard. This mapping between requirements and their lifecycle category is documented in the "Lifecycle" column of Table 2.1.

A vital disclaimer to the outcome of this section is the different interpretations of compliance assessing authorities. The requirements presented in this section mainly focus on the fundamental demand, excluding interpretation by the assessing authority. Furthermore, the requirements contributed by certain documents are biased by the scope of the respective standard. This means the responsibility to fulfill the requirements lies with architects, developers, project managers/owners, or the organizations, depending on the document it originates from. Another challenge during the analysis were document updates. Multiple documents received a more recent version between the document review and its documentation. Consequently, our conclusions represent a snapshot of the current practice. Moreover, our results solely represent a fraction of all requirements that might exist. As discussed in the planning phase, we focused on a collection strategy based on a practitioner's search pattern. This strongly limited the coverage of standards. However, comparing this list of documents with the list provided by our industry partner, we can only identify a marginal gap. Hence, we believe in the correctness of our initial selection.

Finally, we must determine the scope of our results for future research and the rest of the thesis. Our requirements present a current snapshot of practices demanded by documents commonly accepted in industry. Hence, they indicate the existing knowledge in practice and its expectation on how security findings should be managed. This does neither cover the practices that are most commonly performed in industrial software development projects nor can conclusions on the impact on software security by following these requirements be drawn. Therefore, no proof is given that adhering to the practices benefits the security of a software product. Consequently, there might be other approaches for managing security findings, producing more secure products compared to our requirements. Exemplary for this are thousands of open source projects not officially following any standard or guideline while still producing secure software. However, the practices established within these projects are often subsets of our list. OpenSSL, for example, collects findings through multiple sources, investigates and prioritizes each finding, solves them, and communicates the solution to the public [137]. All these practices are part of our requirements list and can be mapped to our "Lifecycle". Hence, our list also seems to cover the crucial management activities for projects outside of industrial development.

Minding the scope of this thesis, the knowledge about requirements existing in practice is sufficient to identify challenges when managing security findings in projects that employ modern software development approaches. As every project and industrial enterprise sets different requirements for security findings management, no generalized list can be established. Hence, we consider the requirements in Table 2.1 as the baseline of practices to be fulfilled when managing security findings in industrial software development projects.

Key Takeaways from this Section:

1. Maturity Models and Secure Software Development Standards pose valuable research objects if interested in the industrial security findings management

2. Communication of data is a core aspect of security findings management

3. The list of requirements in Table 2.1 presents a current snapshot of practices demanded by documents commonly accepted in industry

2.3 Problem Description

In this section, we develop the problem concept of applying modern software development approaches to the practices necessary for security findings management in industry and describe the underlying problem this thesis addresses. Using the requirements identified in the last section, we retrieved a baseline of requirements that should be performed to manage security findings in industry. In this section, we collect the principles of modern software development methodologies representing the constraints applied to practices performed in such projects. We systematically map these principles to our requirements to identify the impact on the way these requirements are fulfilled. This mapping represents the core problem of our thesis, indicating the necessity to customize the security findings management to the applied software development methodology. Finally, we structure our problem into clusters of requirements and dispatch them according to their importance for industrial projects.

2.3.1 Principles in modern, industrial Software Development

The first step towards a systematic mapping of our identified requirements from the last section to modern industrial software engineering is the identification of principles applicable to the latter. This necessitates the definition of the phrase ”modern, industrial software development”. Looking at recent literature of the last ten years, a clear trend towards agile software development in industry can be recognized [111]. Publications claim that the need to deliver software faster [51] and adapt to customer needs more flexibly [144] initiated a transformation to incremental and agile software development methods. However, the initial scope of agile software development restricted its benefit for industrial software

development [51]. Multiple trends have emerged from this challenge, tackling the drawbacks of applying Agile for industrial development. One trend was the refinement of Agile principles to match its industrial setting [51]. The *Scaled Agile Framework* represents one instance of this trend [103], originating from the necessity to scale the Agile methodology to bigger teams [102] and map it to entire organizations. Another trend emerges from the close coupling between industry software and the hardware it operates. The separation of software development and its later operation induces several issues due to gaps in integration, collaboration, and communication [65]. Consequently, the entire lifecycle of a software product must be considered during all stages. This introduces the challenge of multi-disciplinary project teams. To foster productive collaboration between team members with varying backgrounds, organizations turned towards DevOps to bridge the gap between development and operations [65, 145] using cross-functional teams [202, 51]. These trends were confirmed with first-hand experiences by our industry partner, and the hypothesis of a parallel usage between DevOps and Agile practices was confirmed. Consequently, we consider industrial Agile and DevOps as core approaches for modern industrial software development.

The fundamental principles for Agile software development are published as a manifest and publicly available. This manifest's declarations are value propositions when developing software [17]. These propositions indicate the importance of one engineering aspect above another. To map the approaches for Agile development with our list of requirements, we require general principles that can also be mapped. Behind the agile manifesto, 12 principles exist that provide general guidance on aspects to keep in mind when performing agile software development [16]. Since these provide direct guidance for our mapping and still represent the basis for all scaled versions of agile development, we consider them our principles for Agile software development.

In contrast to the fundamental principles of Agile, no single source of truth has been established for DevOps. Neither academia nor industry has a universal definition for DevOps [51]. Humble and Molesky propose four core principles when bridging the gap between development and operations: culture, automation, measurement, and sharing [68]. Similarly, Gene Kim describes the fundamental principles of DevOps as the "three ways" [93], covering parts of the concepts provided by Humble. Moreover, an ISO standard exists describing the principles and application of DevOps. ISO/IEC/IEEE 32675 specifies the practices for operations teams, development teams, and other stakeholders on collaboration for successfully building and deploying systems [80]. Toward identifying DevOps principles, we analyze the ISO standard, the article by Humble, and the three ways by Kim.

Each way Kim describes lays out a set of project characteristics under one umbrella term. The first way describes the necessity of a fast flow from left to right, from the planning phase to the actual product operation. In the second way, he describes the need for a fast and constant flow of feedback from later lifecycle stages to earlier ones to avoid the repetition of errors and the early identification of issues before they can escalate. The third way deals with the culture of DevOps teams, which should promote continuous experimentation and organization-wide learning. For the ISO standard, we follow the descriptions associated with each principle. It defines four principles that must be

minded in DevOps: "Business/mission first", "Customer focus", "Left-shift and continuous everything", and "Systems thinking". Similarly, the principles defined by Humble are analyzed based on their respective description. Our understanding of principles that we can later map to our requirements is that we are interested in actionable properties, implying how some practice is implemented or the demand that it is performed at all. This excludes, e.g., the intended final state that should be achieved by following the principle ("Build verifiable systems").

During our analysis, we identified a clear distinction of principles between our different sources. While ISO/IEC/IEEE 32675 has multiple principles addressing the customer as a key decision maker, Gene Kim strongly emphasizes the role of feedback. Humble presents a balanced mixture of both while highlighting the importance of measuring project progress. Two key principles all sources agree on are the need for automation of repetitive process work and cross-domain collaboration. The list of principles resulting from our analysis can be found in Table B.1 in Chapter B of the Appendix. Comparing the list of principles for DevOps and Agile, we found that multiple principles overlap. To reduce the complexity in the upcoming sections for mapping and identifying tasks, we concluded that further aggregation is necessary. Consequently, we compared the 12 agile principles [16] to our constructed list of 17 DevOps principles to model topics. Summarizing the principles, we derived 12 topics listed in Table 2.2 with their respective sources.

ID	Topic	Principles
I	Measure the project performance according to the business metrics and make the progress visible	• DevOps: Make work visible • DevOps: Measure the performance of the project against business metrics • Agile: Working software is the primary measure of progress.
II	Reduce the size of work packages per delivery cycle to achieve short lead times	• Agile: Simplicity–the art of maximizing the amount of work not done–is essential. • DevOps: Reduce size of work per deployment cycle • Agile: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
III	Maximize the work not done by automating process work and repetitive practices	• DevOps: Reduce process work through automation • Agile: Simplicity–the art of maximizing the amount of work not done–is essential.
IV	Encourage fast, precise, and direct Feedback	• DevOps: Shorten and fasten Feedback Loops • DevOps: Provide feedback to where it is needed • DevOps: Amplify feedback
V	Provide knowledge and support to stakeholders, team members, and the organization requiring it for their work	• DevOps: Work is done with the cumulative and collective experience of everyone in the organization • Agile: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. • DevOps: Keep stakeholders informed about changes without overloading them
VI	Encourage self-organizing, multi-disciplinary teams	• Agile: The best architectures, requirements, and designs emerge from self-organizing teams. • DevOps: Include cross-domain teams into established meetings
VII	Continuously improve in all practices of the project by experimentation	• Agile: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. • DevOps: Support structured experimentation and risk-taking
VIII	Perform all practices continuously and in a repeatable manner	• Agile: Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. • DevOps: Create environments on demand • DevOps: Continuous build, integration, test, and deployment processes

IX	Encourage cross-domain collaboration by direct communication and sharing project aspects between teams	• DevOps: Share project aspects between teams (success, responsibility, tools, infrastructure) • Agile: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. • Agile: Business people and developers must work together daily throughout the project.
X	Conduct decisions with a customer-focused mindset	• DevOps: Balance concerns of risk against value for the customer • Agile: Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
XI	Establish team culture of technical excellence, system thinking, and sustainability	• Agile: Continuous attention to technical excellence and good design enhances agility. • DevOps: Establish a comprehensive, end-to-end view of the system for all stakeholders • Agile: Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
XII	Conduct Tests Early	• DevOps: Prevent defects to reach later stages

Table 2.2: Topic List of Modern Software Development Principles

This final list of aggregated topics represents an artificial abstraction layer across all principles from Agile and DevOps. Since this list was not empirically validated, its topics condense the principles that should be followed. Consequently, any ambition to check the adherence of a process to Agile and DevOps principles can be aided by this topic list but has to utilize the original principles as a benchmark. In the next paragraph, this topic list will be used to investigate the impact of modern software development approaches on the requirements for security findings management in industry.

2.3.2 Mapping of Development Principles and Security Requirements

Next, we investigate how the usage of modern software development approaches affects the security findings management in industry. Towards this goal, we map the list of principles identified in Subsection 2.3.1 to the requirements aggregated in Section 2.2. This mapping provides us with the knowledge **how** to do **which** practices to manage security findings in modern, industrial software development projects. This supports our later-stage research by providing insights into the design of security findings management practices necessary to integrate with modern industrial software development.

To achieve this mapping, we iterate over all topics from Table 2.2 and analyze how they apply to the requirements of Table 2.1. Whenever a principle affects a requirement by altering the underlying process or specifying additional constraints for the implementation of the requirement, we document this interconnection.

Topic I: The first topic addresses the visibility of the project's progress and performance according to business metrics. Consequently, this principle is closely coupled to any requirement measuring the security findings management process. As it further addresses the visibility of this progress, it also covers the communication of the progress to the project team and stakeholders. Consequently, we identify the mapping between Topic I and requirements 3, 5, 8, 9, 20, 23, 24, and 25.

Topic II: The second topic requests a reduction of work packages per deployment cycle to achieve shorter lead times. This implies that the overall planned work during each cycle is diminished. This covers, in particular, the amount of potential work done to improve the security status of the software product as well. Hence, this topic reinforces the necessity of requirement 15 while influencing all requirements that address the reaction to security findings, namely requirements 1, 4, and 19.
Topic III: With the third topic, the amount of repetitive and automatable work performed manually by team members should be reduced. This topic addresses every requirement in our list, as an investigation for automation capabilities throughout the entire security findings management process is crucial.
Topic IV: In topic cluster four, the principles address the need for direct, fast, and precise feedback in the project. Applied to the security findings management, this deals with feedback from the finding sources to the various stakeholders and processes. Each adjective represents one quality criterion when implementing feedback loops for security findings. Fast feedback implies that the data must be provided in a timely manner. Consequently, the processing time between the identification of a finding and its communication must be minimized to provide fast feedback. The demand for direct feedback requests that information is directly provided to the entity that requires it. For our use case, this indicates that multiple communication channels must be established depending on the stakeholders and processes requiring information. Constraints for the feedback presentation are defined more closely by the request for precise feedback. The actual information transported must be tailored to the entity receiving it. Consequently, a one-size-fits-all approach does not conform to modern development principles when communicating security findings data. In summary, this impacted all requirements dealing with the processing or the communication of information and only excludes requirements 4, 19, 23, and 24.
Topic V: The fifth topic addresses the necessity to share knowledge within an organization so that every individual's work is done with the comprised information of the organization. In particular, this means to provide individuals with all means to perform their designated tasks while avoiding an informational overload. Similar to topic four, these demands are closely coupled to any feedback from the security findings management. Hence, this topic impacts requirements 4, 8, 9, 12, 13, 17, 18, 21, 25, 26, 27, 30, 31, 32, 33, 34, and 35, while also applying to any activity that is performed based on data from the security findings management process.
Topic VI: In topic six, the team structure is addressed. Here, DevOps principles promote the importance of cross-domain team meetings, while Agile principles focus on self-organized teams. However, we could not identify any mapping for this topic as the team structure does not impose additional impact further than the requirements clusters four and five already do.
Topic VII: The principles of topic seven state that all practices of the project shall be continuously investigated for improvement by experimentation. In the context of security findings management, this implies that this topic again affects the entire security findings management process. By continuously improving the process, no final or optimal state of the security findings management process can be achieved. Instead, it is continuously re-

fined, improved, and adapted to the respective project team to improve the process. Hence, each project might develop its own process practices independent from other projects requiring a form of customization for all identified requirements. Therefore, we map this topic to all requirements.
Topic VIII: The eighth topic requests the execution of practices in a repeatable and continuous way. In software development projects, this reinforces the necessity for automation through continuous integration, deployment, and delivery. These practices can be realized using pipelines, performing the repetitive tasks of building, integrating, testing, and deploying software components in an automated fashion. Mapped to the management of security findings, this affects the preceding step: identifying security findings by security tests and checks. Consequently, this solely affects requirement 2, which deals with collecting these results.
Topic IX: The principles in topic nine focus on the collaboration aspects within projects. In contrast to topic six, it addresses the communication itself instead of how teams are constructed. For our use case, we conclude that all findings that require some level of collaboration must encourage the input of cross-disciplinary teams. This topic further reinforces the need for a common communication interface shared within the team and a tailored communication strategy for each recipient. Hence, it affects the same requirements as topic five and all processing requirements comprising some level of collaboration for their action additionally.
Topic X: The tenth topic requires that decisions are taken with a customer-focused mindset. Consequently, whenever decisions are taken during the security findings management process, the needs and goals of the customer should be considered. While this might affect all requirements, we identified a particular impact on the finding reaction, as the effort to address findings diminishes the available time for new improvements. Consequently, all requirements that affect the response of the project team to findings and any with impact on the customer are relevant to this topic, namely requirements 1, 4, 7, 10, 11, 12, 15, 16, 19, and 22.
Topic XI: Topic eleven comprises an organization's mindset and culture, which are necessary to develop software successfully. This includes attention to excellence in all domains and a perspective of the system as a comprehensive product instead of distinct silos. As these principles are rooted in each project team's culture, no direct mapping to specific principles can be established.
Topic XII: Finally, topic twelve requests earlier application testing to prevent defects from reaching later stages and potentially becoming more complex to solve. Similar to topic eight, this solely affects the preceding stages, as it changes the circumstances in which the security tests are performed. Consequently, this solely affects requirement 2.

This mapping shows the necessity to investigate the security findings management under the constraints of modern software development principles. Only the principles of Topic XI were not mapped to any of the requirements. All the others matched at least one, while most matched multiple requirements. Switching perspectives to the requirements, we saw that every requirement is impacted by at least one principle. Therefore, this mapping

Figure 2.3: Blackbox Data Flow Diagram

not only reinforces the motivation of this thesis but also provides us with the knowledge of how these requirements should be fulfilled to align with modern software development principles.

2.3.3 Problem Structuring

Based on the last paragraph, we confirmed that managing security findings in modern, industrial software development requires adaption of the traditional requirements existing in industry. To approach the design of a solution, we see the necessity for structuring the problem into manageable work packages. In Subsection 2.2.6, we identified a *Findings Lifecycle* illustrating one approach to structure the requirements. However, the content of the requirements indicates that several other types of categorization are possible. Even though requirements differentiate from one another, a process view would categorize them in the same process step. Treating the process of security findings management as a directed graph, we can perceive security findings as data flowing through this graph. The highest abstraction level is presented in Figure 2.3. *Security Activities*, like automated security testing, provide data in the form of security findings to the *Security Findings Process*. This process is a black box, where the data is processed somehow. Finally, the security information produced by the process is forwarded to the *Depending Entities* that require this information. In this abstraction the *Security Activities* represent the data source and the *Depending Entities* represent the data sink. More details can only be added to this diagram when analyzing the identified requirements of Section 2.2.

The first obvious categorization is the communication of security feedback. This represents the final step before data is provided to various *Depending Entities*. Each requirement of this category describes a different entity to which feedback should be transferred. This refines the original diagram by clarifying the list of relevant *Depending Entities* and necessitating a separate process stage for feedback preparation. The next category deals with the first actions after security findings have been reported. The requirements identify the necessity to collect, parse, aggregate, and enrich the data provided by the sources. These four requirements must be completed before the findings are further processed to ensure a certain level of data quality. Therefore, they change the original diagram by adding a process stage after the data sources generate the security findings. Based on this preprocessing, multiple requirements refer to the analysis of each finding. This includes a general analysis of each finding and specification on particular aspects where special attention shall be paid. These aspects include the validity of each finding, its security risk, its root cause, and the circumstances. Based on this data, each finding should be prioritized. This analysis of

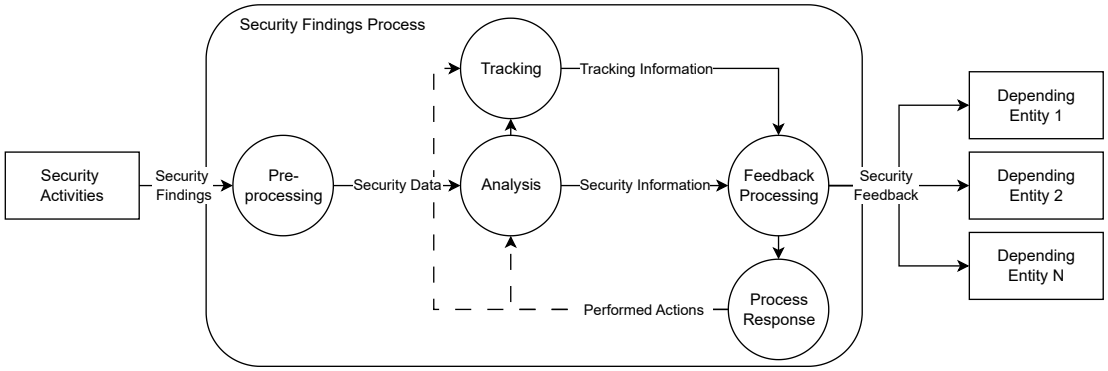

Figure 2.4: Security Findings Process - Data Flow Diagram

each finding provides insights into the finding processing and provides another processing stage between pre-processing and communication. The entire knowledge accumulated by this process must be documented and tracked according to our fourth cluster. The original requirements list contains four requirements requesting that the process and the findings it manages, is tracked, documented, and correlations or shortcomings of the procedure are identified. Since this depends on the data the analysis provides, it is situated between the analysis and the communication stage. Finally, the results of the finding analysis also fuel multiple activities that rely on the resulting information. Primarily, this data enables the team to respond to the identified findings, e.g., by solving the underlying issue. Moreover, it allows further investigations utilizing the data. This cluster comprises four requirements, demanding reactions based on the security feedback. This stage requires the same information as some *Depending Entities*. Consequently, it is settled after the processing of the feedback.

Clustername	**Requirements**
Feedback Processing	8, 9, 12, 13, 17, 18, 21, 25, 26, 27, 30, 31, 32, 33, 34, 35
Preprocessing	2, 14, 28, 29
Finding Analysis	1, 6, 7, 11, 15, 16, 22
Process Tracking	3, 5, 20, 23
Process Response	4, 10, 19, 24

Table 2.3: Clusters of Requirements

These five clusters of requirements are listed in Table 2.3, showing the cluster's name and all requirements. They provide insight into how the data flow diagram of Figure 2.3 is refined if the black box of the *Security Findings Process* is investigated more thoroughly. The discussed changes transform the original diagram into the version depicted in Figure 2.4. This includes our five clusters as the security findings process and shows the data flow between the clusters, the data sources, and the sinks. By considering the original 35 requirements as a process in a data flow graph, we could structure our problem into five

topic clusters of manageable size. This presents the final problem description and the first step towards the design of a solution.

2.4 Objectives of the Methodology

This section finalizes the problem conceptualization by combining the knowledge acquired in former sections and constructs our objectives for the remaining thesis. This construction is performed by designing the objectives from the acquired list of requirements and refining them according to our principles for modern software development. The final list of objectives represents the structure for the remaining thesis chapters.

2.4.1 Designing Objectives

Toward a solution, we need to prioritize each topic cluster and its content identified in Section 2.3 to build work packages. This order can be achieved by analyzing the existing problem instance in industry and mapping it to the different activities of our topic clusters. We performed this analysis by interviewing security practitioners at our industry partner for challenges when managing findings.

As subjects, we solely selected practitioners from a security background who experienced the management of security findings in either Agile or DevOps-oriented industrial software development projects in the last year. The unstructured interview was based on a single question, asking:

> Which challenges regarding the security findings management did you experience in the last year in software development projects that follow either Agile or DevOps principles?

After the interviews, all answers of the 17 subjects were analyzed, and similar responses were clustered. Based on this simple interaction, four pain points perceived by security practitioners were identified.

The first challenge states the significant influence of organizational preparedness on the success of the security findings management process. This interconnects the security side with the development principles in two ways. Suppose the maturity of the project is low. In that case, there might be either no interest in managing security findings or no need, as it does not fulfill fundamental principles like security automation. This reduces the complexity of security findings management since it is not in place. If the demand exists in a project with a low maturity, it can be challenging to reach the full potential of the process as certain preceding activities are not in place. One example was the improvement of the existing security tests by refining test cases based on solved findings as a standardized process. However, no team member was able to write test cases, as a central organization provided the testing tool with no customization opportunity. Our research also supports this statement since specific requirements can only be fulfilled if the respective process it depends on exists and is actively performed. From this challenge, we conclude that our

solution design must be able to support project teams, regardless of the project maturity and established practices. Hence, we must assume that processes may not exist or activities are not performed.

For the second challenge, our subjects emphasized the necessity of implementing practices and changes in practice, as sole knowledge is insufficient. Industrial development projects have a wide variety of information they can consider when building their security findings management. Our research of Section 2.2 showed that at least 21 documents indicate how security findings should be managed. Consequently, missing available knowledge can not be considered a noteworthy challenge. Our practitioners further suggest that the effort investment necessary for implementing practices could be the reason for this discrepancy between available knowledge and implemented knowledge. Hence, we conclude that the effort to integrate a methodology managing security findings into projects and the effort to follow the methodology must be minimized. In particular, the security findings management must be performed with the same efficiency as the software development itself.

The third challenge addresses the need to manage security findings data continuously. This challenge was exclusively identified by one subject, who highlighted the need to get information on the currently existing findings within a system at all times. This challenge originated from the circumstance that one project could not report all currently known findings, as the reports were solely processed on a weekly basis. To follow the agreement with our industrial partner, the circumstances are not disclosed in more detail. However, this challenge shows that our methodology should be able to communicate data about security findings at all times, regardless of the finding status (False Positive, Fixed).

The final and most referenced challenge is the type and frequency of the data provided by the security activities. Iterations in the development sprints and the automation of security tasks lead to continuous security testing, resulting in new reports that must be regularly investigated. Especially the system thinking of DevOps projects leads to a combined management of development and operation findings. New findings are added frequently and originate from all stages of the software development lifecycle. Hence, findings from security activities during development stages like code analysis are managed by the same process as security findings from operation security activities like monitoring. This leads to the challenge that a common process has to support the highly differentiating demands of different team members. Moreover, this leads to different perspectives being necessary to cover all aspects of the software, ranging from code and configuration to containerization and third-party dependencies. Consequently, the sources of the security findings vary, with each source differing in the reliability of the data it provides. Hence, not all information provided by our security tools is equally valuable. This also affects the coverage of the software. Sometimes, security activities overlap in their testing coverage, especially when trying to avoid whitespots. Duplicate findings are a logical consequence of this circumstance. This challenge underlines the complexity of the data and confirms the importance of preprocessing the data to achieve actionable security information. Moreover, the necessity of a platform automating the security findings management process and a target-oriented communication can be derived from this challenge.

Based on the interview results, we designed the objectives of the thesis. First, we can derive the necessity for a platform that orchestrates, automates, and documents the different process steps. This represents the first objective of the thesis. Second, the challenges around the source data indicate the importance of preprocessing the data, including the collection and parsing of reports, the aggregation/deduplication, and the enrichment of the resulting security findings. This preprocessing represents another objective. Next, the interview results showed the importance of a customized communication of data at all times. Even though this motivates the continuous processing of feedback, the first challenge showed the importance of maturity agnostic processing, dynamically adapting to the maturity of each project and organization. Hence, we define a further objective as dynamic security findings communication, which is not bound to specific process requirements but has to support all information communication. To continuously communicate the current state of the security findings management, the underlying data is needed before it can be shared. This requires that all findings are at least tracked within the process by documenting their current state and history. However, the status of a finding can only be acquired if it is analyzed. Consequently, we must consider the requirements on the finding analysis as well. The tracking and finding analysis represent our next objective in this thesis. With this objective, all requirement clusters are addressed except for the process response. The requirements in this cluster demand certain actions from the practitioners trying to fulfill them. Even though these actions belong to the security findings management process and their success is tracked accordingly, we see a substantial impact on the fulfillment by the challenge of project maturity. Similar to the topic of communication, we believe that the completion of some requirements depends on the project's maturity. Hence, we add them as requirements to the communication objective, providing the necessary information to perform the required actions and report the success.

This results in four objectives for this thesis to develop a methodology for the security findings management process in modern, industrial software development projects. In addition, the final objective is to evaluate the entire methodology in ongoing industrial software development projects. In the following paragraphs, we illustrate the precise problem each objective has to address.

2.4.2 Construction of Objectives

This thesis aims to develop a methodology for managing security findings in modern industrial software development projects. By investigating the existing problem, we are able to split this objective, achieving solution-oriented work packages. Based on the coarse-grained objectives defined in the last paragraph, we utilize the overall problem description of Section 2.3 to construct each objective. Each one represents the preliminary problem description for the respective domain-specific problem, constrained by the overall goal of developing a methodology for security findings management in modern industrial software development.

Objective 1 - Platform for Managing Security Findings:
The first objective targets researching and developing a platform that provides automation capabilities for the security findings management process. Consequently, its maturity should surpass a theoretical concept and instead represent an actual implementation. According to our mapping of Subsection 2.3.2, the platform should not only automate the process (Topic III) but further perform the processing of data fast (Topic IV) and in a repeatable, continuous fashion (Topic VIII). Moreover, the processing must be customizable so that it can be tailored to each project, and continuously improving processes can be reflected by the system (Topic VII). Finally, the design of this objective also showed that the platform should be easily integrateable into projects.

Objective 2 - Improving Data Quality of Security Findings:
The second objective deals with the preparation of any data processed by the security findings management by trying to mitigate the shortcomings of the source data. In particular, it deals with the collection of data, its parsing to a common data model, the identification of duplicates and aggregation of findings, and finally, the refinement of each finding's content. Similar to all other objectives covering requirements for the security findings management, the solution of this objective must be automatable (Topic III) and provide results in a timely manner (Topic IV). Furthermore, it represents the interface to any preceding stages of the software development lifecycle, like testing, and consequently deals with the continuous data collection (Topic VIII) and results from early stages of development (Topic XII). The information provided by the solution approach this objective shall achieve defines the source data for the next objective.

Objective 3 - Security Finding Analysis and Tracking:
The third objective is analyzing and tracking security findings and the surrounding process. It relies on the quality-improved security data from the second objective as input. The actual objective is twofold. According to our requirements, each finding should be analyzed for various aspects, including its root cause, priority, validity, or circumstances. The first aspect is how this can be realized as part of our process. Secondly, the objective demands a solution to how each finding and the entire security findings management process can be tracked. This objective is, similarly to the last one, constrained by the principles of automation (Topic III) and rapid data processing (Topic IV). Moreover, this objective comprises several requirements with human interactions. Consequently, the principles of a customer-focused and workpackage-reducing mindset during decision-making (Topic X, Topic II), the fulfillment of actions with proper knowledge (Topic V), and the cross-disciplinary collaboration on tasks (Topic IX) affect this objective. Finally, the project's progress must be made visible according to business objectives (Topic I), necessitating tracking the security findings management process.

Objective 4 - Security Findings Feedback Communication:
The final objective, contributing to the actual methodology, addresses the communication of knowledge acquired while processing security findings. This covers the bi-directional

communication to all stakeholders and the project team. It requires a tailored transport of information to various entities and the collection of user feedback. This further covers the actions to respond to each security finding, like its resolution. This objective is constrained by most principles, as it represents the interface to the actions and processes after the security findings management. Similar to the previously defined objectives, the communication should be conducted highly automated (Topic III) and deliver information in a timely manner (Topic IV). Since this objective deals with the entire communication and its subsequent actions, it is affected by the principles of a customer-oriented mindset (Topic X), the presentation of data in the context of business objectives (Topic I), the necessity of a common interface for cross-disciplinary collaboration (Topic IX) and the tailored transport of information to selected entities (Topic V). Finally, the solution approach for this objective must consider reducing work packages per deployment when designing subsequent actions.

Objective 5 - Instantiation and Evaluation:
The last objective does not contribute to the actual methodology but represents the empirical validation of the final solution instance. For this validation, all previous components instantiated as a single methodology and evaluated in ongoing industrial software development projects. Moreover, this implies that no principles affect this objective, as it is not part of the solution design.

The objectives, including their references to the modern development principles and the requirements, can be found in Table 2.4.

Objective	Title	Principles	Requirements
Objective 1	Platform for Managing Security Findings	• Topic III • Topic IV • Topic VII • Topic VIII	• N/A
Objective 2	Improving Data Quality of Security Findings	• Topic III • Topic IV • Topic VIII • Topic XII	• 2 • 14 • 28 - 29
Objective 3	Security Finding Analysis and Tracking	• Topic I • Topic II • Topic III • Topic IV • Topic V • Topic IX • Topic X	• 1 • 3 • 5 - 7 • 11 • 15 -16 • 20 • 22 - 23

Objective 4	Security Findings Feedback Communication	• Topic I • Topic III • Topic IV • Topic V • Topic IX • Topic X	• 4 • 8 - 10 • 12 - 13 • 17 - 19 • 21 • 24 - 27 • 30 - 35
Objective 5	Instantiation and Evaluation	• N/A	• N/A

Table 2.4: Thesis Objectives

2.4.3 Summary of the Objectives

In the introduction of this thesis, we assumed that a methodology that manages security findings in industrial software development projects must follow the interdisciplinary constraints of modern software engineering. In Section 2.3, we confirmed this assumption by mapping the naive requirements towards the security findings management process to the principles of modern software development and identifying the principles' influence on how these requirements should be fulfilled.

Identifying this gap, we conceptualized the overall problem according to our research methodology (Figure 1.2) and defined objectives for the thesis according to the research process depicted in Figure 2.5. We started by collecting demands for industry security findings management by accessing commonly accepted documents and aggregating them into a unique list of requirements (Section 2.2). We further derived guidance for modern software development from principles existing for DevOps and Agile projects (Subsection 2.3.1). These development principles were mapped to the requirements list, resulting in a mapping (Subsection 2.3.2). Moreover, the requirements were structured according to their overall topic (Subsection 2.3.3), and the resulting clusters transformed into the objectives by applying the knowledge about challenges existing in practice (Subsection 2.4.1). Finally, we combined this coarse design of our objectives with the mapping to construct the final objectives for the thesis (Subsection 2.4.2). This resulted in our five thesis objectives shown in Table 2.4, representing the solution design for the thesis and the instantiation and empirical validation according to Figure 1.2.

Each objective is investigated by following the Design Science Research methodology and presented in separate chapters.

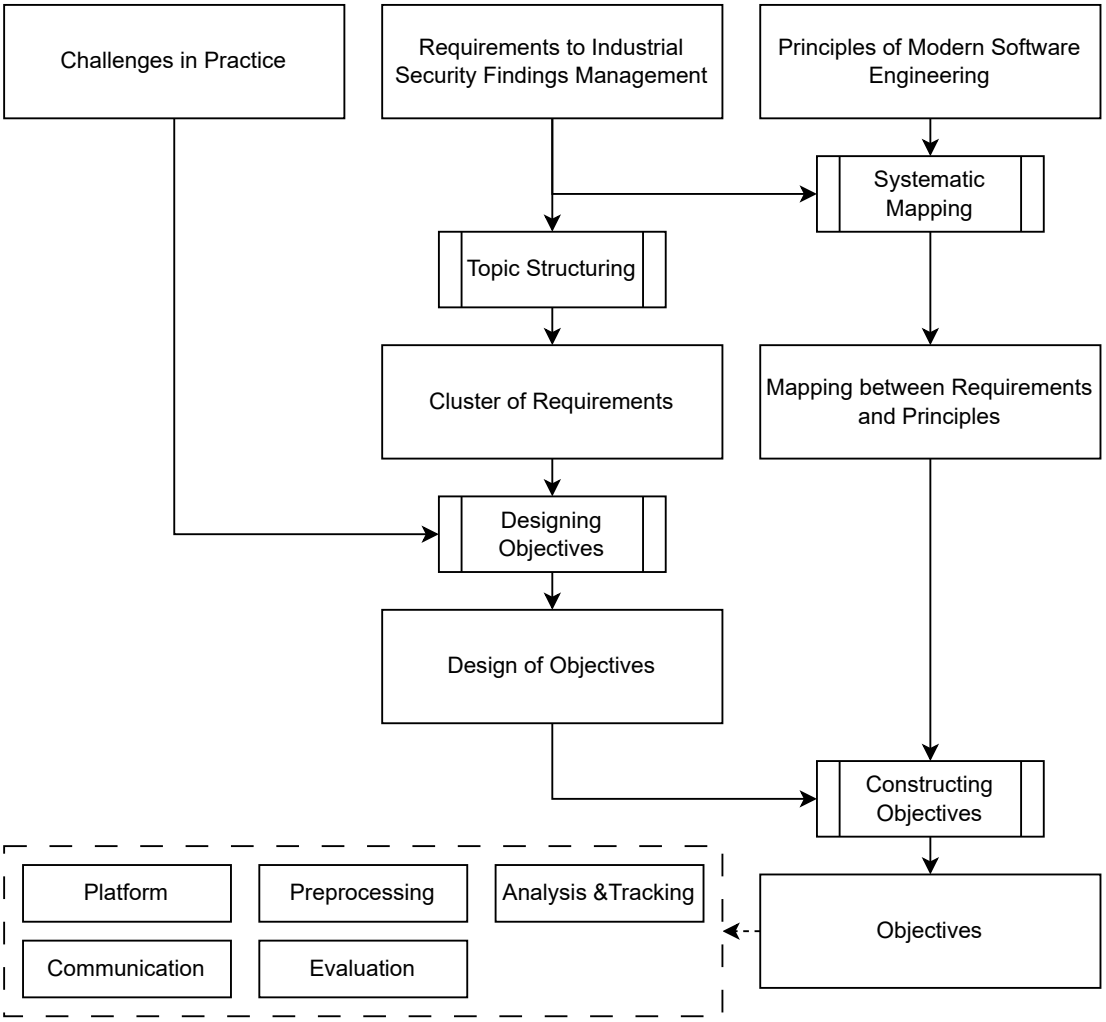

Figure 2.5: Problem Conceptualization Process

Chapter 3

Platform for Security Findings Management

To create a methodology for managing security findings in modern industrial software development projects, a platform for the continuous and automated execution of the necessary process stages is crucial. This chapter describes the problem behind this objective and presents our solution approach. The solution approach was further published in [195].

3.1 Problem Description

In our preliminary problem analysis, we identified the need for a platform that orchestrates, automates, and documents the management of security findings and related data processing. Section 2.4.1 defines four principles that must be followed by our solution approach, representing the starting point for the problem description. These principles are:

- Topic III: The platform must automate the processing steps
- Topic IV: The platform must provide fast feedback, necessitating short processing times
- Topic VII: The platform must be modular to support changing methods of data processing
- Topic VIII: All data processing must be implemented in a repeatable and continuous way

These principles, combined with the functional demands, result in a list of requirements presented in the following paragraphs.

Functionality The first problem that must be addressed is the platform's core functionality. As defined in the objectives of the thesis, the platform has to orchestrate any data processing identified later on automatically. Moreover, the data resulting from this processing must be automatically documented. Consequently, the platform requires data storage and orchestration capabilities.

Automation Project teams should later use the platform in practice, making a theoretical concept insufficient. Instead, automation and implementation of the entire platform are crucial demands. This affects the automation of any data processing introduced later on, the automation of the orchestration, and the automation of any interaction with the platform. Automating data processing implies that we require clearly defined interfaces to all platform components so that any processing approach can use the platform's functionality. Furthermore, any manual effort must be minimized, which affects, in particular, the effort for integrating the platform into each project. Hence, the data interfaces must provide reasonable automation capabilities to cope with adding, changing, or reading data.
Fast Feedback To conform with the principle of fast feedback, the time between data entering our platform and its computation being finished to the degree that it can be communicated must be minimized. We refer to the computation being finished instead of the management process since the management itself depends on manual work accomplished outside the platform. The decrease in computational time affects the efficiency of any data processing, efficient orchestration, and access to the resulting knowledge. As we cannot influence the necessary processing later integrated according to the project's need, the platform has to provide the means for an efficient implementation of any introduced processing. For an efficient orchestration, its overhead must be minimized. Finally, the data stored in the platform must be easily accessible and the respective knowledge efficient to acquire.
Modular Data Processing A considerable challenge for the platform is that the type of data processing required by each project is unknown. Furthermore, the continuous improvement of each project might necessitate new or changed data processing techniques. Consequently, the platform must support changes to the data processing and a modular approach for the orchestration to enable or disable processing steps.
Continuous Data Processing Similar to the security findings that are continuously identified, our platform has to perform the processing and orchestration continuously. This implies that any action orchestrated by the platform must be repeatable at any time. Minding the requirement for fast feedback, the resulting data must be maintained sound at all times, as a new security report might change the current security state of a product. Maintenance for soundness refers, in this context, to any operation that ensures that the information provided by the platform is based on evidence. Hence, the platform must support continuous data processing while maintaining the results sound.

3.2 Related Work

Data processing and transformation represent one of the fundamental research fields of computer science, rooted across multiple research areas.

The work related closest to our platform are existing tools for managing security findings. One of the most prominent tools in this domain is DefectDojo [37]. This platform provides the capabilities to automatically collect, document, and present security findings occurring in projects. Moreover, it processes the data by tracking its occurrence and iden-

tifying duplicates. Also, tools like Faraday [47] or OpenVAS [60] all provide the ability to process data from security testing. While these vulnerability management tools fulfill most of our requirements, they lack the crucial ability for modular data processing. Missing this processing agnostic approach, they cannot be considered as a solution approach for our problem instance.

Logical reasoning and its ability to derive new data from existing data according to certain relations is one step toward this modular approach. An important representative of logical programming is the declarative programming language Datalog. It comprises *Facts*, which are used as ground truth to derive new *Facts* according to predefined *Rules*. Using *Queries*, information about the *Facts* can be derived. In recent years, Datalog has been used in various domains, mainly for logical reasoning [19, 207, 21]. However, several limitations with consistent materialization [197, 120] and dealing with incomplete and potentially incorrect data [44, 92] diminish the effectiveness of Datalog for our setup. Even though promising approaches for overcoming these challenges exist [30, 201], no system could address all identified concerns. The most promising approach was DDlog [156, 190], a programming language for the incremental computation, enhancing Datalog with multiple non-traditional concepts. These additional concepts support further functionalities like changes to the existing fact base. However, the development of this approach was discontinued and still relies on the Datalog-typical Input-Output mapping, which may not be able to support every processing type.

Considering the platform as something more generic, a database could support the requirements for documentation and partial automation. Databases, however, lack the ability for data processing. Therefore, semantic knowledge bases yield promising properties for our platform. Solving challenges related to data processing with a semantic knowledge base is a common approach [123, 85, 42, 200, 91, 187]. Various fields utilize knowledge bases and inferences to address problems in their respective domain. Knowledge bases constitute a key component in platforms where new information must be inferred based on existing knowledge. For these inferences, rules or ontologies exist that allow reasoning about the information stored in the knowledge base. This inference is critical, separating a regular database from a knowledge base [95, 96]. Solutions utilizing knowledge bases cover a broad range of topics such as managing sensor data [123], search engine optimization [42], log data management [85, 86], and the elicitation of high-quality requirements [91]. In the area of software security Wang and Guo created an ontology to represent vulnerabilities with key concepts [200], like countermeasures. The ontology in their work, which comprises data from common databases and knowledge like NVD [125], CVSS [135], or CWE [179], intends to support inferences and decision-making in the area of vulnerability management.

These well-established concepts of logical reasoning and the versatility of knowledge bases present the initial solution approach.

3.3 Semantic Knowledge Base

Derived from the requirements for a platform that automatically orchestrates and documents the data processing steps of security findings management in modern software development projects, we decided to use a semantic knowledge base. This section describes the concept for the knowledge base and each component in detail.

3.3.1 Knowledge Base Concept

To use a knowledge base as our platform, we initially define its concept. As discussed in the related work, semantic knowledge bases have a broad area of application for any task related to data processing. However, each knowledge base must be tailored to the use case. In our case, we mainly make use of two properties. The inherent documentation of data and the continuous inference of new data from existing data, according to predefined rules. As described by Krótkiewicz et al. [95, 96], our knowledge base is also roughly characterized as database with automated inference capabilities.

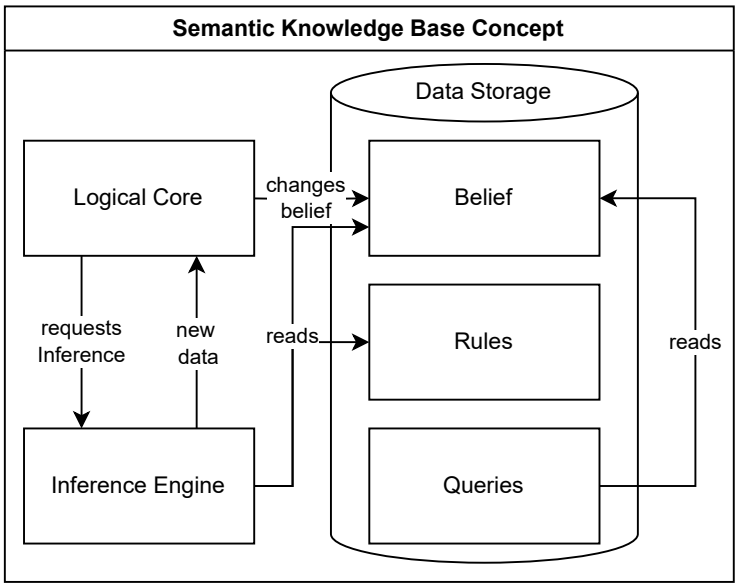

Figure 3.1: Conceptualized Components of the Knowledge base

Merging the deductive reasoning approach from Datalog using *Facts*, *Rules* and *Queries* with the fundamental knowledge base concepts of Krótkiewicz et al. [95, 96], we define our knowledge base to consist of six components.

- Belief
- Rules
- Queries

- Inference Engine
- Logical Core
- Data Storage

Belief represents any security data in the knowledge base that is either given by external data input or logical inference. These logical inferences are formulated by *Rules*, describing how to derive new *Belief* from the *Belief* currently stored in the knowledge base. To apply the *Rules*, the *Inference Engine* executes the respective *Rule* and provides the interface to access the stored *Belief* for read operations. To ensure that new insights are transformed to new *Belief* and changes in the existing dataset are properly corrected in the data that is derived from it, the *Logical Core* orchestrates the *Inference Engine* and maintains the soundness of data in the knowledge base. To provide external access to the *Belief* stored in the knowledge base, *Queries* are employed. These *Queries* represent formalized views on the knowledge base data. To ensure continuous documentation, all *Belief*, *Rules*, and *Queries* are stored in the *Data Storage*. These three components further enable the necessary customization to each project. The six components of our platform and their relations are depicted in Figure 3.1. Five of these core concepts are described in the following sections in detail. The *Data Storage* is excluded as it solely affects the implementation.

3.3.2 Belief

The concept of belief comprises the actual informational content in our knowledge base, often called *fact* in other knowledge base definitions [30, 201]. In contrast to traditional facts, our definition of belief is adapted to match the requirements for our platform.

The naming convention of belief represents the first change to facts. Contrary to traditional knowledge bases, where this information is considered an immutable fact or knowledge, such a view is prone to inconsistency when working with security findings. Security findings comprise information gathered by a specific activity analyzing a product from a certain perspective. Inherent to this perspective is the possibility of incorrect data, which is invalidated or changed over time (e.g., False Positives). Therefore, we broaden the definition and consider them as mutable belief, which is prone to change over time. The new consideration of belief affects data provided by external activities and any information derived from it by logical inference. This distinction between externally provided data and inferred data represents another change in our definition. We consider belief that originates from knowledge base external sources as *explicit belief*, while any belief from logical reasoning is named *derived belief*. Minding that the coverage of multiple security activities in the same project might overlap, the resulting security findings could contain conflicting information. This existence of contrary belief is allowed in our definition of the knowledge base. Finally, each instance of belief must be unique. Therefore, no two instances of belief with the exact same information may be included in the knowledge base.

The abstract concept of belief is realized by belief classes representing real-world counterparts. An exemplary belief class would be ”Security Report” or ”Security Finding”. Each class is defined by the attributes inherent to the real-world counterpart. In addition to the real-world attributes, each class contains the class name, a unique identifier of the instance, a content hash over all real-world attributes, and the type of belief. These classes are instantiated, and their attributes are declared to store data in operation. Below one exemplary instance of the ”Security Report” class is depicted:

```
"Class Name": "Security Report",
"Identifier": "12345",
"Content Hash": "a1b2c3",
"Belief Type": "Explicit",

"Source": "Tool A",
"Source Version": "1.0",
"Timestamp": "YYYY-MM-DD HH:MM+Tz",
"Content": "Security Report in JSON Format..."
```

This example shows a security report being stored as explicit belief. As real-world attributes, the source of the report, its version, the content of the report, and its creation timestamp. Such a class could be implemented as a table in a database or document class in search engines, where each instance would be a row or document, respectively. To ensure the modular properties of the knowledge base, new classes of belief can be constructed at any time. A visual representation of the relation between belief classes and their instances is depicted in Figure 3.2.

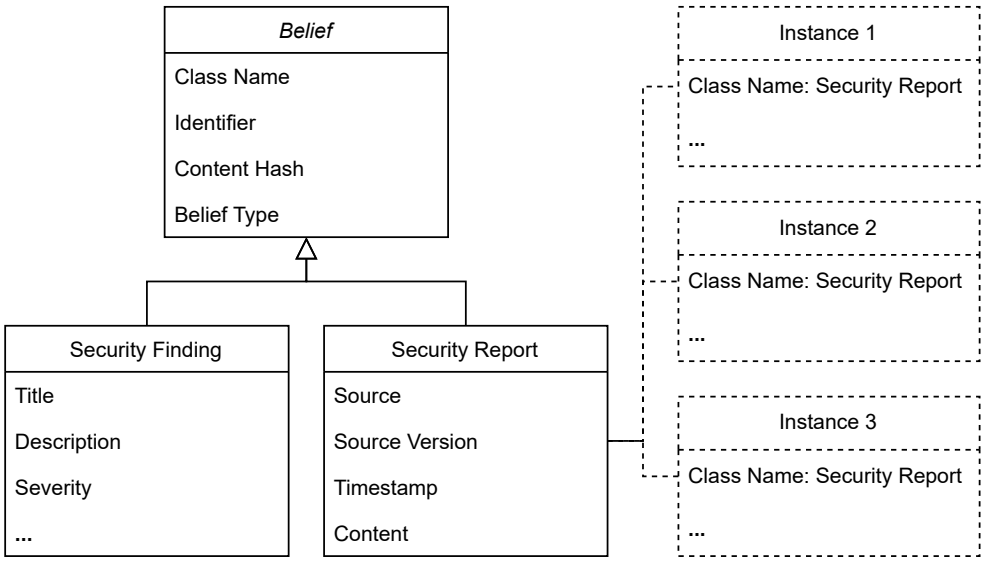

Figure 3.2: Relation between Belief Classes and their Instances

3.3.3 Rules

The concept of rules represents the relation between belief classes existing in reality. In contrast to an ontology existing in common logical reasoning, the set of rules in our knowledge base is not predefined and can be extended on demand. Toward this goal, we defined a separate rule specification.

To ensure the modularity of our inference approach, we refrain in our definition from Input-Output mappings between fields as common in logical reasoning languages like Datalog. Instead, our rules support traditional programming constructs. Moreover, the complexity of real-world relations necessitates inferences that utilize multiple classes of belief. Therefore, each rule may utilize any number of belief instances in the knowledge base.

To ensure that rules can be used to infer new belief from the existing belief in the knowledge base, we define them as a 4-tuple. The first element of any rule is the *Trigger*, defining which classes of belief are used as input for the rule computation. The *Trigger* is further essential to identify when a rule should be executed to derive new data. The actual code that defines how new data is computed is contained by the *Code*. Multiple assumptions apply to this code. First, the code must always infer the same information under the same circumstances. Consequently, a random generation of information is not acceptable for any rule. Next, each code has access to the belief instances used to trigger it and read access to all other belief instances in the knowledge base. This is crucial for complex inferences, as each rule might derive information from multiple belief classes. Finally, each code must be able to trace the source elements used to infer a belief instance. Otherwise, maintaining the knowledge base sound throughout changing data is impossible. The type of belief resulting from the rule is represented by the *Output* element. The *Trigger* and the *Output* are crucial for the orchestration of rules by the inference engine, while the *Code* contains the actual inference. Each rule has a unique *Name* assigned to it for identification. Continuing the example above, we consider a rule that parses security reports into multiple security findings. This results in the following rule 4-tuple:

```
{"Trigger": "Security Report", "Code": <program code>,
"Output": "Security Finding", "Name": "Parser"}
```

The pseudo-code below is an example of such a parsing rule:

```
INPUT: element

report = element
result = list()
if report.source == "Tool A"
    for entry in report.content
        result = result + map(entry)
endif
return result
```

This example's *Code* takes a belief element as input and ensures it is a security report. Next, it checks the source of this report and parses all findings according to the respective

mapping. Each mapped finding is stored in a list of results that is returned subsequently. New rules can be added at any time during operation and will be considered by the inference engine.

3.3.4 Inference Engine

The purpose of the inference engine is to orchestrate the inference of new data using the belief and rules existent in the knowledge base. Traditional semantic knowledge bases often refer to the mechanism that derives new data from existing one as data processing methods. We, however, split this into the functional calculation of data, represented by the inference engine, and the maintenance of data soundness, represented by the logical core, as both serve distinct purposes in our knowledge base. The inference engine receives

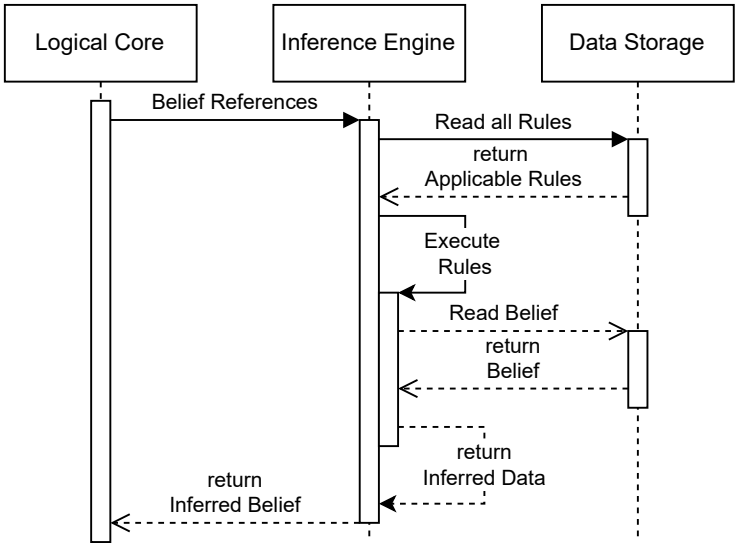

Figure 3.3: Orchestration of the Inference Engine

a reference to belief instances stored in the knowledge base and tries to derive knowledge from it by orchestrating the rules in the knowledge base. Based on the class name of the belief references, it checks all knowledge base rules for applicable triggers. The respective rules are executed, and the resulting data elements are transformed into the belief classes they should represent as defined by the rule tuple. The so-derived belief is not stored in the knowledge base yet but returned to the logical core, including references to all other belief instances used as sources for the inference. The logical core decides on the subsequent actions with the newly created belief instances. This process is depicted in Figure 3.3.

3.3.5 Logical Core

Our concept of a logical core ensures the decisive aspect for the correctness of information in our knowledge base: its *soundness*. We define *sound* as the state of the knowledge

base, where it solely contains explicit belief and all information that can be derived from it using the currently available rules. A sound knowledge base state excludes, e.g., belief that could have been inferred during an earlier state of the knowledge base but cannot be inferred anymore as source data was deleted or is outdated. Any decision taken using the knowledge of an unsound knowledge base can be incorrect, as the knowledge itself could be flawed or outdated. If the knowledge base is in an unsound state, maintenance operations must be performed to regain soundness.

Our logical core follows an inductive approach to ensure that all potential conclusions are also drawn. This implies that any external information provided to the knowledge base, like a new security testing report, is processed, and conclusions are drawn immediately. Otherwise, the information requested by users of the knowledge base would be calculated as when requested during runtime(deductive), slowing down the feedback provisioning. Therefore, our knowledge base is built according to the *Bottom-Up* principle. This inductive reasoning not only affects the data being added but also change or delete operations. Since different sources might claim conflicting information, belief could be changed or deleted over time. To ensure a continuous inference throughout new data being added while maintaining soundness, our logical core has to handle delete and change operations on belief instances. This necessitates a revisable inferences structure. These changes to existing elements potentially invalidate all instances of belief derived from it, as they depend on outdated information. To identify and correct these instances of belief, the relation between elements must be tracked. However, these relations may not be stored within the belief elements, as they do not represent data related to factual information but meta-data. Therefore, the logical core must separately track the belief relation automatically. In contrast to other maintenance approaches, e.g., in Datalog, our logical core does not support incremental maintenance. With incremental maintenance, the changes to existing belief instances would be minimized so that only a single field of a belief instance could be changed without changing other fields. However, the necessity for uniquely identifying belief instances by hashes and the modularity of rules possibly requiring arbitrary mappings between fields prevent this feature.

The central functionality of the logical core is the maintenance of knowledge base soundness. Solely external changes to the knowledge base may introduce unsoundness comprising the addition of new data, the change of existing belief, or the deletion of belief. Furthermore, changes to the rule set also infringe the knowledge base soundness. These operations either invalidate belief stored in the knowledge base or allow additional conclusions to be drawn. To identify unsoundness, the logical core is notified about any external changes to the knowledge base. To reinstantiate soundness in the knowledge base, the logical core can add new belief, change existing belief, or remove outdated belief. Details on the maintenance operations and the overall inference structure is described in Section 3.4.

3.3.6 Queries

Our concept of queries serves the same purpose as the classical Datalog query. They represent an abstraction layer for the elements of belief in the knowledge base intended to be used by external entities accessing the data. These queries present a particular view of the data customized to the knowledge intended to be acquired. Therefore, they can also combine the data of multiple belief instances across any belief class, providing the means for top-down knowledge calculation. Examples range from simple listing of all security findings to more complex statistics on the occurrence of security reports.

3.4 Inference Logic

The inference logic employed by the logical core represents one of the most significant contributions to our knowledge base. In this section, we present the Inference Structure given by the relation between instances of belief in Subsection 3.4.1 and the strategies for maintaining the soundness of the knowledge base in Subsection3.4.2.

3.4.1 Inference Structure

The inference structure of our knowledge base defines the possible relationship between instances of belief. This relationship is inherently defined by rules deriving new data from given data and consequently building a dependency graph. Crucial for understanding the inference structure is the distinction between belief classes and instances of these classes as depicted in Figure 3.2. One instance of derived belief might depend on an arbitrary number of belief instances from different belief classes. For the tracking of soundness and the overall structure, solely the belief instances are relevant. Consequently, the relations presented in the following refer to the number of belief instances, regardless of the classes they belong to.

Instances of belief can have a *0..* — is derived from — 0..** relationship between each other. This means each belief instance is derived from an arbitrary number of belief instances and can again be used to derive an arbitrary number of belief instances. Since each relation has a different impact on the logical consequences when maintaining soundness, they are discussed in the following paragraphs.

Belief Relation 1: 0—1..*, 1..*—0

The two most trivial relationships include the belief relations where instances of belief are not derived from any other instance and where instances are not used to derive further data. The first belief relation exclusively affects explicit belief, resulting in instances of belief not relying on the validity of other belief instances. The explicit belief, however, may still be used to derive further belief instances. The other belief relation affects belief instances that are not used to derive any other belief instances. This happens either if no rule exists that utilizes this class of belief as input or if the belief instance's content

is irrelevant to any rule (e.g., an empty security report that cannot be parsed into single findings). This belief relation is visualized in the top left corner of Figure 3.4.

Belief Relation 2: 1—1..*
The second belief relation addresses inferences, where one instance of belief is used to infer an arbitrary number of belief instances. This belief relation represents a simple dependency with a linear approach for resolving unsoundness. The enrichment of security findings gives an example of a one-to-one relationship. Here, one belief instance representing a security finding is processed and results in another, representing the enriched version of the finding with a dependency solely on the original security finding. On the other hand, parsing a security report is an example of a one-to-arbitrary relationship, as it is unknown how many findings can be derived from a security report. In this example, the validity of each security finding depends on the validity of the security report from which it was derived. This belief relation is visualized in the top right corner of Figure 3.4.

Belief Relation 3: 1..*—1..*
The last and most challenging belief relation is the usage of multiple belief instances to derive an arbitrary number of belief instances. Examples include the deduplication of security findings, where multiple belief instances are analyzed for duplicates, and the resulting cluster is stored as a single belief instance. Consequently, the validity of this cluster element depends on multiple instances of belief. Maintaining soundness becomes even more challenging when an arbitrary number of belief instances is used to derive an arbitrary number of belief instances. This belief relation is visualized at the bottom of Figure 3.4.

3.4.2 Soundness Maintenance

Maintaining the soundness of the knowledge base is crucial for the correctness of the information provided by it. The strategies for maintaining soundness are explained based on the inference structure described in the last subsection.

To develop a strategy for maintaining soundness, all events infringing soundness must be identified. An empty knowledge base is sound by definition, as no conclusions can be drawn and the contained data is not incorrect. As soon as operations that originate from outside the knowledge base add the first belief instances, unsoundness is introduced, as additional data can be inferred. Consequently, our maintenance strategy must consider the *ADD* operation. With new reports added over time, formerly correct information might be invalidated. For example, new security findings could falsify the current deduplication clusters as they are not exhaustive. Therefore, existing belief must either be changed to match the new information or even deleted if it shows to be incorrect. Therefore, the additional *DELETE* and *CHANGE* operations must be minded. Finally, the information processed by the knowledge base is queried by external actors. As these queries do not

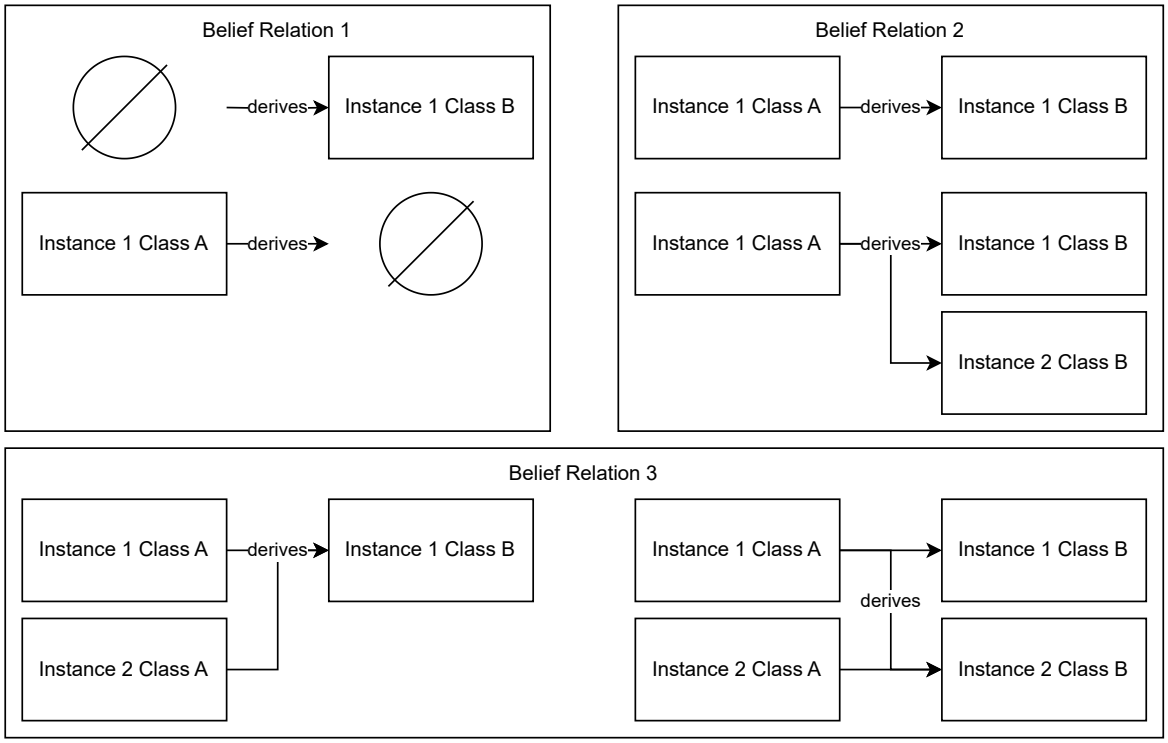

Figure 3.4: Relationship between Instances of Belief

affect the belief stored in the knowledge base, this operation may not be considered in the strategy.

In contrast to well-established maintenance protocols, like those employed in Datalog, our approach works on entire instances of information instead of single pieces. Since our belief classes contain attributes with complex relations to attributes of other classes, attribute-based maintenance (i.e., if one attribute changes, other attributes are changed instead of the entire belief instance) is impossible. The reason for these complex relations is the ability to use traditional programming constructs in inferences. With those, a precise relation between attributes is impossible. However, the fundamental idea of correcting the minimal amount of data that might be affected by a change in the data set is adopted from the traditional approaches.

To rebuild the knowledge base's soundness we start the maintenance at the instance affected by the operation and resolve all conflicts of instances depending on it. Each operation requires a different strategy for restoring soundness. These are described for each operation below.

Maintenance of ADD Operation

After new belief instances are added to the knowledge base, regardless of whether this is the result of an external operation or an internal inference of new data, all possible conclusions must be drawn from this new information. Therefore, the new belief instance is

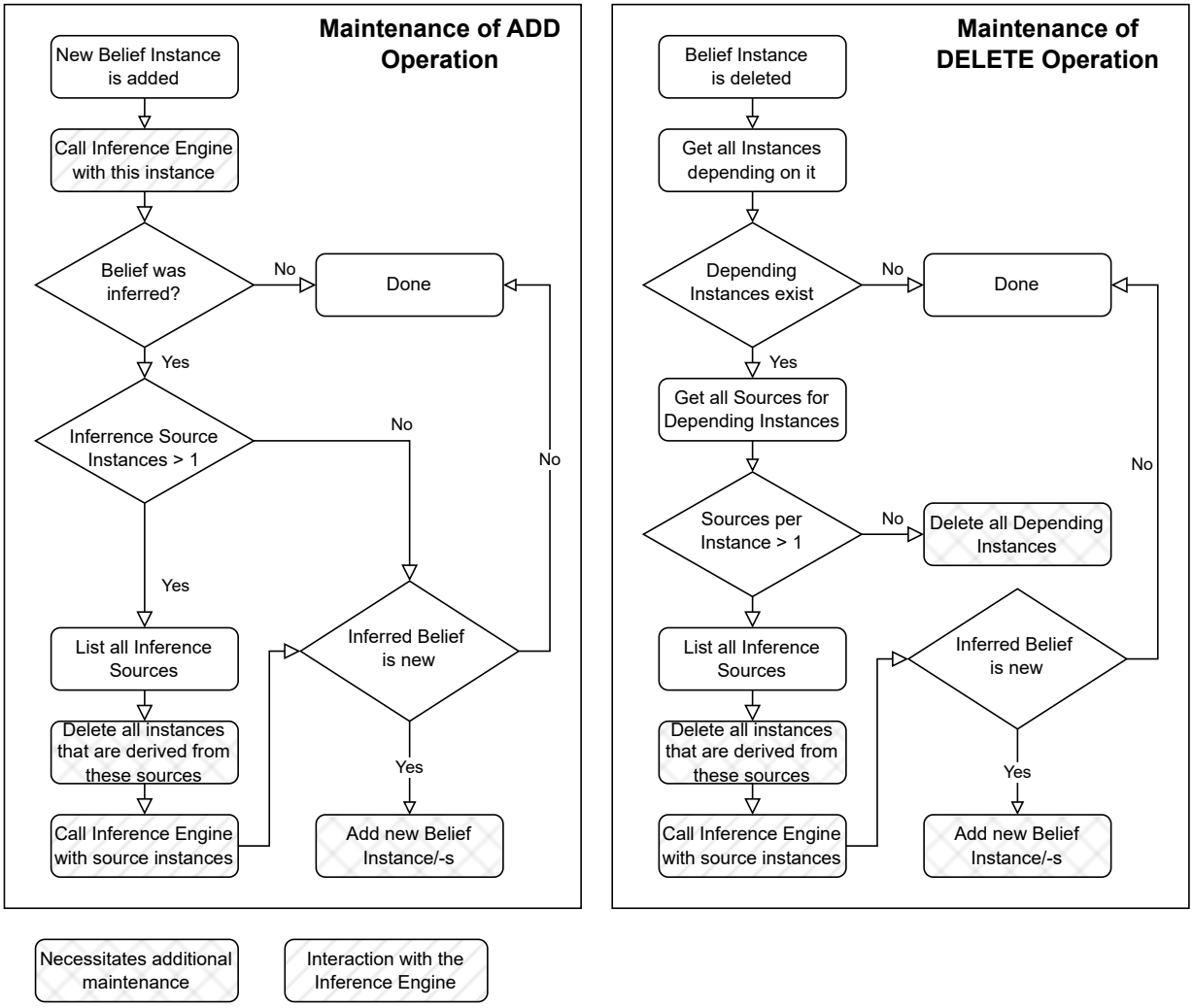

Figure 3.5: Flow Chart for Maintaining Soundness after an Add and Delete Operation

provided to the inference engine, and the derived belief and references to all belief instances used in the inference are also returned (See Belief Relation 3 of Figure 3.4: arbitrary number of source instances). Suppose the inference engine could not derive information from the given element. The knowledge base is sound in this case, as all possible conclusions have been drawn (see Belief Relation 1 of Figure 3.4). Otherwise, we focus on the instances used as source elements for the inference. If only one belief instance was used as source (see Belief Relation 2 of Figure 3.4), we have to check whether the derived belief instance is not yet in the knowledge base based on its content hash and store it if this is the case. This triggers another iteration of the knowledge base maintenance, as another new belief instance was added. Already stored belief instances are dropped, resulting in a sound knowledge base. However, if more than one belief instance was used as source (see Belief Relation 3 of Figure 3.4), the maintenance has to mind cross dependencies. Consider the deduplication of security findings as an example of this circumstance. For each security finding, duplicate findings are identified, and the resulting cluster is stored as belief in the knowledge base. If a new security finding is processed, the cluster this new finding belongs to might already exist in the knowledge base. Hence, the existing cluster must be overwritten to maintain soundness. Otherwise, two clusters with conflicting information would exist in the knowledge base, resulting in an unsound state. For our maintenance strategy, this implies that all source belief instances are collected and all instances derived from them are deleted. Afterward, these source instances are provided to the inference engine to re-generate the currently valid belief instances. These are again checked for their existence in the knowledge base and added if they do not exist. This process is depicted in the left Flow Chart of Figure 3.5.

Maintenance of DELETE Operation

If existing belief in the knowledge base is deleted, this invalidates all information depending on it. Consequently, the first step after a belief instance is deleted is identifying all belief instances that were inferred by it. If no instances utilized the deleted knowledge as source (see Belief Relation 1 of Figure 3.4), the maintenance is done, as the deletion invalidated no information. If, instead, belief instances depended on it, the number of source elements of each depending instance is determined. If the deleted element was used as the only source for the inference (see Belief Relation 2 of Figure 3.4), all depending instances can be deleted. If more than one belief instance was combined with the deleted element (see Belief Relation 3 of Figure 3.4), the maintenance must mind the cross dependencies. Utilizing the example of deduplication once again, this implies that the deletion of one security finding would not result in the deletion of the entire cluster but must result in the recomputation of the cluster, excluding the deleted element. Otherwise, the other security findings that have formerly been clustered are incorrectly separated again. Therefore, we follow the same approach employed during the *ADD* maintenance and delete all belief instances derived from these sources. Afterwards, these source instances are provided to the inference engine to re-generate the currently valid belief instances, avoiding the situation described in the example. Any belief instance not existing in the knowledge base is added,

and the maintenance continues with the newly added belief instances. This process is depicted in the right Flow Chart of Figure 3.5.

Maintenance of CHANGE Operation

Whenever existing belief instances in the knowledge base are changed, this potentially invalidates all information derived from the now outdated information. Consequently, its maintenance effort is equal to the maintenance of a delete operation, with the difference being that the re-generation of the currently valid belief happens regardless of the number of source elements. Therefore, the right Flow Chart of Figure 3.5 correctly represents the maintenance process, excluding the second conditional box, as the recalculation must be conducted every time.

Maintenance of Rule Changes

The only other external change to the knowledge base that might infringe the soundness are changes to the rules. This includes the addition, deletion, and changes to the existing rules. Since the impact of rules cannot be quantified like belief changes can, the entire knowledge base has to be rebuilt. Towards this goal, all derived belief instances are deleted, and a maintenance operation for adding all explicit belief instances is started. This rebuilds all derived belief instances and ensures a sound knowledge base after rule changes.

3.5 Implementation

Crucial for the usage of a semantic knowledge base as a platform for the continuous and automated execution of security findings management process steps is its practical implementation. Therefore, the knowledge base concept presented in the previous sections was implemented as a framework. The framework itself does not contain rules, belief or queries by default, as these depend on the process steps. In this section, we present the implementation of the framework and introduce its functionality based on two examples.

3.5.1 Implementation

The implementation of the semantic knowledge base is a containerized Python application, realizing the concept of Figure 3.1. Figure 3.6 depicts all components and their respective interfaces. The biggest challenge for the implementation were the constraints towards belief instances and rules/queries. While belief instances are similar to database entries, the rules and queries contain executable code. Therefore, the *Data Storage* consists of an Elasticsearch search engine supporting the *Belief Storage* and a filesystem dealing with the *Rule/Query Storage*. The *Belief* is implemented as Elasticsearch documents, where a separate mapping represents each belief class. Since Elasticsearch allows the dynamic addition of data without formal mapping, any new belief class can be added dynamically. *Rules* and *Queries* are split into the executable code, which is stored as a file on the filesystem, and the remaining meta-data, which is maintained as a dictionary. Consequently, the rules are implemented as 4-tuple

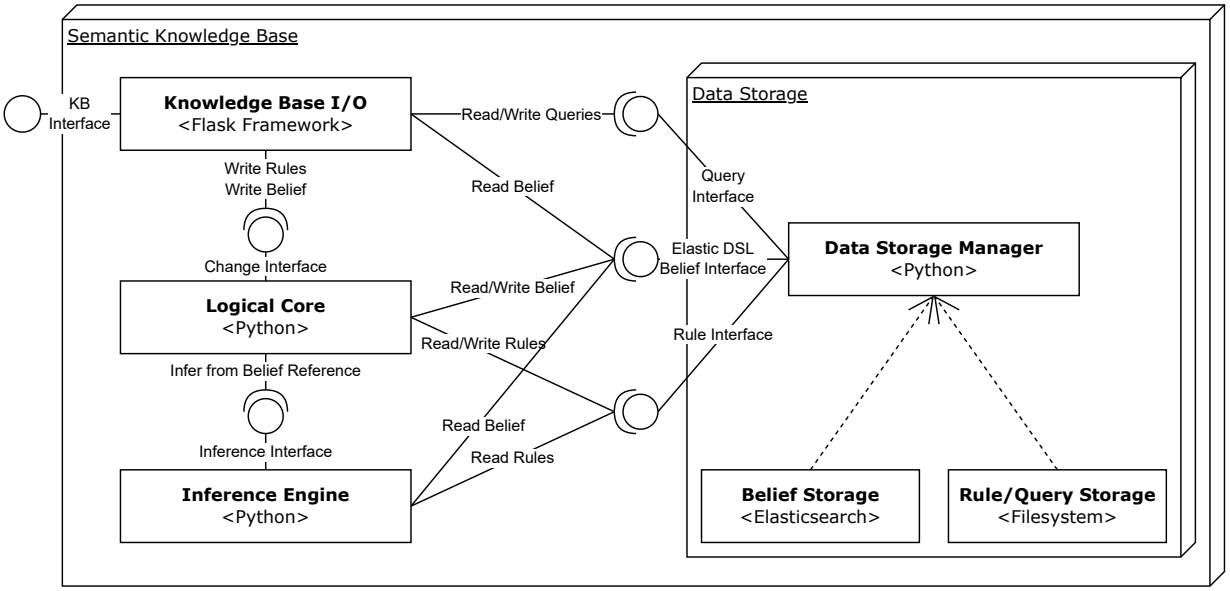

Figure 3.6: Component Diagram of the Semantic Knowledge Base

```
{"Trigger": "<Triggering Class of Belief>", "Code": "<Reference to the File>",
"Output": "<Class of Belief Ouitput>", "Name": "<Name of Rule>"}
```

while the queries are implemented as 2-tuple

```
{"Name": "<Name of Query>", "Code": "<Reference to the File>"}
```

To access the data storage including all instances of belief, rules, and queries, an abstraction layer is implemented. This *Data Storage Manager* orchestrates all storage requests and manages the separation between belief and rule/query storage. Furthermore, it offers three interfaces for the belief, the rules, and the queries. The belief interface employs the Elasticsearch Query DSL as the underlying mechanism, while the rule and query interfaces are developed from scratch. They allow to search for queries based on the name, returning the filesystem path, and the rules can be searched based on their *Trigger* or *Output* field, returning the entire 4-tuple.

The *Inference Engine* is implemented in Python and offers an interface towards the *Logical Core*, where a list of derived belief instances is returned, if provided by a list of references to belief instances in the knowledge base. The crucial part of its functionality is the search for rules that contain the provided list of belief instances as trigger. To search for applicable rules, it utilizes the data storage abstraction layer. Based on the returned filesystem path, each found rule is executed. If the rule contains belief queries and, therefore, needs access to the knowledge base, the belief interface is used. Any data generated by the rule execution is parsed to the respective format and stored temporarily. After all applicable rules are executed, the list of derived belief instances is returned.

The interface offered by the *Inference Engine* is exclusively used by the *Logical Core*. For the *Logical Core* to maintain the soundness of the knowledge base according to Section 3.4,

it has full read and write access to the belief and rules in the data storage. It further contains a mapping to track the relation between belief elements, where these relations are documented and maintained. The mapping contains two entries for each belief instance, stating all elements that have been used to derive it and all elements that have been derived from it. This eases the effort for the maintenance of soundness. Moreover, it offers an interface that manages all write operations on rule and belief elements in the data storage.

Finally, the knowledge base contains the *Knowledge Base I/O* posing as a single point-of-contact for all operations originating from external actors. Implementation-wise, this component is based on the Flask Framework and exposes a RESTful API dealing with the requests. As all operations with queries can not result in unsoundness of the knowledge base, they are directly managed by this interface. This includes the addition or change of queries and the execution of queries to access the belief instances. However, all write operations on rules and belief may lead to unsoundness. Therefore, these operations are forwarded to the *Logical Core*.

3.5.2 Example 1: Add Belief

With the first example, we present the functionality of the knowledge base if a new belief instance is added to it. We assume that the knowledge base initially contains no belief instances for this example. We further assume that it contains exactly one rule. This rule is depicted below

```
{"Trigger": "Security Report", "Code": "<Reference to the File>",
"Output": "Security Finding", "Name": "Parser"}
```

The code of this rule parses a security report into the security findings it consists of. The belief classes ”Security Report” and ”Security Findings” are also defined.

Our example, depicted in Figure 3.7, starts with an external actor adding a new security report (1). This element is forwarded by the *Knowledge Base I/O* to the *Logical Core* for processing (2). The *Logical Core* employs the data storage interface to store this belief element (3) and starts with the maintenance of soundness. Following the process for maintaining soundness after an add operation (see left diagram of Figure 3.5), it first calls the *Inference Engine* with a reference to this newly added belief instance (4). Since the referenced belief instance belongs to the class ”Security Report”, the *Inference Engine* requests all rules with ”Security Report” as trigger (5). The data storage interface returns the parser rule shown above since it is the only rule with an applicable trigger (6). By executing this rule, the *Inference Engine* parses the newly added security report into single findings (7). These are parsed into instances of the belief class ”Security Findings” and returned to the *Logical Core* (8). The logical core follows the maintenance operation and notices that although belief was inferred, the source elements are solely the one added security report (9). Therefore, the *Logical Core* checks that the inferred belief does not yet exist in the knowledge base and stores the new belief using the data storage interface (10). This results in another maintenance operation as new belief instances were added.

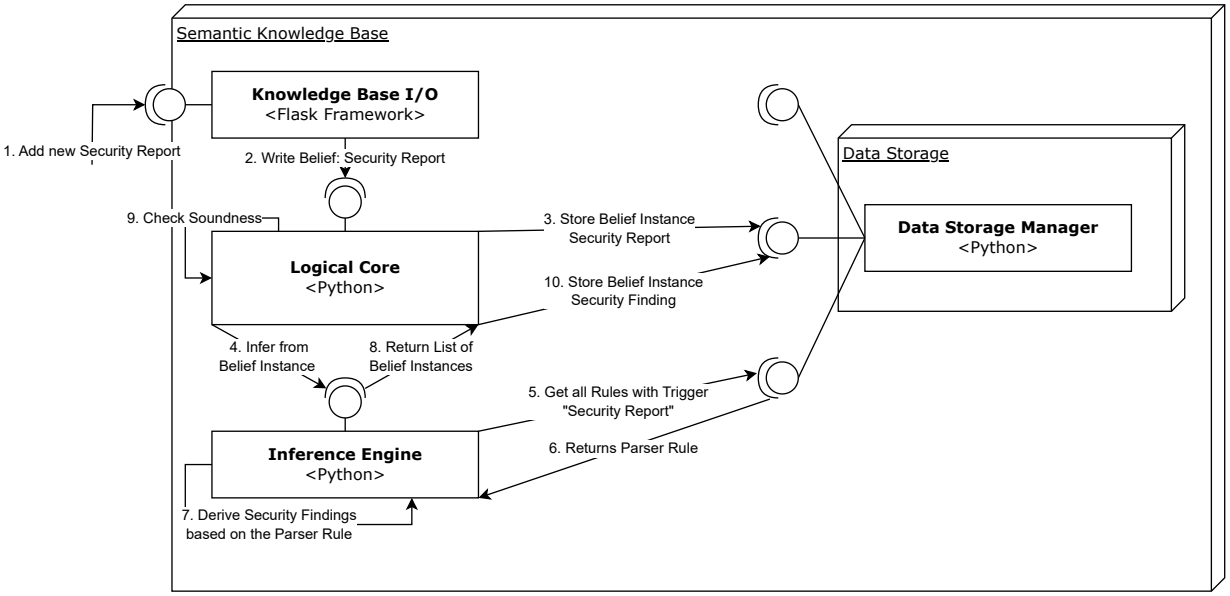

Figure 3.7: Example Process for Adding Belief

This time, the *Inference Engine* cannot derive belief as no rules have belief instances of the ”Security Findings” class as trigger. Therefore, the *Logical Core* concludes that the knowledge base is sound.

3.5.3 Example 2: Read and Delete Belief

In our second example, an external actor intends to acquire currently existing data in the knowledge base and decides to delete the security report that was added during the first example. Our second example starts with the knowledge base state reached after the first example.

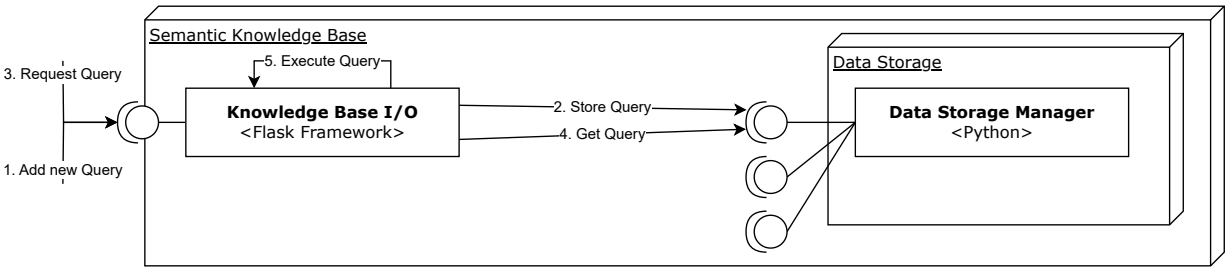

Figure 3.8: Example Process for Reading Belief

The read access to the knowledge base is depicted in Figure 3.8. To receive the necessary view on the data, our external actor first adds a query aggregating the belief instances (1). Since this new query does not affect the soundness of the knowledge base, the *Knowledge Base I/O* directly adds it to the data storage using the interface (2). This newly added

query is afterward requested by the external actor (3). To fulfill this request, the query is searched (4), and the program code belonging to the query is executed by the *Knowledge Base I/O* (5). In the end, the data resulting from the query execution is returned to the external actor.

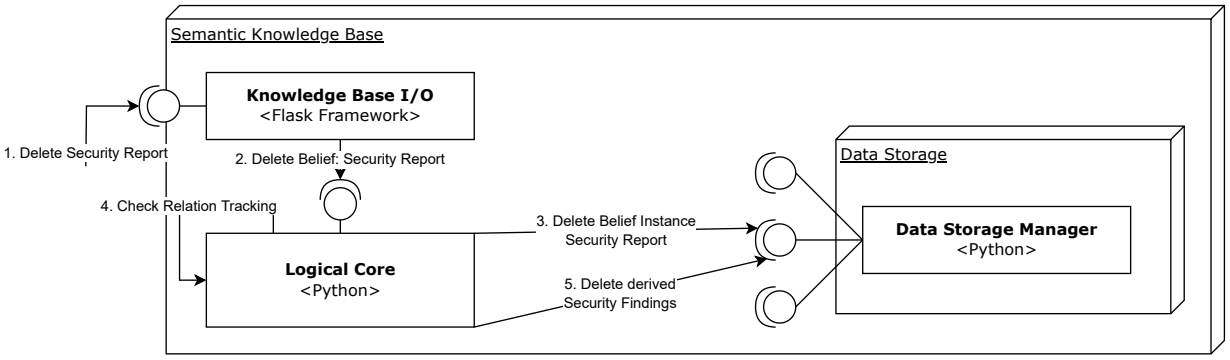

Figure 3.9: Example Process for Deleting Belief

In a second step, the external actor intends to delete the formerly added security report, as shown in Figure 3.9. First, the request to delete the belief instance is sent to the *Knowledge Base I/O* (1). As this deletion operation potentially infringes the soundness of the knowledge base, it is forwarded to the *Logical Core* (2). The *Logical Core* deletes the belief instance by using the data storage interface (3) and starts the maintenance of soundness(see right diagram of Figure 3.5). According to the maintenance strategy, the *Logical Core* identifies all belief instances derived from the deleted security report and checks the number of source elements used during the inference. As all security findings have been derived exclusively from the security report, they are also deleted (5). This results in another maintenance operation, as belief instances were deleted. As these security findings were not used to derive further belief instances, the maintenance operation terminates as a sound knowledge base state is reached.

3.6 Preliminary Evaluation

To ensure our platform solves the initial problems, we conducted a preliminary evaluation with the implemented version of the semantic knowledge base. This section describes how this evaluation was conducted and presents the results. Finally, we discuss the evaluation results and their impact on the rest of the thesis.

3.6.1 Evaluation Planning

The evaluation was planned following the initial problem description that the knowledge base attempts to solve. Section 3.1 presents the five challenges for evaluating the instance of our semantic knowledge base concept. For each of the five challenges, we defined a

question that the evaluation should answer and a strategy to acquire an answer to each question, documented in Table 3.1. To analyze every criterion, the knowledge base must be tested for its functionality.

Challenge	Question	Evaluation Strategy
Functionality	Does the platform orchestrate and document the data processing	Test the documentation for each processing step
Automation	Does the platform automate the data processing	Measure manual effort for operating the knowledge base
Fast Feedback	Is the feedback provided by the platform fast	Measure the feedback time after new data is added
Modular Data Processing	Are arbitrary processing operations supported	Test operation success for each category of inference structure
Continuous Data Processing	Is the knowledge base maintained sound throughout changing operations	Manually evaluate soundness after a defined set of operations

Table 3.1: Knowledge Base Evaluation Question

To achieve a realistic setup, four rules were established representing the four inference structures described in Subsection 3.4.1:

- Parser: One Security Report results in multiple Security Findings (1 - *)
- Duplicate Clustering: Multiple Security Findings result in one Cluster (* - 1)
- Aggregator: One Cluster results in one Aggregated Finding (1 - 1)
- Recommender: Multiple Aggregated Findings result in multiple Recommendations (* - *)

These four rules further necessitate five belief classes. The *Security Report* stores the explicit report data provided by external actors, which is parsed into multiple instances of the *Security Finding* class. Duplicate instances are identified between them, and the resulting clusters are stored as instances of the *Cluster* class, which is transformed to an instance of the *Aggregated Finding* class by aggregating the data of all findings contained in the cluster. Finally, the information provided by the aggregated findings is transformed into instances of the *Recommendation* class by deriving actionable recommendations for project teams from it. Furthermore, the data must also be accessible to external actors. Towards this goal, we introduce five queries, each listing all instances of one belief class.

To analyze how far the knowledge base covers the identified problems, a real-world simulation of its operation must occur. Therefore, a set of add, delete, and read operations are executed, mimicking a software development project. Angermeir et al. suggest that tools that can be directly integrated into pipelines are more likely used in modern software

development than ones with higher integration effort [11]. Therefore, we consider the standard tools integrated by default in Github and Gitlab. As only Gitlab[1] informs about the used secrets scanner and dynamic application security testing tools, we select the respective tools Gitleaks and OWASP ZAP. To cover static application security testing and third-party vulnerability scans, we decided for the Github[2] tools CodeQL and Dependabot, which have also been referenced by Angermeir et al.[11]. This results in four tools producing data for our knowledge base. As a system under test, we use version 8.7.2 of JuiceShop[3]. It is a vulnerable web application generally used for hacking challenges. However, it also provides a variety of publicly known vulnerabilities common in software. This also solves ethical considerations as, most likely, all findings identified by our analysis are intended. Hence, it is a reasonable product for our evaluation. We tested the code base of this system daily with our selected tools for two months until version 8.7.3 of JuiceShop was released, resulting in 53 reports per tool. Each add operation consists of one report per tool for a selected day. Hence, 53 add operations took place, each comprising four reports being uploaded. Since delete operations are scarce in reality, each add operation is followed by a delete operation with a 10% chance. Such deletions are limited to security report instances since delete operations on derived belief instances might be revoked if new data is added (e.g., deleted cluster instances would be re-inferred). Whenever a delete operation is triggered, it selects a security report instance at random and requests the deletion. Five read operations take place after every add or delete operation, requesting each query once. Before any read operation is requested, the maintenance process of the previous operation must be finished. Otherwise, the data would be constantly unsound. This sums up to 53 add operations with four reports each, approximately 270 read operations, and approximately five delete operations.

The data to evaluate the knowledge base is acquired by three measurements. First, the time to complete each operation in the knowledge base is tracked and mapped to the respective operation. This provides insights into the feedback time. Second, the results of the read operations are stored and manually evaluated against a ground truth to analyze the soundness and documentation properties of the knowledge base. This indicates whether the information provided by the knowledge base is correct. Combining the first and second measurements provides further insights into the success of each processing step. Finally, the effort for initializing the knowledge base is measured. Even though the manual effort during operation is limited to sending API requests, the initial overhead for the knowledge base creation still impacts its automation capabilities. The results acquired by measuring these aspects are presented in the next subsection.

3.6.2 Evaluation Results

The evaluation of the semantic knowledge base consisted of 53 add operations, 271 read operations, and six delete operations. During the entire evaluation period, the knowledge

[1]https://docs.gitlab.com/ee/user/application_security/secret_detection/
[2]https://docs.github.com/en/code-security/getting-started/securing-your-repository
[3]https://owasp.org/www-project-juice-shop/

base never reached an unstable state and could always respond to read operations after the maintenance of soundness was completed.

The first measurement in advance to the actual evaluation protocol, was the measurement of manual effort. The manual effort to create the belief classes varied between 19 and 35 minutes. The necessary time correlated with the size of the belief classes, implying that belief classes with more attributes took longer to implement. The queries took approximately 15 minutes each without major differences between queries. The most significant manual effort was required for the creation of the rules. While the Parser rule for the selected four tools took 2:15hrs, the Aggregator rule required 3:45hrs. This time was almost doubled for the Duplicate Clustering, requiring 6:20hrs for the implementation. The most manual effort was required for the Recommender, which took 7:00hrs to implement. All times are listed in Table 3.2. In summary, the initialization of the knowledge base took 22 hours and 53 minutes.

Category	**Implemented Component**	**Effort in Hours**
Belief Class	Security Report	0:19
Belief Class	Security Finding	0:34
Belief Class	Cluster	0:23
Belief Class	Aggregated Finding	0:35
Belief Class	Recommendations	0:28
Query	Security Report Query	0:16
Query	Security Finding Query	0:15
Query	Cluster Query	0:14
Query	Aggregated Finding Query	0:16
Query	Recommendations Query	0:13
Rule	Parser	2:15
Rule	Aggregator	3:45
Rule	Duplicate Clustering	6:20
Rule	Recommender	7:00
Summary	Summary	22:53

Table 3.2: Manual Effort Times for the Initialization of the Knowledge Base

Next, the correctness of the information given by the knowledge base was analyzed. The data of all 271 read operations matched the ground truth dataset that was constructed according to the series of knowledge-base-changing operations that occurred before the read operation. Hence, no unsound data was found during the evaluation.

Finally, the time required for each operation was collected. For read operations this time increased with more belief instances stored in the knowledge base. However, this only affected the queries if more belief instances were also returned, excluding, e.g., Cluster instances, as they solely increased marginally with more reports being added over time. The response time for read operations varied between 0.1 and 0.25 seconds. For delete and add operations, the maintenance overhead outside of calls to the inference engine

was under 1 second. However, all data storage calls, comprising addition and deletion of belief elements, scaled with the request size, implying that, e.g., requests deleting several elements took more time than deleting a single element. Moreover, calls to the inference engine notably impacted the maintenance time. Rules with one source element, like the Parser or the Aggregator, showed a static execution time, while rules with multiple source elements, like the Duplicate Clustering or the Recommender, scaled with the number of belief instances covered. In Table 3.3, the minimal and maximal encountered execution times are listed for each rule.

Rule	Min Execution Time	Max Execution Time
Parser	0.2 seconds	0.4 seconds
Aggregator	0.1 seconds	1.3 seconds
Duplicate Clustering	0.8 seconds	12 seconds
Recommender	0.2 seconds	6.1 seconds

Table 3.3: Rule Execution Times

The maintenance of delete operations has shown to be at least as fast, but in most cases faster than add operations. The worst-case scenario for maintenance was the last addition of a security report. While the Parsing and Aggregation times stayed at 0.3 and 1.1 seconds, the Duplicate Clustering took 12 seconds, and the Recommender finished after 6.1 seconds. This resulted in almost 40 seconds of processing time, as the duplicate clustering and the recommendations had to be executed twice.

3.6.3 Discussion and Limitation

The evaluation results indicate that our semantic knowledge base is a promising platform for the management of security findings, but it also discloses some limitations. In the following, we discuss these results according to the problem description of this chapter.

First, the correctness of data produced by the inference rules and the ability to add exemplary rules indicate that our knowledge base supports arbitrary data processing. The correctness of data further showed that it maintains a sound state throughout multiple operations. Moreover, each inference was traceable whenever the knowledge base reached a sound state. Therefore, the platform also orchestrates and documents the data processing correctly. Even though the platform automated all of the data processing, a considerable manual effort is necessary to instantiate the knowledge base. Especially, the complexity of Elasticsearch queries increased the development time, as seen by the effort required in hours of the Recommender and Duplicate Clustering listed in Table 3.2. Both rules contained additional Elasticsearch queries that have been mentioned as one of the reasons for the increased effort. Hence, the knowledge base fulfills the automation requirement, but the investment upfront for its setup must be considered when using it in practice. Finally, the maintenance time between data being added to the knowledge base and results being shown to external actors is substantial, so the fast feedback requirement is not entirely

fulfilled. With up to 40 seconds of processing time for new data, the timeframe can only be considered acceptable. However, the *Bottom-Up* principles followed by the logical core ensures that feedback can be given instantaneously whenever a sound knowledge base state is reached. Further, the evaluation reinforces the importance of well-constructed Elasticsearch queries, which are key for efficiently handling belief instances. Consequently, improving these queries might further enhance the knowledge base performance.

Several threats constrain the validity of our preliminary evaluation results. First, the data collected on the manual effort can not be generalized, as it was set up by somebody with knowledge of the platform and its interfaces. Hence, another developer implementing rules will most likely perform worse. In a realistic setting, however, the knowledge base would already contain several queries, belief classes, and rules, as certain inferences are helpful for the management of security findings in general. Second, the degree of realism achieved by this evaluation is limited as the overall setup solely mimics an actual project. We still believe this threat is acceptable, as the evaluation is considered preliminary. The third threat is the relation between our knowledge base performance and the underlying hardware it runs on. The knowledge base received four Gigabytes of RAM and two virtual cores for processing in our lab setup. Any other setup might increase or decrease the performance and hence falsify the data shown in our evaluation. Finally, the modularity of our platform is not proven by this evaluation. Even though we tested all possible relations between belief instances, our evaluation setup cannot verify that the knowledge base rule structure supports every type of rule. Whether the platform supports all necessary inferences will, therefore, be identified in the upcoming chapters.

3.7 Conclusion

In this chapter, we presented our concept and implementation of a semantic knowledge base as a solution for a platform that orchestrates, automates, and documents the management of security findings and its related data processing. Our platform takes advantage of the automated documentation and continuous inference properties of semantic knowledge bases and combines them with the flexibility and transparency of logical reasoning. We presented the concept and implementation of our semantic knowledge base and instantiated it with four inference rules to evaluate it in a simulated project environment.

This chapter comprises three contributions. Our first contribution is the concept of a modular semantic knowledge base supporting the flexible addition of data classes and inference rules. The second contribution is the proof of applicability to real-world scenarios by implementing and initializing it with an initial set of rules and data. Finally, we preliminarily evaluated our implementation against the initial problem description and concluded that our semantic knowledge base yields a satisfactory solution to the identified problems.

Consequently, we conclude that our semantic knowledge base is a reasonable platform for managing security findings in modern industrial software development projects. The final conclusion for selecting a semantic knowledge base as our platform can only be drawn

when the necessary inference functions are known, and the platform is evaluated in industrial software development projects.

Chapter 4

Improving Data Quality of Security Findings

One of the most significant challenges of managing security findings in modern industrial software development is the quality of reports generated by automated and manual security checks. Affected by challenges like duplicates, low semantic textual value, and varying terminologies, the data quality must be improved before findings can be managed efficiently. In this chapter, we present the problems necessitating the improvement of the security findings data quality and describe our solution approaches for the collection, parsing, deduplication, and data enrichment of security findings. The integration of these solution approaches into our semantic knowledge base, and its preliminary evaluation concludes this chapter. The research on security findings deduplication was further published in [161], while the preliminary evaluation on the parsing and deduplication is published in [194].

4.1 Problem Description

In our preliminary problem analysis, we identified the need to improve the data quality of security findings. According to Subsection 2.3.3, four requirements affect the preprocessing of security findings. These requirements include

1. The **Collection** of security findings data from the respective sources and its storage at a central instance

2. The **Parsing** of security findings to a common data format

3. The **Aggregation and Correlation** of security findings to avoid duplicates and identify dependencies

4. The **Enrichment** of security findings

The research on these problems and their respective solution approaches contribute to our second objective. However, any solution approach must additionally mind the principles arising from modern software development:

- Topic III: The manual data quality improvement must be minimized
- Topic IV: The data quality improvement processing must provide fast results
- Topic VIII: The data quality improvement processing must be continuously executable
- Topic XII: The data quality improvement must consider data from all development stages

This results in four problems this chapter aims to solve. These four problems are described in the following paragraphs.

Collection of Security Findings

The first problem is represented by the distribution of security checks in modern industrial software development projects. A single platform rarely covers the entire security testing strategy of a project. In most cases, the security findings generating checks are distributed across the entire software development lifecycle, ranging from tests in CI/CD pipelines to monitoring in the operations environment. Furthermore, not all tests are automated or orchestrated. A central activity in any software development project is a Penetration Test. These tests are, by definition, primarily conducted manually and solely supported by security tooling. However, these tests also contain security findings that must be handled centrally. To fulfill the requirement of a central security findings management, a collection of security findings must adhere to strategies customized to the different sources of security findings.

Parsing of Security Findings

The second problem is the varying textual representation of security findings depending on the data source. Similar to the collection of security findings, the variety of security activities producing security findings results in different data formats and various terminologies for the information transported by it. To ensure a common understanding of the provided information by a human user and construct a reliable data structure for subsequent process steps, the data provided by the security activities must be parsed to a common data model. This covers the transformation of source data to security findings and the development of a common data model supporting actors of the security findings management process and subsequent processes.

Aggregation and Correlation of Security Findings

The third problem is the existence of duplicate security findings between multiple security reports. To avoid gaps in the coverage of a software product, employing activities with

overlapping coverage is common in industry. Identical test cases can result in duplicate security findings from different sources, increasing the workload for practitioners. To avoid this unnecessary effort, security findings must be correlated and duplicates eliminated. This includes the identification of duplicates as well as their elimination by aggregation.

Enrichment of Security Findings

Finally, each source of security findings provides a different level of information quality and knowledge density in its reports. While the information provided by manual activities like Penetration Tests often provides detailed and reliable insights about the software, this is not the case for less sophisticated tests like pattern matching. To construct actionable information for actors of the security findings management, certain information must be provided in every security finding. The identification of this information and a strategy for the enrichment of security findings is our last problem in this chapter.

Each solution approach must also comply with modern industrial software development principles. Consequently, they should be mostly automated within our platform, achieve fast results, allow for continuous process execution, and ensure that data arises throughout all software lifecycle phases.

4.2 Related Work

Processes to improve data quality is a research field spread across most computer science domains. Whenever data is processed, its quality and soundness are assessed or improved, affecting the domain of secure software development as well. This section presents the work related to the collection, parsing, deduplication, and enrichment of security findings.

Data collection from various sources is a widespread problem in empirical research. With increasing research on Social Media or investigation of publicly available data like software repositories, the demand for continuous, automated, and reliable data collection is reaching an all-time high. When collecting data from specific platforms like Social Media providers, typical challenges include rate limitations or access to authentication keys. This research type mainly uses APIs exposed by platform providers [105], resulting in collection strategies dealing with the API employed or efficiently constructed requests [171, 166, 113]. When broadening the focus to generic web content, the challenges shift to the efficient collection of data structured in website components [41, 64]. The work by Ullah et al. [186] on automated security compliance checking for the cloud is particularly interesting for our use case. Their data collection approach considers APIs, tool-specific interfaces, administrative protocols for machine access, and manual data upload. With these measures, they collect data from security activities necessary for claims on security compliance of cloud infrastructure. Similar to the research in the Social Media domain, our collection approach has to keep authentication procedures in mind when data is collected. Since security tools contain confidential information on the security shortcomings of software products, this data is protected by authentication and authorization mechanisms. However, receiving

access keys, rate limitations, or unstructured data should not be a problem in industrial software engineering, as the project team desires the collection and data is supposed to be mostly structured.

A typical subsequent action after data is collected is the parsing of information to a format that can be processed later. In the work by Schäfer et al., they identify the necessity to parse HTML data containing security intelligence data into a structured file solely containing text [163]. They propose a system for collecting security intelligence information by scraping the dark web and solving the challenge of processing different HTML site structures by employing a separate parser model for each forum they encounter. More closely related to the security findings our methodology aims to manage is the work by Alqahtani et al. [6]. They suggest a modeling approach to trace security vulnerabilities between software repositories and vulnerability databases. Since their data originates from multiple sources that employ different formats, they developed a unified ontological representation and parsed all data to this format. Our approach will similarly rely on a homogeneous structure for all security findings and a respective parsing operation.

The existence of duplicate findings across several security activities reporting about a product is an existing challenge in industrial software development. However, the related work on the deduplication of findings is seemingly scarce. By reducing the scope to vulnerability deduplication, Peng et al. proposed an approach for clustering Use-After-Free (UAF) vulnerabilities [143]. Even though related to the deduplication of security findings, their process uses UAF-specific properties for the deduplication, which is not given in every finding. More closely related to our work is the procedure by Huang et al. [67]. In their work, they classify vulnerabilities of the National Vulnerability Database (NVD) by clustering them according to their textual properties. Even though this neither covers all types of security findings, as NVD solely contains publicly known vulnerabilities nor does it identify duplicates, the idea of investigating textual properties for deduplication is promising. Another related domain is the management of software bugs, which are often considered to include security findings. Research in the domain of software bug deduplication is exploring multiple techniques ranging from deep learning networks [153] to Natural Language Processing (NLP) using Linear Discriminant Analysis (LDA) [101, 1]. Therefore, an approach for deduplicating security findings by processing the natural language contained by them seems reasonable. In multiple other domains of the software engineering field, the application of NLP techniques has solved existing challenges [38, 107, 1]. Using Latent Semantic Indexing, Kuhn et al. [97], for example, analyzed linguistic data in source code. Similarly, Tan et al. assessed software quality by creating clusters of related problems in the source code and reducing the effort to review code [176]. Even though their approach focuses on all quality aspects of code, the concept of semantic clusters is promising for our use case. The method of clustering semantically similar data is applied in various other publications, including similar requirements [46] or app reviews [160]. However, our literature analysis also showed that no publications related to NLP focusing on the deduplication of security findings exist.

The enrichment of existing information about the shortcomings of software in the security domain is also a topic that has been addressed in recent scientific literature. In 2019,

Viertel et al. proposed an approach for detecting vulnerabilities in software code by clone detection from publicly known vulnerabilities [188]. As part of their approach, they use the metadata provided by CVE to enrich the vulnerability information. One step further is Glanz et al.'s proposal, which suggests the enrichment of vulnerability entries in the NVD database with the affected software versions and information on potential fixes [55]. Hence, their recommendation does not focus on reports provided by a tool or activity but applies the enrichment at the origin of information used by many tools: the vulnerability database. Also, Althebeiti and Mohaisen follow this idea by proposing a process that enriches vulnerability descriptions of publicly known vulnerabilities [8]. They leverage the hyperlinks often given in the NVD database to collect more information about the vulnerability from third-party sources and construct a description from the found data. We conclude from the existing work in this domain that the enrichment of security findings using vulnerability databases is a commonly used approach. However, the proposals to improve the data provided by the vulnerability databases indicate that the existing data is not optimal.

4.3 Security Findings Collection and Parsing

The first step towards improving the data quality of security findings as part of the management process is the collection and parsing of security data to the desired format of security findings. Most projects distribute their security analysis strategy across multiple security activities, resulting in distributed data sources for the security findings. Moreover, the format and terminology used to store the data in security reports differentiate between sources. Therefore, this section presents the collection strategy for security reports and the parsing process including the common data model every security finding is parsed to.

4.3.1 Collection

The first step to improve the data quality of security findings is the collection of security data containing findings from their respective sources. In contrast to most of the related work in this domain, we do not just consider the proactive collection of data from multiple sources, but also the reactive acceptance of data. Especially tools that are orchestrated in CI/CD pipelines as runners solely have short-lived data storage. Therefore, the data must be sent in real-time after the tests have been conducted. Otherwise, the security data is deleted shortly after. Consequently, we identify the need for two collection strategies.

For the first collection strategy, the security data is proactively collected at its sources. This strategy is employed for any source activity that maintains a data storage for its security data. The collection itself is agnostic of the activity type and its storage and solely depends on the interface provided to the data. Following the strategy by Ullah et al. [186], our strategy covers proactive data gathering at APIs, custom tool interfaces using the HTTP/-s protocol, and accessing machines via administrative protocols. While the primary use case for the proactive data collection is standalone activities outside of CI/CD

pipelines like security monitoring tools, it can also apply to reports from manual activities where reports are stored in a suitable storage like a database.

However, the proactive strategy can only be applied in scenarios with persistent data storage. Most security tests orchestrated in CI/CD pipelines are deployed on temporary "runners" that are deleted after completing their job, affecting the data stored on them as well. Therefore, our strategy must be extended to include reactive collection, where data can be sent to our methodology. This reactive collection is realized by an interface supporting data uploads. Including the fourth data collection approach mentioned by Ullah et al. [186], this interface must also be available for manual data uploads by human users. A typical use case would be the manual upload of a security report by testers after conducting a Penetration Test.

This results in a two-fold collection strategy depicted in Figure 4.1.

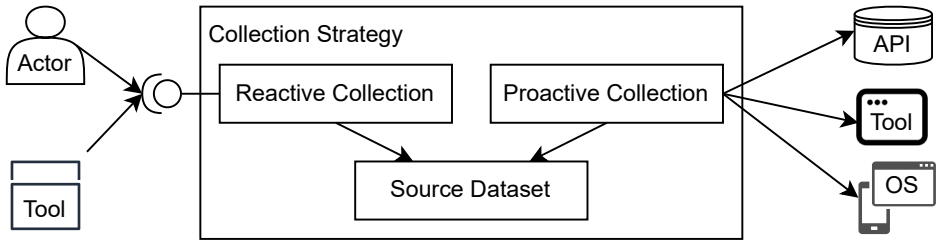

Figure 4.1: Proactive and Reactive Collection Strategies

Integration Process

To integrate our strategy of collecting security findings data into software development projects, particular preceding steps must be fulfilled. First, all sources of security data in the project must be identified. Next, the collection approach for each source must be known. All sources using the reactive strategy must receive access to the interfaces the reactive component provides. For those sources requiring the proactive collection of data, access to the data must be ensured, including authentication and potential network access. Moreover, the triggers for data collection must be defined and documented, as the collection process must be able to trigger the proactive collection.

4.3.2 Security Findings Datamodel

Another fundamental requirement for the preliminary improvement of data quality is parsing all security activity reports to a common data format. This necessitates a data format suitable for all sources of security findings. This subsection presents the data model and the underlying research to achieve it.

The data model has to fulfill two requirements. First, it must be able to comprise and document all relevant information provided by security activities. Since these activities highly differentiate in the type of information they transport, depending, e.g., on the

software aspect they analyze or the stage of the software engineering lifecycle they are integrated in, the data model must cope with various data fields and terminologies. Second, the model has to contain all information necessary for the later stages of the security findings management process. Otherwise, the parsing would inflict data loss with a disastrous impact on any stage requiring this information.

The second requirement is especially challenging since not every inference in our knowledge base is known upfront. Consequently, we cannot distinguish information that might become relevant for the security findings management from irrelevant information. Therefore, all data provided by security activities with potential impact on the security findings management must be included in our data model. This automatically fulfills the first requirements since the data model comprises all data fields. However, the varying terminology between activities still represents a challenge. Towards a common terminology, the content of all fields offered by security activities must be semantically understood, and a field expressing its semantic meaning must be added to the data model.

Identification of Sources

To construct the data model, we first create a representative list of security activities that generate security findings and use exemplary reports of these activities to identify common data fields. However, no generally accepted list of security practices in modern software engineering exists. Angermeir et al. categorize security activities into five types, namely "3rd Party Vulnerability Scanning", "Static/Dynamic Application Security Testing", "Secure Configuration/Hardening", "Compliance/Hardening Checks" and "Secrets Management" [11]. The OWASP Guideline for DevSecOps and security automation extends this list with "Scan git repositories for finding potential credentials leakage", "Software Composition Analysis", "Static Application Security Test", "Infrastructure as Code Scanning", "Interactive Application Security Testing", "API Security", "Dynamic Application Security Test", "Cloud Native Application Protection", "Infrastructure scanning", "Continuous Scanning from other tools", and "Compliance Check" [206]. These two sources already show substantial conflicts in the clustering and selection of activities. As a source for the security findings management methodology, we solely consider activities that inform about shortcomings. Therefore, the "Secure Configuration/Hardening" section mentioned by Angermeir et al. is, for example, not relevant to our data model. Combining the security activities mentioned above with the three most influential documents for the security findings management (BSIMM 12 [26], OWASP SAMM [140], Safecode [158]), we define the activities in the "Category" column of Table 4.1 as relevant sources for our methodology.

Area	Category	Instances
SAST	Secret Scanning	• git-secrets • Gitleaks • truffleHog

SAST	3rd Party Component Vulnerability Scanning and SCA	• OWASP Dependency-check • RetireJS • Clair • Anchore • Trivy • Dependabot • Snyk
SAST	Code Review (Application and Infrastructure Code)	• SonarQube • CodeQL • Coverity • Brakeman • Bandit • Checkov • tfsec • terrascan • KICS • Manual Code Review
SAST	Configuration and Hardening Checks	• conftest • InSpec • Kubebench • Bane
SAST	Security Requirements Testing	• Unit Testing
DAST	Security Functional Testing	• Unit Testing
DAST	Dynamic Interface, API and Black-box Testing	• ZED Attack Proxy • Acunetix • Netsparker
DAST	Interactive Application Security Testing	• Contrast
DAST	Penetration Testing	• Customized Report
Operations Security	Security Operations Monitoring (Logs, Configuration, Hardening, Patch State)	• Wiz • Sysdig • Crowdstrike • Qualys
Operations Security	Infrastructure and Network Testing	• driftctl • Kubediff • NMap

Table 4.1: Security Activity Categories and their respective Tools

The reviewed documents solely mention an activity type while omitting specific tools or reporting schemes that should be used. An exhaustive list of all potential report formats would necessitate an in-depth review of all security tools and activities on the market. Due to the ever-changing list of new tools joining the market and others reaching an end-of-life status, such a list is connected to an unreasonable effort as it solely provides a snapshot. Therefore, we solely consider the most prominent reporting schemes used by our industry partner and mentioned by OWASP [206]. These exemplary instances of the security activities are listed in the rightmost column of Table 4.1. Exemplary reports or data formats are publicly available for most of these activities. Due to the confidential

nature of Penetration tests, these are not publicly disclosed. Consequently, we requested an exemplary test report from our industry partner.

Datamodel Construction

Our data model is constructed using this list of commonly used security reporting schemes. For its construction, we follow a protocol that is applied to each reporting scheme in the list.

1. Identify all data fields a single finding consists of
2. Eliminate any field from this list that has no impact on the security findings management
3. Map each field to the existing list of common data fields based on its semantic meaning
4. For any field that cannot be mapped, determine the semantic context and add it to the common data fields

Applying this protocol to the Pentest report retrieved from our industrial partner would result in the following procedure. First, all data fields are identified. In our example this results in ten data fields: "ID", "Title", "Finding Number", "Cricitality", "Summary", "Background", "Approach", "Approach", "Prerequisites", "Impact" and "Countermeasures". We eliminate the "Finding Number" field, as this solely relates to the order of each finding in the report without any relevance for the later management of these findings. Since our list of common data fields is initially empty, each field is added with its respective semantic context, resulting in the list depicted in Table 4.2.

Data Field	**Semantic Context**
ID	Unique identifier of the finding with respect to the activity
Title	Describe the finding in one sentence
Criticality	Criticality of the findings for the security of the software
Summary	Brief description of the finding
Background	Context of the finding; explains components/actors/aspects as background information necessary to understand the finding
Approach	Explanation on how the finding was identified by the activity
Prerequisites	Circumstances or resources necessary for the exploitation of the finding
Impact	What are the consequences of an attacker exploiting the finding
Countermeasures	Recommendations on how to resolve or mitigate the finding

Table 4.2: Initial List of Common Data Fields

Resulting Datamodel

The list of common data fields resulting from applying the protocol to all reporting schemes can be found in Table 4.3. During the analysis, several challenges arose. The first challenge is the varying information provided for specific fields like location. While code reviews pinpoint particular lines in the code, 3rd party component checkers mainly mention the respective component. Furthermore, this information is sometimes spread across multiple fields or summarized within one. To cover these different formats and information types, the data remains remains under-specified and is addressed by the parsing in a second step. The second challenge is activities that test using rulesets tailored towards each project. In these cases, the report and its data field are closely coupled to the test case written by the project team. To compensate for the unpredictable content of custom rules, we included the JUnit-XML format in our analysis as it is a common reporting format in CI/CD environments. Finally, some tools were extremely verbose, providing detailed information regarding the type of activity. As this information was only provided by one category of security activities, we added the "Activity-Specific Information" field to our data model. The additional fields "Source" and "Source Version" were introduced to provide detailed information on the activity that generated the finding.

Moreover, multiple distinctive features in the reports were identified. First, the most common data fields include the location of the finding (Location: 27), a summary of what was found (Title: 20, Description: 19), and the impact of the finding on the software security (Severity: 17). While almost every report informs about the finding location, the title is sometimes replaced by an identifier or rule. Especially activities testing software for publicly known vulnerabilities like the "3rd Party Component Vulnerability Scanning and SCA" provided the respective CVE-IDs with each finding. Another aspect is the varying terminology for a multitude of fields. An example would be semantically similar field names "check_name" and "test_name", which contain either the title of a finding or an alphanumeric identifier of the applied test case. Furthermore, the severity of a finding was found in field names including "risk", "criticality", or "impact" while not following the commonly accepted definition discussed in Subsection 2.1.2. These inconsistencies in the terminology reinforce the necessity of a common data format.

Data Field	Semantic Context	Terminology of Sources	Frequ.
Location	Data on where the finding can be found in the software. Either locator in the source code, repository or running software	location, filename, line_number, commit, file, repository, filePath, component, software, feature-name, package, cpe, PkgName, displayFile, displayComponent, code_block	27
Title	Describes the finding in one sentence	Title, name, code, check_name, query_name, shortname, Vulnera-bility Name,	20
Description	Brief description of the find-ing	summary, description, info, mes-sage, text, displayType, issue_text, rule_description, desc, otherinfo, DetailedInformation, test_info	19

Severity	Criticality of the findings for the security of the software	criticality, severity, issue_severity, riskdesc, riskcode, impact	17
Identification Type	Rule that was tested to identify the finding or the approach for the check	policyId, test_id, Approach, Detector Type, evidence, detection, user_input, test_name, check_id, check_class, rule_id, pluginid, Rule Name, test_number	15
ID	Unique identifier of the finding with respect to the activity	ID, issue, key, rule, cid, query_id, Vulnerability ID,	11
Timestamp	Timestamp, when the finding was last identified	timestamp, reportDate, last_evaluation, LastModifiedDate, creationTime, updateDate, lastDetected, ScanTime, LastSeenDate	11
Countermeasures	Recommendations on how to resolve or mitigate the finding	Countermeasures, fix, fixedby, resolution, fixedIn, action, guideline, solution, Recommendation, test_desc	10
CWE	Common Weakness Enumeration identifier and descriptions	cwe, CweIDs, cweid, CWEList	9
CVSS	CVSS Score and Vector	cvssScore, cvssvec, cvssVector, cvss_v3, cvss_v2, cvss, CVSSv3	8
Certainty	How certain is it, that the finding is a True Positive	is_verified, confidence, issue_confidence, Certainty	7
Finding Category	Umbrella category this finding belongs to	category, type, warning_type, Category	7
CVE-ID	Unique identifier in the Common Vulnerabilities and Exposures program	cve, name, vulnerability, VulnerabilityID	5
Prerequisites	Circumstances or resources necessary for the exploitation of the finding	Prerequisites, exploit, audit	3
Impact	What are the consequences of an attacker exploiting the finding	impact	2
GHSA	GHSA Identifier	GHSA	1
Fixing Effort	Time necessary to fix the finding and consequently the technical debt	debt, effort	1
Severity Source	How was the severity calculated or defined	SeveritySource	1
Background	Context of the finding; explains components/actors/aspects as background information necessary to understand the finding	Background	1

Activity-Specific Information	Field for leftover data, only relevant for the specific activity	N/A	N/A
Source	Activity name that provided the finding	N/A	N/A
Version	Version of the activity that provided the finding	N/A	N/A

Table 4.3: Common Data Model derived from Security Activity Reports

The construction of the data model results in the fields listed in Table 4.3 with their respective semantic meaning. This model covers any information provided by our representative list of security activities that might be relevant to the security findings management process. Using the "Activity-Specific Information" it is further possible to include currently not considered information. The following subsection describes the strategy for parsing reports to the data model.

4.3.3 Parsing to Datamodel

In a second step, every security report collected previously is parsed to the common data model described above. The constraints for the parsing procedure and its realization are described in this subsection.

During the construction of the data model, several constraints arose that must be addressed by the parsing procedure. The analysis of the exemplary security reports showed that the parsing of the source data to the constructed data model has to support several operations and cope with the particularities of some activities. The most obvious challenge is the list of different data formats the parsing operation must handle. During our analysis, we encountered JSON, XML, CSV, HTML, and PDF documents containing exclusively textual data, at least in a semi-structured format. Hence, the parsing procedure must be able to deal with textual data structured according to multiple formats. To map the information produced by the security activities to the common data model, each report has to be broken down into its unique findings and an instance of the data model filled with the given information. However, reports differ in their structure, as some findings are aggregated by location, title, or CWE-ID, resulting in clusters of findings in some reports that must be split up. Therefore, the parsing operation must be able to iterate through each received data set, depending on the structure of the security report. Moreover, some reports comprise data fields containing information that must be mapped to multiple fields in the data model. Examples include HTML reports, where the first line of the description fulfills our semantic interpretation of a title while the remaining paragraph maps to the description. In these cases, data fields may not be mapped one-to-one, but the parsing operation must be able to split data fields based on regular expressions. Similarly, some reports spread their information across multiple data fields, necessitating a summary of field content. Typical for this occurrence is the location field. Especially activities analyzing static properties of software can pin the exact locations of findings in the code.

Consequently, this information is often split across fields informing about the location in the repository, the exact line and column of the finding within the file, and a code snippet of the finding. In contrast, dynamic activities are mostly limited to stating the general area of a finding's location, resulting in one data field informing, e.g., about the affected machine's server resource or IP address. Therefore, the parsing operation must be able to merge multiple data fields according to predefined rules. Finally, some reports are missing data that are inherently given by the activity itself (Severity Source) or the circumstances during the parsing operation (Timestamp). In these cases, a preliminary enrichment of each parsed finding is necessary. This enrichment process is described in Subsection 4.5.1.

Summarizing all challenges, the strategy for parsing security activity data to the data model has to support the parsing of JSON, XML, CSV, HTML, and PDF documents with the capability to freely iterate through each report and split or merge the data fields on demand. Moreover, the differentiating formats and report structures require a separate parsing strategy for each security report. Since activities might change the report format over time, a new version of an activity might necessitate a new parsing strategy for this activity. Hence, each activity has a distinct "Parser Model" for each report format, resulting in a mapping between activities with their respective version and "Parser Models":

```
Activity A:
    Version < 1.1:          Parser Model 1
    1.1 <= Version < 2.3:   Parser Model 2
    2.3 <= Version:         Parser Model 3
```

Each "Parser Model" consists of a set of operations to be executed on the respective activity report, resulting in a list of security findings that can be derived from the report. The resulting list of security findings is afterward semantically enriched, as described in Subsection 4.5.1, to improve the information's maturity. To successfully complete the parsing operation, each data model instance must contain at least the data fields "Title", "Location", "Timestamp", "Source" and "Version". These fields have been selected based on their ability to provide the minimal insight necessary for an informed decision about the identified finding. It informs about where the finding is located, what kind of finding occurred, and when it was seen the last time. The preliminary enrichment of Subsection 4.5.1 ensures that these fields are given, if possible. An exemplary "Parser Model" for JSON reports of the Secret Scanner Gitleaks is depicted below in pseudo-code:

```
finding_list = List()
For object in JSON_Element:
    finding = Datamodel()
    finding.title = object.description.split('\n')[0]
    finding.source = "Gitleaks"
    finding.version = "1.0"
    finding.description = object.description.split('\n')[1]
    finding.location = "[" + object.file + "," + object.line + "," +
                       object.column + "]"
    finding_list.add(finding)
```

The example shows how the original report is iterated to access the findings data (for loop), how one data field is split into two for our data model (the description field is divided at the newline symbol), and how multiple fields are aggregated into one (construction of location field). Moreover, the source data lacks any timestamp. Hence, the preliminary enrichment would add this information after parsing.

Integration Process

To integrate the parsing strategy into ongoing projects, all activities producing security reports must be identified. In most cases, this step is already covered by the preliminary integration of the collection. Afterward, a separate "Parser Model" must be written for each activity and the mapping between model and activity versions must be established. Due to the volatile nature of report formats, new versions of activities must be continuously monitored and a new "Parser Model" must be written if the source format changes.

4.4 Security Findings Aggregation and Correlation

A security testing strategy that avoids whitespots in the product coverage is crucial for any industrial software development project. While this strategy ensures that all aspects of the product are covered, it comes at the risk of duplicate coverage. This duplicate coverage of aspects typically results in findings being identified multiple times. Consequently, the occurrence of duplicate or almost identical findings is inevitable. Since these duplicates falsify a project's security overview, their identification and elimination are crucial to developing software efficiently.

This section presents our research on the deduplication of industrial security findings using NLP-based techniques. In the first step, we identified promising NLP-based approaches for the deduplication of security findings and put them into the context of industrial software development. Selecting the best-performing technique, we integrated the selected method into a more extensive process, including the aggregation of duplicate security findings.

The initial research on industrial security findings deduplication was performed as a master thesis by Abdullah Gulraiz in collaboration with the Technical University Munich, represented by Phillip Schneider. For this initial research, the background regarding semantic similarity algorithms and experimental design has been provided by Phillip Schneider. The experiments and analysis of results have been performed by Abdullah Gulraiz. The problem statement and evaluation dataset originate from the author of this thesis. In all remaining aspects, all authors contributed equally.

The collaboration results are published in [161], while the extended research and evaluation are published in [194].

4.4.1 Definition and Identification of duplicate Security Findings

The first step towards identifying and eliminating duplicate security findings in industrial software development projects is to define the relation between two findings considered duplicates. In the first paragraph, we develop our definition of a duplicate security finding, while the second paragraph provides guidance on how to identify this relation in practice.

Definition of Duplicate Security Findings

A first hint on the different types of duplication relations can be found in Subsection 4.3.2. Here, we investigated reports from various security activities and found that some activities aggregate findings according to certain properties. In the following, we discuss four different strategies for the deduplication of security findings.

Some investigated activities aggregated their findings according to their location. This means that all findings at the same location were listed within one cluster. This strategy was typically employed by 3rd Party Component Vulnerability Scanning activities, where all publicly known vulnerabilities affecting the same component were listed under the same location. We define this type of aggregation as ”location-based”. The advantage of this strategy is that all findings at one location can be treated at once. Hence, a team member working on solving issues at one spot in the software can address all problems at this location at once. Moreover, it provides insights into the ”Security Hotspots” of the software, as clusters with more duplicate findings indicate that multiple security shortcomings impact the location.

Another strategy employed the title of a finding as characteristic for clustering findings. All findings with the same content in its data fields were aggregated in these cases, and the respective locations were appended as a list. Hence, all findings in these reports contained a data field listing all applicable locations. We found this approach in Code Review activities where the same insecure coding pattern can be found at multiple places in the software code. We define this type of aggregation as ”problem-based”. The idea behind strategy is that solving the same finding at one location will likely enable the team member to solve this finding at all occasions since the same problem is presumably solved similarly.

The most straightforward approach for identifying duplicates is to consider identical issues at the exact same location as duplicates. Since all investigated activities already ensure this property, the deduplication would happen over time as new reports must be integrated into the existing list of findings. In these cases, the same finding at the exact same location might be identified again or potentially found by another activity considering overlapping product coverage. We define this type of aggregation as ”issue-based”. The advantage of this strategy is that each finding is unique and independent of other findings. Moreover, this deduplication approach requires the least effort as activities already ensure this property.

The last strategy for identifying duplicate findings is typically employed when deciding on actions with the greatest impact on the security of a product. Since multiple findings might be solved by the same action (e.g., selecting pre-hardened system images), this deduplication approach focuses on clustering all findings with the same solution to mitigate

or fix them. We define this type of aggregation as "solution-based". The fundamental advantage of this approach is the opportunity to identify actions with the most significant impact on software security, considering the findings aggregated under the same solution. Especially if multiple findings are all bound to the same root cause, this approach reduces the overall number of findings and enables meaningful decisions.

For our aggregation and correlation of security findings, only one strategy can be applied. Issue-based aggregation and location-based aggregation both share the challenge that their deduplication approach has to cope with the location field. This is problematic, as the location of a finding might change over time considering changes to the code base (a new line of code where the finding is reported) or a modified architecture (the finding is located at another resource of an API). Therefore, these strategies might give an incorrect perception of reality as the findings that are not found anymore are unresolved but can be found at another location. Additionally, different activities refer to the same location in the software with varying terminology, making identifying duplicates a challenge. These problems and foreseeable negative impacts on the security overview cannot be compensated by the advantages of either of the two strategies. Neither the problem-based aggregation nor the solution-based aggregation is affected by the challenges of including the findings' location into the calculation of the duplicate relation between findings. However, identifying duplicates for the solution-based aggregation is challenging in other aspects. Even though it provides the most meaningful insights of all approaches, identifying a finding's root cause and its respective solution is highly complicated and error-prone. Preliminary investigations showed that this stage has a major manual component, limiting the potential of automated processing. Especially dynamic software analysis often does not indicate a finding's root cause and cannot provide insights into potential solutions. Consequently, the deduplication results are prone to a low recall, necessitating manual intervention for this step. Hence, this aggregation cannot provide fast results with minimal user intervention as required by our development principles, excluding it from the list of potential deduplication strategies. Finally, the problem-based strategy has the challenge of different terminologies being used by activities for the same problem. This means that the same problem might be described with varying phrasing, depending, e.g., on the type of activity identifying it. Therefore, the semantic content of each finding has to be considered during deduplication. This strategy presents the best balance between benefits for the security findings management process and the feasibility of including it in our security findings management process.

In summary, we define two findings as duplicates if they relate to the same problem, regardless of where this problem is located in the software product.

Identification of Duplicate Security Findings

Using the last paragraph's definition, we can decide if two security findings are duplicates. This decision is based on the semantic content of each finding, describing its underlying problem. Consequently, all data fields contributing information to this description are crucial for identifying duplicates. According to our data model and the semantic context

of each field, we consider the fields "Title", "Description", "Countermeasures", "CWE", "Finding Category", "CVE-ID", "Prerequisites", "Impact", "GHSA", and "Background" as relevant. These fields all describe the underlying problem to some degree.

One straightforward approach to identify duplicates would be an exact match between these fields. However, a direct matching between fields cannot compare semantic content. The reason why this is problematic is shown in the following example. The 3rd Party Component Vulnerability Scanning and SCA tool Trivy was able to identify the publicly known vulnerability *CVE-2022-43551* [1] in 2022 when scanning for 3rd party dependencies. In its initial reporting, the following description was used:

```
A vulnerability was found in curl. The issue can occur when curl's HSTS
check is bypassed to trick it to keep using HTTP. Using its HSTS support,
curl can be instructed to use HTTPS instead of an insecure clear-text HTTP
step even when providing HTTP in the URL. [...]
```

At the beginning of 2023, this description was changed to:

```
A vulnerability exists in curl <7.87.0 HSTS check that could be bypassed
to trick it to keep using HTTP. Using its HSTS support, curl can be
instructed to use HTTPS instead of using an insecure clear-text HTTP step
even when HTTP is provided in the URL. [...]
```

Even though large parts of the description and its semantic meaning persisted, the syntactic content is not equal. Another change happened at the end of January 2023, where an additional CWE-ID and CVSS score were added. This suggests that the data provided by our activities continuously changes, even if the same finding is referenced. When applying an exact matching of data fields, our exemplary data set would consist of three findings after deduplication, even though all refer to the same problem, resulting in three False Negatives.

This problem is further reinforced, considering that not only *intra-activity* duplicates, as presented in the example, exist in practice but also *inter-activity* duplicates. This term describes duplicates occurring between reports from different source activities. In the case of *inter-activity* duplicates, the same finding might be explained with two different terminologies. An example of these differentiating terminologies can be found when comparing the reports of *KICS* and *tfsec*, two security tools to analyze infrastructure code. The example below shows two findings with different titles referring to the same problem:

```
KICS:
    ALB Not Dropping Invalid Headers
tfsec:
    Application load balancer is not set to drop invalid headers.
```

In these cases, the semantic meaning of both findings is equivalent, but the syntactic format differs. Therefore, we conclude that any solution approach for the problem-based deduplication of security findings must include the semantic meaning of data fields as matching criteria.

[1] https://nvd.nist.gov/vuln/detail/CVE-2022-43551#VulnChangeHistorySection

4.4.2 Evaluation of Natural Language Processing Techniques

The importance of semantically understanding the content of a security finding for its correct deduplication indicates that processing the natural language employed in each finding is crucial for our solution approach. Hence, the processing of finding data and comparison of its semantic similarity to other findings is the foundation for our clustering approach. As discussed in Section 4.2, our use case has no recommended technique in scientific literature. Therefore, we decided to investigate commonly proposed semantic similarity-based techniques, namely knowledge graph-based similarity with *WordNet* [116], *LSI* [99], and *SBERT* [150]. Since these are mostly used as baseline models, their tailoring to our specific use case is possible.

Dataset Construction and Description

The investigation of semantic similarity-based techniques requires a dataset of correctly clustered security findings to measure the performance of each selected technique. Since the semantic understanding of each finding is crucial for the construction of the dataset, two domain experts employed at our industrial partner supported this research by clustering and annotating the findings of given security reports. Due to the differences in semantic content between static application security testing (SAST) and dynamic application security testing (DAST) activities, we decided to split the dataset according to this criteria. Consequently, this construction resulted in two datasets consisting of clustered SAST findings and clustered DAST findings. We define these datasets as the "golden dataset" for the following paragraphs.

Dataset	Activity Category	Instance
Static	3rd Party Component Vulnerability Scanning and SCA	• Anchore • Trivy • Dependency Check
Static	Secret Scanning	• Gitleaks
Static	Code Review	• CodeQL • Semgrep • HorusSec
Dynamic	Dynamic Interface, API and Blackbox Testing	• Arachni • ZED Attack Proxy

Table 4.4: Selection of Activities for the Golden Dataset

The first challenge to construct the dataset was creating a realistic set of reports with security findings containing semantic duplicates. On the one hand, this necessitates a software product with shortcomings in the security domain so that security findings can be identified. To ensure reproducibility and an ethically justifiable disclosure of the resulting dataset, we decided for the open-source, intentionally vulnerable web application

JuiceShop [2]. This application is the target for our security activities and presents the root cause of any identified security finding. On the other hand, a selection of security activities used in practice is necessary. In particular, these activities require overlapping coverage of JuiceShop to introduce duplicates into the dataset. For reproducibility reasons, all activities must be publicly available and free to use at the time of the experiment (2022). The selected activities for both datasets are listed in Table 4.4. We chose seven commonly used SAST tools from three categories and two DAST tools from one category. We disregarded activities like infrastructure scanning on purpose, as this would necessitate the creation of an artificial and vulnerable infrastructure, falsifying the dataset. Consequently, JuiceShop was tested by all mentioned activities, and one report from each activity was added to the dataset.

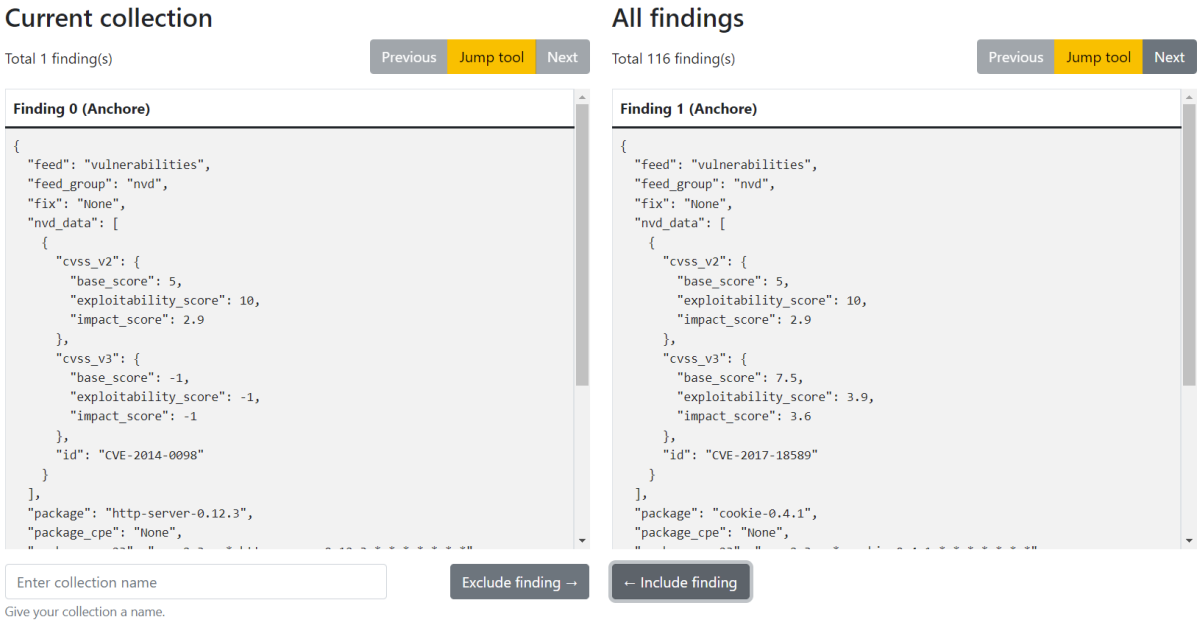

Figure 4.2: SeFiLa Webinterface

This introduced the first challenge for our experiment, as every activity utilized a different reporting format and schema, as described in Section 4.3. Hence, identifying the relevant information to comprehend the semantic content of each finding requires manual effort from our security experts. This includes the parsing of security reports into single findings as well as the identification of data fields with relevance to the semantic meaning of each finding. To ease this part of the clustering process, the Security Findings Labeler (*SeFiLa*) [3] was developed. This web application provides the capability to upload security reports from the selected activity, splits these reports into security findings, and marks the relevant semantic information in each finding. The decision on relevance was performed in collaboration with the domain experts. Moreover, it enables domain experts to create,

[2]https://owasp.org/www-project-juice-shop/
[3]https://github.com/abdullahgulraiz/SeFiLa

manage, and annotate groups of security findings. The visual interface employed by the domain experts is depicted in Figure 4.2. The left side of the figure shows the current collection containing one finding, while the right side shows one of the currently ungrouped findings.

During the process of constructing the dataset, domain experts iterated through all security findings in the respective area (SAST and DAST) and assigned them to groups with the support of *SeFiLa*. The initial, unprocessed dataset consisted of 1351 security findings for SAST and 36 security findings for DAST. Table 4.5 lists the distribution across all activities.

Dataset	**Instance**	**Findings Amount**
Static	Anchore	117
Static	Dependency Check	26
Static	Trivy	31
Static	HorusSec	927
Static	Semgrep	86
Static	CodeQL	91
Static	Gitleaks	73
Static	**Summary**	**1351**
Dynamic	Arachni	32
Dynamic	ZED Attack Proxy	4
Dynamic	**Summary**	**36**

Table 4.5: Distribution of the Original Dataset across Activities

The groups were created by collecting all findings belonging to the same problem and summarizing its semantic content as the group's title.

This manual clustering and annotation process resulted in our golden dataset: two JSON documents comprising a list of deduplicated security findings for SAST and DAST reports, respectively. The golden SAST dataset consists of 183 annotated clusters. 149 of these clusters solely comprise findings of one tool (intra-tool deduplication), while 31 contain findings of two or more tools (inter-tool deduplication). The size of clusters varied between one and 408 findings, with a mean of seven findings. The golden DAST dataset consisted of ten annotated, intra-tool clusters. The number of findings per cluster varied between one and 25, averaging at three findings with a median value of one finding.

The construction of this dataset gave several vital insights for the remaining research. First, our source data contained noticeably more SAST findings than DAST findings. With 97.4% of all total findings being SAST findings, this is not explainable just by the higher number of selected SAST activities. One reason for this discrepancy could be the outlier activity HorusSec, which reported 927 findings and consequently contributed 68.6% of all SAST findings. Moreover, the activity is also responsible for the largest cluster in the golden dataset, consisting of 408 findings. This inequality between SAST and DAST dataset is continued in the length of semantically relevant information. On average, the

relevant information contains 169 more characters for DAST findings, resulting in a 55.9% higher verbosity than SAST findings. This can be explained by the consistency in the data scheme achieved by our selected DAST activities. The DAST reports consistently contained information on the title, description, and a solution for the finding, while the SAST reports mainly comprised a single feature describing the finding. Moreover, the content of the data fields was sufficiently verbose for our domain experts to understand the underlying problem reported by the DAST tool. Contrarily, the content of SAST reports was perceived as relatively brief, making it challenging to comprehend each finding just from the given features. This is problematic for our evaluation, as each finding is represented by the semantic content of its features. Hence, data with low semantic value limits the effectiveness of any applied technique, as it insufficiently represents the finding. However, insights provided by our domain experts showed that unique identifiers like the CVE-ID or CWE-ID provide additional guidance for identifying duplicate findings and can make up for the low semantic value of information.

Based on this golden dataset, the evaluation of our selected semantic similarity-based clustering techniques was performed.

Evaluation Planning

To evaluate our selected semantic similarity-based techniques *WordNet* [116], *LSI* [99], and *SBERT* [150], we compared their suggestion of semantically similar findings to the golden dataset developed in the last paragraph.

One requirement to identify similarity between findings for all techniques was the construction of a problem-specific *finding string*. As discussed in Subsection 4.4.1, some features describe the underlying problem of a finding better than others. Moreover, the construction of the golden dataset showed that even with an optimal selection of features, the finding string might be under-specified due to the data reported by the activity. To counteract this problem and follow the advice given by our domain experts, we make use of a finding's CVE-ID. These unique identifiers are assigned to most publicly known vulnerability and provide the capability to identify duplicates in the dataset based on simple text matching. Using this preliminary clustering mechanism, the *finding string* can be constructed from the data of multiple findings, allowing for a longer and more verbose representation. Consequently, we constructed two types of corpora for each dataset type (SAST and DAST). For the less verbose SAST findings, we constructed one corpus with solely the finding description as *finding string* and another corpus utilizing the preliminary clustering approach based on CVE-ID and concatenated the descriptions of all findings in one cluster as *finding string*. For the information-dense DAST findings, we also constructed one corpus with solely the finding description as *finding string*. As the DAST findings also consistently provided title, description, and solution texts, another corpus was constructed where the *finding string* was made of a concatenation of these fields. These *finding strings* represent the finding in the NLP-based similarity techniques to determine the similarity between two findings.

This similarity between two findings is expressed as a numeric score between 0 and 1 and returned by each of our evaluated techniques. With 1 representing the highest

possible similarity between two findings, a threshold defining the value, above which we consider two findings as duplicates, is necessary. We define this threshold as *similarity theshold*. Whenever the technique reported a similarity score between two findings higher than the *similarity theshold*, they were grouped into one cluster. This grouping introduces the special case of finding clusters that do not fulfill the transitive property. In practice, this transitive property is crucial. If any security finding addresses the same problem as two other findings, these findings consequently also address the same problem. Hence, all three must be in the same cluster, representing this exact problem. However, this property might be violated by applying the semantic similarity concept. Imagine a finding (*Finding 1*) being similar to two other findings (*Finding 2, 3*), implying that the similarity score between *Finding 1* and *Finding 2* as well as *Finding 1* and *Finding 3* is above the *similarity threshold*.

```
Finding 1 - [Finding 2, Finding 3]
Finding 2 - [Finding 1, Finding 4]
Finding 3 - [Finding 1]
Finding 4 - [Finding 2]
```

Consequently, these three findings would compose one cluster. However, the similarity is also calculated for all other finding strings. For *Finding 2*, this could result in the following cluster: Since *Finding 1* is similar to *Finding 2*, this also applies vice-versa. In addition, *Finding 2* might also be similar to *Finding 4* while having a similarity score lower than the threshold when compared with *Finding 3*. This would result in 2 different clusters that violate the transitive property of deduplication existing in practice. Due to the various terminologies and verbosity of reports employed by the security activities, this threat could affect the experiments. For the performance of the semantic similarity-based techniques, this situation would indicate that either the relation between, e.g., *Finding 2* and *Finding 4* was not found or the similarity score between, e.g., *Finding 1* and *Finding 4* was wrongly reported as higher than the threshold. To avoid these problems, we postprocess the results by applying the transitive property and summarizing all similar results in one cluster. This corrects the above-mentioned clusters to:

```
Finding 1 - [Finding 2, Finding 3, Finding 4]
Finding 2 - [Finding 1, Finding 3, Finding 4]
Finding 3 - [Finding 1, Finding 2, Finding 4]
Finding 4 - [Finding 1, Finding 2, Finding 3]
```

Since the selected techniques may vary in performance depending on the chosen threshold, we decided to perform experiments for *similarity threshold* $0.1 \leq$ and ≤ 0.95.

After the postprocessing, the results of each combination of technique, corpus, and similarity threshold are compared to the golden dataset. This comparison measures the *False Positives*, *True Positives*, and *False Negatives*. In this context, a *False Positive* is a cluster existing in the predicted dataset but not in the golden dataset. This is based on excess or missing findings in the cluster. Any cluster in the predicted dataset containing the same findings as the golden dataset is counted as *True Positive*. If a cluster in the golden dataset does not exist in the predicted dataset, it is counted as a *False Negative*. The

collection of these values enables the calculation of the performance metrics *precision*, *recall*, and *F-score*. In our experiment, the *precision* measures the ratio of correctly predicted clusters to the overall number of predictions.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

The *recall* measures the ratio between correctly predicted clusters and all clusters existing in the golden dataset.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegatives}$$

Finally, the *F-score* balances both proceeding measurements and represents an overall performance metric.

$$F - score = \frac{2 * TruePositive}{2 * TruePositive + FalsePositives + FalseNegatives}$$

With these three measurements, we can quantify the performance of all techniques. A high *precision* indicates that a technique produces fewer False Positives, while a high *recall* informs that most of the clusters that should have been found were also found. Finally, the harmonized *F-score* drives our conclusion on the performance of all techniques.

On top of this first level of quantitative evaluation, the incorrectly predicted data (False Positives, False Negatives) was given to the domain experts for feedback. Since they constructed the golden dataset, their opinion and perception of the incorrect processing provide valuable insights for reasons or improvement. To ensure the same level of information for the domain experts and the NLP techniques, each finding was represented solely by its finding string. Each incorrect cluster was provided to the domain experts, and possible reasons for the poor performance of the duplicate identification were gathered from them. This resulted in a dataset where each incorrect cluster was assigned at least one reason.

Evaluation Results

The quantitative evaluation provided us with the precision, recall, and F-score for each combination of NLP technique, corpus, and similarity threshold.

Figure 4.3 presents the F-score of all techniques for both SAST corpora, in regards to the selected similarity threshold. The figure indicates that with increasing similarity threshold, the performance of all techniques, represented by the F-score, improves. Most combinations reach their peak at a similarity threshold of 0.9. The only outliers are the knowledge graph-based semantic similarity clustering techniques, which spike around 0.3 and decrease at higher thresholds. The best performance for the standard corpus is reached with Latent Semantic Indexing, with an F-score of 0.739. The best overall performance is shown by Latent Semantic Indexing with the preliminary CVE-ID-based clustering, with an F-score of 0.816.

In Figure 4.4, the same information is presented for both DAST datasets. Similar to the SAST results, the techniques' performance with the DAST corpora initially improves

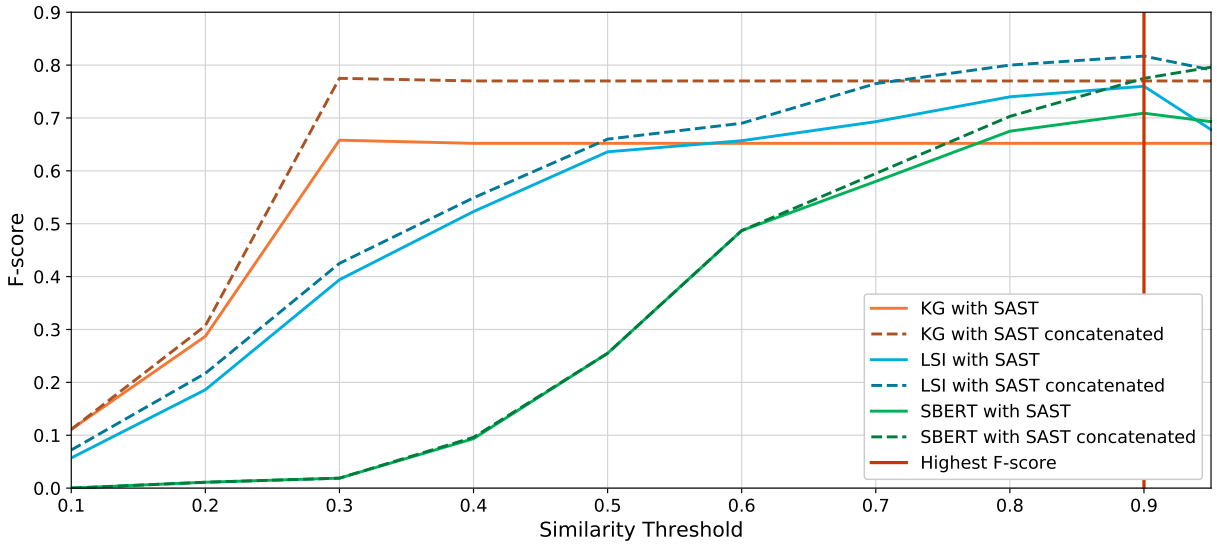

Figure 4.3: Results of KG-Based, LSI, and SBERT Clustering of the SAST Dataset

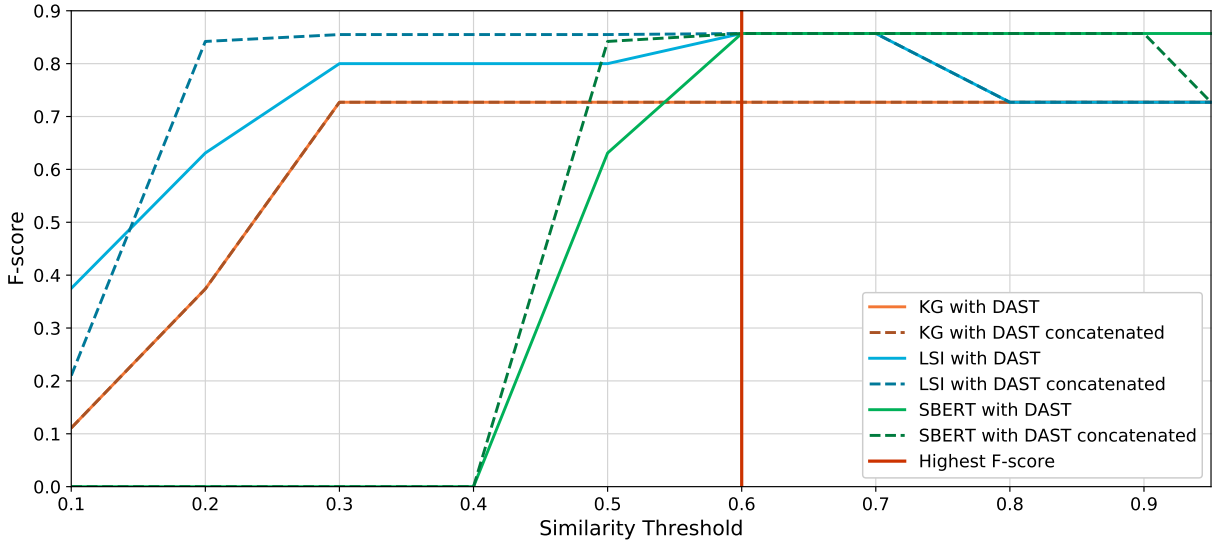

Figure 4.4: Results of KG-Based, LSI, and SBERT Clustering of the DAST Dataset

with increasing threshold. However, most techniques reached a plateau or even decreased in performance after reaching their peak at 0.6. At this threshold, LSI and SBERT peaked at the same F-score of 0.857 for both corpora. Moreover, neither corpus construction approach impacted the performance of KG-based clustering, resulting in the same graph for both corpora.

Corpus	Technique	Recall	Precision	F-score
SAST	SBERT	0.825	0.621	0.709
SAST	**LSI**	0.842	0.658	**0.739**
SAST	KG	0.809	0.556	0.659
SAST concatenated	SBERT	0.923	0.701	0.797
SAST concatenated	**LSI**	0.918	0.734	**0.816**
SAST concatenated	KG	0.913	0.676	0.777
DAST	**SBERT**	0.900	0.818	**0.857**
DAST	**LSI**	0.900	0.818	**0.857**
DAST	KG	0.800	0.667	0.727
DAST concatenated	**SBERT**	0.900	0.818	**0.857**
DAST concatenated	**LSI**	0.900	0.818	**0.857**
DAST concatenated	KG	0.800	0.667	0.727

Table 4.6: Performance Indicators of KG-Based, LSI, and SBERT Clustering

Table 4.6 lists the recall and precision for the highest F-score for each combination of technique and corpus. The best-performing technique is marked for each corpus, showing that LSI and SBERT performed best for the DAST corpora.

During the qualitative evaluation, the datasets with the highest F-score were given to the domain experts for review. This included the dataset resulting from applying LSI to the preliminary clustered SAST dataset and from applying SBERT to the concatenated DAST datatset. Since our quantitative evaluation showed no difference between LSI and SBERT for both DAST corpora, we selected the knowledge denser concatenated corpus with SBERT, which was disregarded for the review of the SAST dataset. The domain experts solely investigated the 72 incorrect SAST predictions and two incorrect DAST predictions. Only one reason for both incorrect predictions was stated for DAST, while nine different explanations have been identified for the incorrect SAST predictions. In summary, 114 reasons have been assigned, so each reason was applied to multiple flawed clusters, and sometimes multiple reasons have been assigned to the same cluster. Table 4.7 lists all given explanations, sorted by affected dataset and frequency.

ID	Dataset	Frequency	Explanation
1	DAST	2	In the context of the product, this result can only be identified by somebody knowing the context of the application.
2	SAST	1	Human annotation error and the suggested clustering by the algorithm is correct.
3	SAST	3	One tool addresses the issue of using an *eval* function, while the other one has the problem of user controlled values in it. However, it would not be considered as a major false positive.
4	SAST	3	The tool describes the finding precisely according to the location of occurrence. Hence the finding text is over-specified.
5	SAST	5	Different tools use a different phrasing to explain the same issue.
6	SAST	5	Additional review necessary due to an unknown reason for the decision.
7	SAST	19	Some tools provide more and some tools provide less text in their description, which reduces the impact of actual relevant features.
8	SAST	39	The sub-optimally constructed feature string could be the reason for the incorrect clustering.
9	SAST	39	The tools sometimes provide no description of the finding. Hence, the features could only rely on the title.

Table 4.7: Reasons for the Incorrect Clustering, given by the Domain Experts

Evaluation Discussion and Conclusions

The ultimate goal of this comparision between different NLP-based techniques was the identification of a clustering technique for the deduplication of industrial security findings. Considering the SAST dataset, we see that Latent Semantic Indexing combined with a preliminary clustering of security findings based on the CVE-ID yields the best performance in our experiment. Even though SBERT also showed a promising performance, considering its F-score of 0.739, the lower precision compared to LSI makes it less applicable for industrial projects. In practice, a False Positive leads to findings being incorrectly assigned to the same cluster. If practitioners believe the cluster-specific problem has been solved, an incorrectly clustered finding might be missed or actively classified as False Positive by the activity. On the other hand, a False Negative only results in more work for practitioners as further clustering would have been possible. Therefore, the first case might put the security of a product at risk, while the second solely increases the workload. As seen in Figure 4.3, the preliminary clustering supported all techniques in providing better results

for the SAST dataset, which leads to the conclusion that this is an essential part of the deduplication process.

This trend was not continued for the DAST corpora. While the inclusion of more semantically relevant features into the finding string provided better results for most techniques at lower thresholds, it had no impact when looking at the best performance of each technique. As depicted in Table 4.6, each technique achieved the same recall, precision, and F-score for both corpora. Moreover, SBERT and LSI showed the same performance under the optimal threshold. However, for similarity thresholds ≥ 0.6, SBERT shows a better or at least equal performance in comparison to LSI. Since a higher similarity threshold reduces the number of potential False Positives in practice, we consider SBERT to be preferable over LSI for this use case.

Even though the knowledge graph-based similarity clustering technique showed promising results for the concatenated SAST corpus, its performance compared to the other two techniques has proven to be worse for all corpora. Regardless of the corpus, the knowledge graph-based technique plateaus at a similarity threshold ≥ 0.3 with an F-score lower than both other techniques. Consequently, the knowledge graph-based similarity clustering is irrelevant for our use case.

Due to the qualitative evaluation, we identified the key challenges for clustering security findings with NLP-techniques: the finding string. Since this string is constructed from the available features each security finding contains, its quality is constrained by the content of the source data. This problem is also indicated in the explanations provided by our domain experts, as the two most mentioned reasons (reasons 8 and 9) refer to a sub-optimally constructed finding string and a missing description of the security finding. The finding string of any finding that just gives a brief title will consequently be represented by a finding string of insufficient semantic context for similarity identification. This challenges both algorithms and practitioners, as indicated by reason 2. However, not only the existence of data but also its length bears a challenge. Since the similarity between two findings depends on the similarity of features within the finding string, longer descriptions will not always yield better results. As described in reason 7 by our domain experts, a finding elaborating on the general category of problems it belongs to might introduce terms that are not relevant to the actual finding problem. Therefore, it reduces the accuracy of similarity-based clustering with semantically underspecified findings since the relevant terms are less frequent.

Consequently, we conclude that SAST security findings are best clustered by LSI and DAST findings by SBERT. Moreover, the different F-scores between the DAST dataset and the less verbose SAST dataset indicate that a better performance can be achieved by improving the semantic content of each finding string. A preliminary clustering based on CVE-ID has proven promising, while a challenge for further research is constructing a semantically rich and precise finding string.

4.4.3 Semantic-Similarity based Clustering of Industrial Security Findings

In order to eliminate duplicate security findings in industrial software development projects, the knowledge acquired by our experiments must be transformed and adapted to industrial practice. Consequently, we investigated how a process for semantic similarity-based clustering in the industrial software engineering process could look based on previous learnings.

Analysis of Previous Results

The first step towards a process clustering duplicate security findings is to set the results of the last section into the context of industrial practice. This ensures that our process is designed according to evidence-based research while avoiding negative impact on the process in the industrial practice.

One of the conclusions of our previous research is the importance of the finding string that represents each finding for the semantic similarity-based technique. For this finding string, we picked the relevant semantic content from each report by hand and developed a parser for each activity report to derive the selected features. In practice, it is not feasible to manually select features each time a new report is given to the deduplication. Instead, an abstraction layer is needed so that the creation of the finding string is decoupled from the identification of semantically relevant finding features.

Another conclusion from the previous research is the impact of preliminary clustering based on the CVE-ID. Especially for the SAST dataset, this has proven to positively affect the clustering results. However, relying solely on the CVE-ID to construct a semantically rich finding string is insufficient. CVE-IDs are only consistently provided by security activities that check for publicly known vulnerabilities. Insecure coding patterns, violations of security principles, or missing system hardening rarely have such an identifier assigned. Hence, only a fraction of all security findings contain a unique identifier in practice. Furthermore, the same finding might be referenced by different unique ids depending on the database (”NSWG-ECO-428” from nodejs/security-wg vs. ”GHSA-rvg8-pwq2-xj7q” from Github Advisory Database). Consequently, the varying sources and structures for unique identifiers further complicate a correct deduplication. Even if the CVE-ID is given, the effect of preliminary clustering on the finding strings of those findings containing a CVE-ID is limited. Since most activities with CVE-IDs rely on the same semantically rich data provided by vulnerability databases, the content of their reported security findings is already semantically rich or an exact copy of the vulnerability database entry. Hence, using only the CVE-ID to ensure a context-rich finding string is insufficient for an optimal construction approach.

The previous research instead emphasizes the importance of a well-designed finding string construction approach. It mentions multiple pitfalls, including the length of the finding string or the challenge of source data with low semantic context. These challenges persist when applied to industrial practice and must be considered during finding string construction.

However, even with a semantically rich finding string, an incorrect clustering of security

findings might not be avoidable. While in the previous research, it was sufficient to identify faulty clusters and investigate reasons, these flaws impact later steps of the findings management and, therefore, necessitate a correction in industrial practice. As stated in the reasons for incorrect clustering given by the experts, knowledge about the context of a security finding might be essential to understand and correctly identify duplicates. Hence, even an optimal finding string used by an optimized NLP model might not be able to identify all duplicate relations correctly due to the missing context. Moreover, we saw that even the manually deduplicated security findings contained errors in the clustering that were only identified after the domain experts were asked to explain the incorrect clustering. Minding that the automated approach using semantic similarity-based techniques supposedly yields a comparably lower precision and recall, the need for a cluster correction step is further reinforced. However, the necessary context information to correct clusters can only given by project-aware human entities after an analysis. An example of the importance of context information would be a security finding identified by *Dynamic Interface, API and Blackbox Testing* detecting an open port in an application's attack surface. The underlying problem of this open port could be that software running on the system image of the application is not needed. This lack of hardening measures would also be detected by *Configuration and Hardening Checks* but from the perspective of system hardening instead of port scanning. Based on the different testing perspectives, the respective security finding would either contain the context of open network ports or missing system hardening measures. The context information that both findings are duplicates can only be provided by a context-aware human with knowledge about the underlying problem and the relation of both findings. Consequently, any process that aims to cluster duplicate findings has to offer the capability for correcting operations by input from practitioners.

The fundamental result of the previous research is the applicability of semantic similarity-based clustering approaches for identifying duplicate security findings. The results suggest that security findings reported by security activities testing the software in a static state are best performed by applying Latent Semantic Indexing. At the same time, SBERT showed the best results for dynamically acquired security findings. However, this distinct separation between datasets is not practicable in reality. Many security findings identified by dynamic scanning or testing techniques have their root cause in aspects of the software that are also covered by static methods. An example would be a publicly known vulnerability in a web framework identified by static *3rd Party Component Vulnerability Scanning* as well as *Dynamic Interface, API and Blackbox Testing*. Therefore, duplicate security findings can also occur across static and dynamic datasets, requiring a unified dataset for industrial practice.

Security Findings Clustering Process

Our process for clustering duplicate security findings is based on previous research in the context of industrial practice. This process depends on an arbitrary list of security reports as an input and provides clusters of security findings, each representing one unique problem as output.

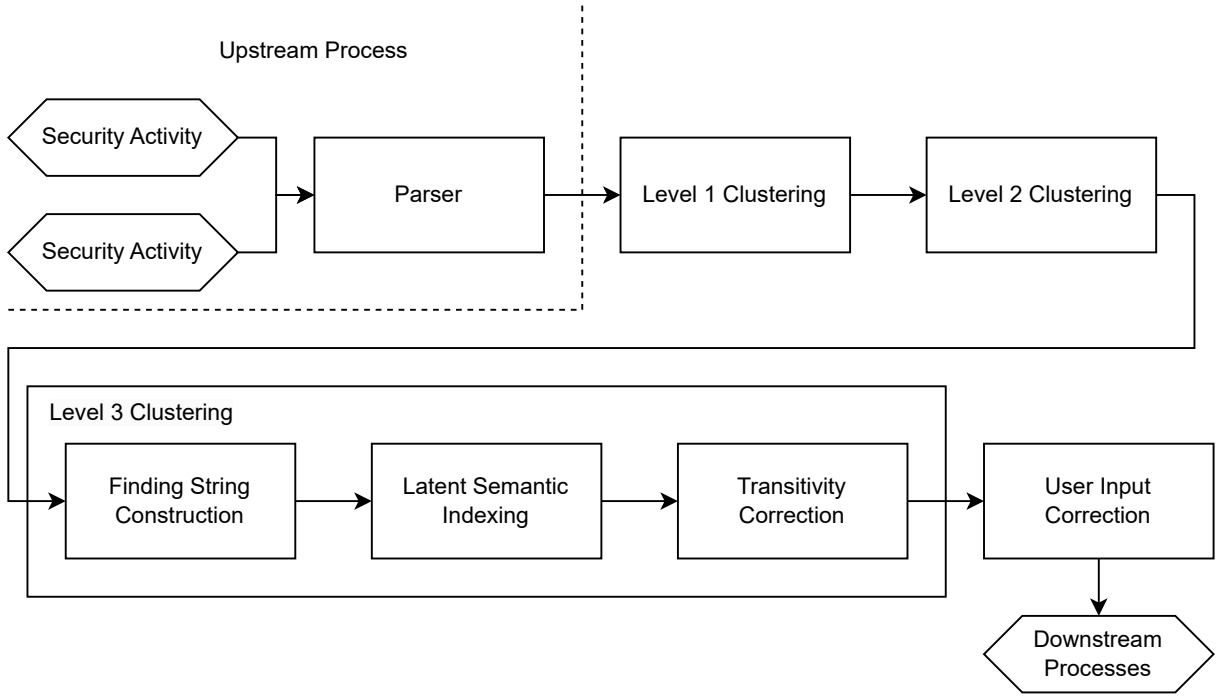

Figure 4.5: Clustering Process for Industrial Security Findings

The first stage of the process is a parser that unifies the format of all security findings. This stage solves the challenge of identifying the data fields of each finding that are semantically relevant and represent the underlying finding problem. Since each security activity provides a different data format, report structure, and terminology in the field naming, the parser must apply different strategies to transform the data into a data model that is understandable for later stages. This stage creates a list of security findings following a common data format.

In our second stage, the clustering process is started. Since security findings management is a continuous process with new reports arising constantly, the source dataset might contain multiple reports from the same source at different times. As the same activity might report security findings with the same data in several reports, the most straightforward approach is an *intra-activity* clustering. This *intra-activity* clustering considers two findings as duplicates if they contain the exact same information, excluding the "Timestamp", activity "Version" and "Location" fields. Otherwise, these fields would falsify the duplicate identification since the same problem is still addressed, regardless of the activity's version, the problem's location, or when it was found. Since this approach works on the exact textual data of each finding, False Positives are almost non-existent. We define this type of clustering as *Level 1 Clustering*. This stage results in clusters of security findings.

This clustering is further refined in the next stage. Based on the clusters formed by the *Level 1 Clustering*, we follow the recommendations of the previous research and cluster all findings with the same CVE-ID. Even though other unique identifiers exist, the CVE-ID

is most widespread in practice. Consequently, this stage groups all findings cluster from the last stage that contain the same CVE-ID. Since the previous stage ensured that all clusters contained the same information for each finding, we can assume that each cluster has one CVE-ID at max. With this stage, we can realize *inter-activity* deduplication, as CVE-IDs are assigned by multiple activities. During this stage, any False Positive in the clustering is caused by an incorrectly set CVE-ID by the source activity. We define this type of clustering as *Level 2 Clustering*. Similar to the last stage, this stage again results in clusters of security findings, where each finding of a cluster contains the same CVE-ID.

Based on these preliminary groups of security findings, the semantic similarity-based clustering is applied. This fourth stage is three-fold, including creating the finding string, applying the NLP-technique, and checking for the transitive property of clusters. With the construction of the finding string, the goal is to create a semantically rich representation of the security finding. Towards this goal, all features/data fields of every finding in one group from the *Level 2 Clustering* are removed except for "Title", "Description", "Countermeasures", "Finding Category", "Prerequisites", "Impact", and "Background". The fields containing unique identifiers are removed on purpose, as an equal identifier indicates a duplicate finding, while a similar identifier gives no insights at all. The values from all findings in one cluster for each selected feature are deduplicated and concatenated to one text afterward. This step removes, e.g., duplicate titles or descriptions to avoid adding repetitive information that supplies no additional semantic value. In contrast to the recommendation of the initial research, this step does not further enrich each finding string by the knowledge of vulnerability databases, as any unique identifier indicates that the source activity already considered the knowledge of the respective database. Hence, adding the information given by the vulnerability database does not improve the semantic value of the finding string. Finally, the stopwords of each string are removed to eliminate semantically irrelevant information from the string further. This procedure is applied to every cluster of findings, resulting in one finding string per cluster.

During the second part of the stage, these finding strings are used to identify duplicate clusters given by the *Level 2 Clustering*. Since all findings should be clustered and deduplicated by the same process, only one technique can be applied. We selected LSI as the technique to cluster similar finding strings. There are two reasons for this choice. First, LSI showed better results for the SAST dataset and performed equally for the DAST dataset when compared to SBERT. Second, industrial software development tools employ a wide variety of static security testing activities, while the amount of dynamic tools is limited [11]. Hence, better performance on the SAST dataset is preferable for industrial practice. We define this type of clustering as *Level 3 Clustering*. This second part results in a list of similarity values between all clusters.

The last part of the fourth stage ensures the transitive property of the clustering approach and constructs the respective clusters. As recommended by the previous research, we consider two clusters as duplicates with a similarity ≥ 0.9. Identically to the post-processing of the initial process, the transitive property is ensured by merging all clusters where the similarity between two clusters is ≥ 0.9. This step finalizes the fourth stage and results in a list of clusters that contain the initial security findings.

The fifth and last stage realizes the experience-based correction of the clustering results by user input. This external input has two potential options. If it identifies a False Positive, two findings are suggested as duplicates but are distinct findings in practice. This results in the removal of one finding from a cluster. If the external input identifies a False Negative, two findings are presented as distinct but are duplicates. In this case, the two clusters are merged. Each cluster identified by the *Level 3 Clustering* is investigated for applicable user input, and potential actions based on the user input are executed. If conflicting user input is present, e.g., informing that two findings are simultaneously duplicates and distinct, the distinction is prioritized to avoid False Positives. This stage provides the same results as the fourth stage but potentially changes the amount or size of clusters.

The entire process for clustering industrial security findings is depicted in Figure 4.5. In summary, it constructs a list of clusters, each containing a list of security findings considered problem-based duplicates.

4.4.4 Aggregation of Clusters

To deduplicate industrial security findings, the identification of duplicates is only the first part of the process. Additionally, it is crucial to eliminate duplicate findings and solely persist unique and distinct findings. Our security findings clustering process provides several groups of findings, where each group addresses one security problem. The information contained by each cluster must be aggregated into one security finding that represents all findings of that cluster simultaneously.

An intuitive representation of all findings in a cluster is the finding string belonging to this cluster. However, the string neither fits the format of a security finding nor does it describe the finding in a way that subsequent processes or human actors can understand it. Therefore, the finding string is not suitable for the aggregation of clusters. The main requirement for the aggregation strategy is that subsequent processes can be performed with the information contained in each security finding. Therefore, each finding should follow the existing data model for security findings. Moreover, the data of each security finding should be persisted to avoid data loss. As there are, e.g., also multiple approaches on how to mitigate security findings, this information must be maintained.

Therefore, the aggregation of clusters must create a common representation of the cluster while persisting the information of each cluster element. Since we can assume that all security findings follow a common data model due to the preceding parsing operation, our process iterates through every data field in Table 4.3. The content of each data field from all findings of one cluster is collected, duplicate entries are eliminated, and the remaining values are stored in a list. This list represents the content of the respective data field for the aggregated security finding. After aggregating each data field, the newly constructed security finding represents the entire cluster. Hence, the cluster aggregation creates a new security finding for each cluster.

Integration Process

In contrast to the preceding processes to improve the quality of security findings data,

this process has no requirements for its integration into projects. It solely relies on the collection and parsing operation described in Section 4.3.

4.5 Finding Information Enrichment

The rising number of automated security checks supported by security tools reduces the barrier of integrating new security activities into each project. However, the quality of each security finding varies, depending on the characteristics of the activity that reports it. Consequently, no reliable level of information can be expected across all security activities. However, a minimum level of information is necessary to manage security findings. Moreover, the subsequent stages of the security findings management might benefit from additional information about each finding not provided by the activity reporting the finding. Therefore, the enrichment of information comprised by each security finding is essential for managing security findings in industry.

In this section, we present the information enrichment process for security findings. We distinguish between preliminary enrichment, ensuring a minimum level of information each finding comprises, and internal/external information enrichment, refining each finding with additional insights.

4.5.1 Preliminary Enrichment

The first type of enrichment addresses the problem that specific security findings lack fundamental information necessary for subsequent management actions. Subsection 4.3.3 identified the necessity to ensure each finding contains the fields ”Title”, ”Location”, ”Timestamp”, ”Source” and ”Version”. We define this process of supplying additional information to security findings during parsing as *Preliminary Enrichment*.

Consequently, the preliminary enrichment is part of the parsing operations and supplies each finding with data available during the parsing process. Depending on which fields are missing, the remaining ones are derived from context information, given data fields, or circumstances during the operation. The collection mechanism can supply the ”Source” and ”Version” fields since the source and the version of the activity are provided during the collection of security reports. The ”Timestamp” field is trivial to create, as it can either be supplied during the collection mechanism or set to the current date and time of the enrichment operation. Even though the latter one falsifies the timestamp, it represents the closest timestamp possible if no other information is provided. Finally, ”Title” and ”Location” represent the only fields that can not be supplied by context information or environmental circumstances but can be supplied solely from other data fields. These two fields map to the original requirement for the enrichment of security findings, requesting that each finding contains at least the impact of a finding on the software and its nature. A missing title field can only be replaced by information that ”Describes the finding in one sentence”. Consequently, the first 100 characters of one of the following fields are re-used, ordered according to preference:

- Description
- Background
- ID
- CVE-ID
- GHSA

If no of the fields mentioned above are available, the supplementary text "Security Finding by " concatenated with the "Source" is used. The location is more complex to supply, as no other field indicates where the security finding is located in the software. Therefore, this field is directly filled with the name of the project/product to suggest that the finding is located somewhere in the software. This approach is used for all types of security activities equally, implying that no additional, project-specific integration effort has to occur.

4.5.2 External Enrichment

After the security findings have been clustered and aggregated, the content of each finding can be enriched with additional information. In contrast to the preliminary cluster, this step does not focus on achieving a certain base level of information in each finding but on providing the opportunity to refine the existing information of each security finding. This enrichment utilizes data external to the security findings management process, as the information is not provided by any of the activities as a report. Therefore, we consider this type of enrichment as *External Enrichment*.

As the related work section identifies, enriching information by knowledge contained in vulnerability databases is a common approach. However, not every project can benefit from this type of enrichment. The reasoning against a general recommendation for enriching all security information with data from vulnerability databases is similar to the explanation in Section 4.4. Adding more details to each security finding does not inherently result in more efficient or effective security findings management. Instead, more information might confuse entities interacting with the data or even introduce semantic duplicates of what each finding already contains.

Consequently, no general recommendations or sources are given for the external enrichment of security findings. This implies that the relevance and content of the external enrichment are solely determined by the project and organization following the security findings management methodology. Hence, this part of the security findings management process allows actors to enrich their findings on demand. This approach further separates the information lifecycle of the security findings management from the external data, as the most recent version of any external information is continuously gathered. Hence, no maintenance operations for the external input are necessary during the security findings management and are instead assumed to be performed by the parties responsible for the external data. The enrichment strategy follows an if-then structure. If a security finding fulfills certain criteria, an enrichment operation takes place during which new data might be added to the security finding. This is formalized in pseudocode as follows:

```
if <finding fulfills property x>
    new_information = run <enrichment operation>
    finding = finding + new_information
endif
return finding
```

The enrichment operation to collect additional knowledge is formally not defined closer, as it highly depends on the use case and source of the information.

One example of a possible enrichment operation could be the information provided by the GitHub Advisory Database[4]. This database contains known vulnerabilities and malware from various sources. In particular, it includes community contributions, often not found in another database. Hence, an enrichment based on this source can be beneficial for projects with components of less public interest. In this case, any security finding with a GHSA identifier would be considered for an enrichment operation. During the procedure, the GitHub Advisory Database would be queried using the identifier and, e.g., the "Details" field of the entry added to the description of the finding. Another example more closely coupled to the organization would be domain-specific solutions for specific findings. Software components like Identity and Access Management solutions are often in-house developments in industry, resulting in an under-representation in public vulnerability databases. However, the component supplier might be able to provide deeper insights into the mitigation of findings. In this case, any finding in the particular component would be considered for enrichment. During the operation, an operations manual might be queried for the keywords, and a respective link to this manual could be added as a potential solution, concluding the enrichment of the security finding.

Integration Process

To integrate external enrichment into projects, each enrichment operation and its respective condition for execution must be formalized. As this might further include ensuring network or access requirements are met, the integration effort must be planned for.

4.5.3 Internal Enrichment

Another strategy for enriching security findings with additional information relies on information that does not originate from outside the security findings management process but derives information from existing data. New knowledge can often be obtained from combining the available security activity data with context information, providing new perspectives for a particular security finding. This type of enrichment considers information of a security finding after the external enrichment and gives additional insights based on this knowledge. We define this type of enrichment as *Internal Enrichment*.

In contrast to the external enrichment, the logical rules to derive information originate from the process and are maintained during the process. Hence, no external platform contains knowledge that is added to the security findings, but the existing data is leveraged

[4]https://github.com/advisories

to gain further insights. Moreover, this affects the information lifecycle as no external party maintains the enrichment data, but a periodic review of all internal enrichment strategies is necessary to ensure a reliable, correct, and valuable enrichment.

Like external enrichment, internal enrichment is solely a capability of the data quality improvement process, while the project team must provide the actual enrichment strategies. With each project having different security activities in use and a distinct development environment, a standard set of applicable enrichment rules cannot yield a universal positive effect on the management process. Especially as their maintenance has to be performed as part of the security findings management process, pre-defined rules for how to enrich security findings must be decided on and understood in each project. The methodology solely proposes the format for defining the internal enrichment rules. Similar to the external enrichment, they are based on an if-then structure, where security findings are enriched with data, if they fulfill a specific criteria.

An example of an internal enrichment rule would be adding context information if findings originate from a specific security activity. With the rise of modern software development, the automation of security checks is becoming extremely popular. As part of this trend, the complexity of integrating security checks is reduced or guided by best practices, enabling non-security experts to interact with these checks. However, interpreting the respective results in the context of software development is still challenging. Hence, an internal enrichment could inform developers about the implications of certain security findings that are obvious to domain experts but unclear to other groups. One example would be that all findings from an activity checking third-party libraries can be found either in the dependency manager or may also be sub-components of installed dependencies. While this is obvious to domain experts, less experienced teams might be challenged by findings not explicitly listed in the dependency manager. The respective rule in pseudocode could look like the following:

```
if finding.source in [Activity A, Activity C, Activity X]
    finding.location = finding.location +
        "Guidance: Problem might be located in sub-dependencies as well"
endif
return finding
```

However, this information must be maintained by the project. If a new security activity is employed, where this enrichment should be applied as well, the rule's condition must be adapted. As no external actor manages the rule, this is the project team's responsibility.

Integration Process

Integrating the internal enrichment into a project is comparable to the external enrichment. As each rule must be defined upfront, the definition and formalization process must be considered for integration. As the internal enrichment relies entirely on the data given by previous stages, no additional dependencies must be considered for integration.

4.6 Implementation

The first step to ensure the relevance of our solution approaches to industrial practice and evaluate them is their integration into the platform for managing security findings. This ensures that the current theoretical concepts can be assessed in practice. Therefore, we created separate rules and belief classes for each solution approach and integrated them into the semantic knowledge base presented in Section 3.5. This section presents the instantiation of each solution approach and its integration into the given knowledge base framework. The overall process is depicted in Figure 4.6.

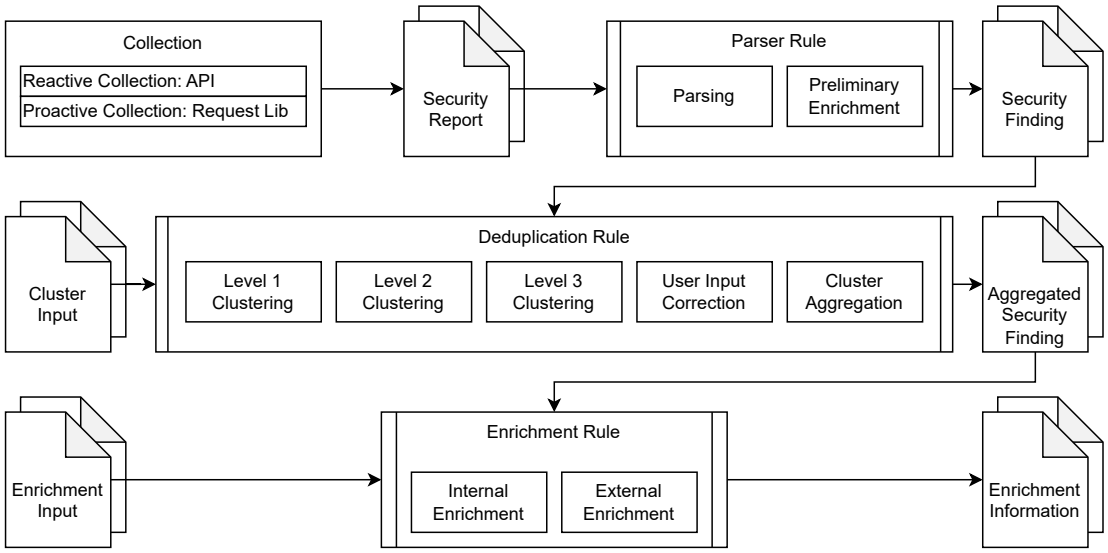

Figure 4.6: Diagram of the Data Quality Improvement Process

4.6.1 Implementation of the Collection Approach

The first solution approach was the collection of security reports from all activities. In contrast to all other approaches described in this chapter, the collection process extends the knowledge base framework further instead of being exclusively represented by rules and belief classes. The implementation of the collection approach has to address two features: the reactive and the proactive collection of security reports. For the reactive collection, we extend the existing knowledge base with a RESTful API that allows the upload of security reports. The proactive collection is realized by an interface that is able to send requests to external parties, like HTTP requests to other APIs. Both functions are implemented as part of the ”KB Interface”, found in the top left corner of Figure 3.6.

The reactive collection is implemented by utilizing the existing Flask implementation of the Knowledge Base I/O and extending it by the resource */data*. To upload security reports, this API supports an HTTP POST request. Each POST request must be sent

with a JSON payload containing the information about the security report and its context. Table 4.8 lists all keys that can or must be used when uploading a security report with their respective explanation.

Key	Type	Required	Description
Project	String	Compulsory	Name of the project this report belongs to
Data String	String	Optional	The report in its textual representation
Data	File	Optional	The report in its representation as a file
Time	String	Optional	Datetime of when the report was created
Source	String	Compulsory	Activity that created this report
Version	String	Compulsory	Version of the activity that created this report

Table 4.8: JSON Parameters in a Security Report Upload Operation

The keys ”Data” and ”Data String” represent a special case, as precisely one of them has to be used. Any security report delivered to the semantic knowledge base must utilize this API to upload the information.

However, not every activity can actively send security reports to our platform. Therefore, a proactive collection is implemented that requests information from the respective sources. To implement this collection mechanism, the existing KB Interface is extended to include HTTP requests that the interface can send. A separate set of operations must be defined for each source that collects the compulsory information, listed in Table 4.8. This set of operations is implemented in Python and represents the collection function for the source. This customization is crucial as some sources might host the most recent security report at one API resource and the activity version at another. There are two approaches to define when this collection function for a source is executed. If a web hook or a similar notification technology is possible, the process is executed whenever the knowledge base is notified about a new report. Otherwise, the function is run regularly, e.g., daily or weekly. Since Python is employed as a programming language for the collection, other protocols than HTTP for querying information are also possible.

Regardless of how the security report reaches the knowledge base, it must be stored as belief. Therefore, we define the belief class ”Security Report” by the following structure:

```
"Class Name": "Security Report",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Explicit",

"Source": <>,
"Version": <>,
"Timestamp": "YYYY-MM-DD HH:MM+Tz",
"Content": "Security Report in textual form"
```

In addition to the meta-structural information necessary in each belief class, it contains all collected information, excluding project identification. Since each project has a separate

instance of the knowledge base, the project is provided inherently by the data storage.

As security reports contain confidential information about software, the malicious upload of reports might infringe the project's success. Therefore, an authentication and authorization concept is in place that projects the knowledge base from malicious actors. This is currently irrelevant to the methodology, so it is not presented in more detail.

4.6.2 Implementation of the Parsing Approach

In the second solution approach, the uploaded reports are parsed to a common data model. This is implemented by one inference rule, defining the parsing operation, and another belief class representing security findings.

The attributes of the belief class are directly derived from the data model described in Subsection 4.3.2. In addition to the meta-structural information necessary in each belief class, it contains all data fields listed in Table 4.3. Since this class is derived from the security report by a parsing operation, the belief type is "Derived" in this case. This results in the following belief class with the placeholder "<Data fields of the data model>" being represented by the 22 data fields of the data model.

```
"Class Name": "Security Finding",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Derived",

<Data fields of the data model>
```

The parsing operation itself is implemented by the parsing rule with the 4-tuple:

```
{"Trigger": "Security Report", "Code": "<Reference to the File>",
"Output": "Security Finding", "Name": "Parsing"}
```

This rule is triggered by an instance of the security report class and creates instances of the above-defined belief class for security findings. Since each source requires a different parsing approach, the Python code is structured into one parsing approach per combination of source and version, resulting in the format for the code file.

```
if security_report.Source = "A" and security_report.Version < "1":
    <Parse reports from source A with a version smaller 1>
elif security_report.Source = "A" and security_report.Version >= "1":
    <Parse reports from source A with a version larger or equal 1>
elif security_report.Source = "B" and security_report.Version >= "0":
    <Parse reports from source B with a version larger or equal 0>
<Preliminary Enrichment of Security Findings>
```

A pseudocode example for an actual parsing operation is given in Subsection 4.3.3. The preliminary enrichment ensures a certain baseline of information is held by every security finding. The resulting list of security findings is afterward enriched as described in

Subsection 4.5.1, ensuring that each finding contains at least ”Title”, ”Location”, ”Timestamp”, ”Source” and ”Version”. The actual data model is communicated alongside with the implementation to ensure a common understanding of the data fields.

4.6.3 Implementation of the Clustering and Aggregation Approach

To eliminate duplicates between security findings, the third solution approach clusters and aggregates the security findings previously parsed. This is implemented by two belief classes and one inference rule.

The first belief class represents a clustered and aggregated security finding, summarizing all information the comprised security findings contain. Therefore, the belief class for an ”Aggregated Security Finding” consists of the meta-structural information necessary in each belief class and a list of entries for each data field of the data model listed in Table 4.3.

```
"Class Name": "Aggregated Security Finding",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Derived",

"Location List": [<>],
"Title List": [<>],
"Description List": [<>],
...
```

Each non-meta attribute of this class has a list as data type.

In addition to the belief class storing the aggregated security finding, practitioners might provide input to correct or refine the clustering results. Therefore, we introduce a second belief class dealing with this type of user input. Except for the meta-structural information, the belief class for ”Cluster Input” contains information about the user providing the input and the consequence of the input. Since this represents knowledge directly originating from outside the knowledge base, it has an explicit belief type.

```
"Class Name": "Cluster Input",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Explicit",

"User": <User Email>,
"Action": <Duplicate/Distinct>,
"Rule": <Content Hash A and Content Hash B>
```

The ”User” field contains a unique identifier of the user, which utilizes the user’s email address. The ”Action” informs whether the user identified a False Positive or False Negative. The ”Rule” attribute contains the unique content hashes of both security findings

that should be corrected. Since this information originates from outside the knowledge base, the resource */userinput* extends the RESTful API of the collection approach. This allows upload, modify, or delete correction input from users.

The clustering and its subsequent aggregation is implemented by an inference rule with the following 4-tuple:

```
{"Trigger": ["Security Finding", "Cluster Input"],
"Code": "<Reference to the File>",
"Output": "Aggregated Security Finding", "Name": "Deduplication"}
```

It is triggered by an instance of the security finding belief class and returns instances of the aggregated security findings. Moreover, it can also be triggered by an instance of the cluster input class, as a proposed correction by the user might introduce inconsistencies that can only be remediated by correcting the clusters. The inference code clusters the findings given as triggers with all other findings existing in the knowledge base by the three levels described in Subsection4.4.3. First, all findings with the same content, excluding the timestamp, version of the activity, and location, are clustered by utilizing a content hash. For each cluster, all other findings with the same CVE-ID are queried and summarized. Finally, the finding strings are constructed, and the LSI algorithm is applied using the *gensim* library for Python, and all clusters with a similarity score of ≥ 0.9 are merged. As part of this third clustering level, the transitive property between cluster entries is ensured. For every finding in each cluster, the content hash is used to search for user corrections. These corrections are afterward applied by either merging or splitting clusters accordingly. Finally, each cluster is aggregated by listing all unique entries of each data field in the respective list attribute of the belief class.

4.6.4 Implementation of the Enrichment Approach

The last solution approach is the enrichment of aggregated findings. To implement this solution approach, the semantic knowledge base's limitations must be considered. Our solution approach relies on the assumption that internal and external enrichment is project-dependent and will consequently be defined during the ongoing project. Therefore, new strategies to enrich the information might be added continuously. Since these strategies can be arbitrarily complex, using logical inference rules seems obvious. However, changes to inference rules are not as easily maintained as belief changes. The maintenance effort is substantial as changes to an inference rule invalidate all elements created from it and potentially create new belief instances. Hence, we implement this solution approach with one inference rule containing the actual inference and two belief classes. One belief class contains the resulting additional information for security findings, while the other represents the knowledge required during the enrichment process. Even though this still necessitates changes to the inference rule, detaching the enrichment information from the actual rule reduces the maintenance effort in case of changes to the enrichment strategy.

Regardless of whether an aggregated security finding is internally or externally enriched, it potentially supplies the knowledge base with additional information. This information is stored in a separate belief class, containing all enrichment information belonging to one finding. Since the feasibility of data enrichment depends on multiple factors, not every security finding might have additional enrichment information. Moreover, we decided against listing all contents of the aggregated security findings again in this class to avoid duplicate information. Instead, we relied on queries to retrieve the necessary enrichment data when querying for aggregated security findings. This results in the following belief class:

```
"Class Name": "Enrichment Information",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Derived",

"Finding Identifier": <>,
"Enrichment Type": [<internal, external>],
"Enrichment Information": {<Data Field 1> : <Enrichment Data>,
                           <Data Field 2> : <Enrichment Data>,...}
```

In addition to the meta-structural information, this class contains a list of all applied enrichment types: internal enrichment, external enrichment, or both. The additional information derived during the enrichment operation is stored in the "Enrichment Information" field as key-value pairs. Finally, it references the aggregated security finding to which it belongs.

The knowledge relevant during the enrichment operation is stored in instances of the "Enrichment Input" class.

```
"Class Name": "Enrichment Input",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Explicit",

"Enrichment Identifier": <>,
"Enrichment Content": {<ID 1> : <Input>,
                       <ID 2> : <Input>, ...}
```

Similar to the "Cluster Input" class, it contains the explicit belief provided from knowledge base external sources. Each instance includes an identifier that allows associating them with the respective enrichment operation and key-value pairs holding the relevant information. To add this information to the knowledge base, the */userinput* resource of the RESTful API is employed.

The internal and external enrichment is implemented by an inference rule that utilizes instances of the "Enrichment Input" class to enrich findings. In addition to the belief interface available during all logical inferences, as shown in Figure 3.6, Internet-facing access is crucial for all external enrichment operations. The inference rule consists of the following 4-tuple:

```
{"Trigger": ["Aggregated Security Finding", "Enrichment Input"],
"Code": "<Reference to the File>",
"Output": "Enrichment Information", "Name": "Enrichment"}
```

Since the inference code depends on the project, it is also constructed during its integration into the project.

4.7 Preliminary Evaluation

Finally, the approaches presented in this chapter to improve the data quality of security findings in modern industrial software development must be preliminarily evaluated against their initial requirements. This section presents the integration of all approaches into our semantic knowledge base, the evaluation, and the discussion of its results. The evaluation focused on the success of the security findings clustering approach was published in [194].

4.7.1 Evaluation Planning

The data quality improvement process presented in this chapter was evaluated against the problem description it aims to solve. The problem is split into the functional requirements each solution approach has to fulfill and the principles of modern software development that must be adhered to. In addition to evaluating each solution approach, the process comprising all solution approaches must also be analyzed for its compliance with modern software development principles. In contrast to the results presented in [194], the focus of this evaluation is broadened to cover all aspects of the quality improvement process instead of focusing on deduplication. However, both evaluations have been conducted under the same circumstances.

The research questions to be answered by the preliminary evaluation are split between fulfilling our process requirements, represented by the first four entries of Table 4.9 and the adherence to the principles of modern software development assessed by the fifth research question.

ID	Challenge	Question
RQ1	Collection Functionality	Are security findings collected from the respective sources and stored centrally
RQ2	Parsing Functionality	Is data from various security tests and assessments combined into a common format
RQ3	Deduplication Functionality	Are all security findings aggregated and correlated, avoiding duplicates in the dataset
RQ4	Enrichment Functionality	Does each security finding contain at least contain fundamental information about it

RQ5	Principle Fulfillment	Are all solution approaches and the overall process fulfilling the principles arising from modern software development

Table 4.9: Research Questions for the Preliminary Quality Improvement process

To answer our research questions, a realistic setting is crucial. To test the functional requirements, an actual project that manages security findings originating from security activities is beneficial. To answer RQ1, the project should employ a realistic set of security activities to test whether our solution approach can cover them. For the second research question, the capability of the parsing methodology to parse any report to our data model without data loss must be analyzed. Research question three is assessed by analyzing the correctness of the clustering and aggregation approach. In particular, the False Positive and False Negative rates are crucial indicators for this measurement. RQ4 is assessed by checking whether each security finding contains at least fundamental data about its cause and whether the enrichment can be applied within the project. Finally, for the fifth research question, the potential for continuous execution, the speed of results being produced, the potential data input from various development stages, and the amount of manual effort must be analyzed.

To perform all of these measurements, we selected a traditional software development project that uses the implementation of Section 4.6 as a system for managing security findings for six months. Its security activities producing security findings cover one secret scanner (Gitleaks), three Code Review tools (tfsec, Semgrep, and Bandit), and two tools for 3rd Party Component Vulnerability Scanning and SCA (Trivy and NPM Audit). The necessary customization of the knowledge base for its integration into projects is reported as part of the evaluation results.

4.7.2 Evaluation Results

The results of the evaluation are reported in two steps. First, the integration success is documented, including all customization necessary to integrate the semantic knowledge base into the project. Second, we report the data measured to answer the research questions.

As the first step of the solution approach, the API implemented the collection of all security reports. Since all tools were either orchestrated in a CI environment or a dedicated testbed, the upload was primarily performed automatically. Solely Bandit was network-wise separated from the knowledge base, resulting in weekly manual uploads of the test report. To parse these reports, one Parser Model was written per tool upfront. During the evaluation period, the project changed the Trivy reporting format from JSON to JUnit XML, necessitating one additional model. This resulted in two models for Trivy with different version numbers. The clustering and aggregation required no further changes and were directly functional. Initially, no enrichment rules were added. After two months, the project requested an enrichment, where each tool category received additional information

on the testing strategy of each tool so that practitioners could comprehend the context inherent to their reports. This information was added to every security finding. Finally, the access to all instances of the "Aggregated Security Finding" was implemented by a query. This query merged the information of the security finding with its respective instance of the "Enrichment Information" class and returned a list of all security findings upon execution.

Over 6 months, the project comprised 309 security findings with 40 to 110 reports per security activity. These 309 findings were reduced to 73 aggregated findings within 30 seconds. Even though no False Negatives were identified, two findings were incorrectly clustered. These two False Positives were corrected by user input. Finally, every security finding contained at least the attributes "Title", "Location", and "Timestamp".

4.7.3 Discussion

In the following, we want to discuss our preliminary evaluation results. Concerning the first research questions, we can see that the reports from all encountered security activities were successfully collected. Even though the subject project did not require the proactive collection approach, its reactive counterpart using the API ensured that all security reports were collected and stored centrally in the knowledge base. Towards the second research question, we were able to parse all encountered security reports to the common data format and further deal with changing report formats over time. A drawback of the evaluated project was the limited selection of security activities that did not cover every type of activity. However, this is inevitable when using realistic projects as a perfect test strategy without gaps is rarely encountered in practice. The answer to research question three can only be considered partially positive. Even though it was possible to aggregate and correlate the reported security findings, two False Positives have been encountered. This is sub-optimal, as it might lead to security findings being ignored, as discussed previously. We still consider the clustering and aggregation of security findings as successful, considering a reduction in the number of findings by more than 75%. Regarding research question four, we found that every security finding contains at least the baseline information necessary to understand the cause of it. Moreover, we validated the functionality of the internal enrichment since each security finding had enrichment information associated with it after introducing the enrichment rule in the project.

For the last research question, we investigated the fulfillment of modern development principles when using the data quality improvement process. Since the implementation of our solution approach was able to run continuously, we consider the continuous execution of our process as successful. Even though a notable manual overhead while integrating the knowledge base into the project was necessary, it is negligible when compared to manually performing the data improvement actions. The manual effort for the quality improvement was limited to the user input correcting the incorrectly clustered findings. Hence, we consider the manual effort during the process execution as acceptable. Unfortunately, the project solely provided input from the development and deployment stages, excluding, e.g., monitoring activities. Hence, not every software lifecycle stage contributed data. Time-wise, the clustering was the only processing step that notably increased the processing

time of security findings. During the evaluation period, the worst-case scenario resulted in a 30-second processing time for the clustering. With less than one minute of processing time overall, we still consider this a fast timeframe in the given context.

Threats to Validity

Even though the preliminary evaluation results indicated that our process and its implementation cover the requirements, there are multiple threats to the validity of these results. First, the selected project only covered some of the security activities employed in industrial practice. Hence, the external validity is threatened, as another project might utilize activities or necessitate collection strategies that this evaluation has not covered. Second, we did not evaluate whether the process was perceived as beneficial by the project team, even though this is the core indicator for the relevance of our methodology. However, this was partially covered by the previous evaluation [194], and the given results are sufficient as a preliminary evaluation.

4.8 Conclusion

In this chapter, we presented our process for the data quality improvement of security findings and its implementation into our platform for managing security findings. The process collects security reports from their source, parses them to a common data format for each finding, clusters and aggregates these findings to eliminate duplicates, and enriches each finding with additional information.

This chapter comprises three contributions. Our first contribution is a data model for security findings focusing on the semantic content of security findings across multiple terminologies from security activities. The second contribution is a process for the clustering and aggregation of industrial security findings using the semantic similarity-based technique LSI. Finally, we presented a coherent process for the data quality improvement of security findings and performed a preliminary evaluation of this process in a real-world scenario.

Consequently, we conclude that our process for data quality improvement mostly fulfills the demands originating from modern software development principles and the state-of-practice for industrial security findings management. However, its benefit to practitioners managing security findings in modern industrial software development projects can only be measured if subsequent processes working with the improved data are also evaluated.

Chapter 5

Security Finding Analysis and Tracking

One of the key goals to be achieved by the management of security findings is ensuring an acceptable security status of a software product according to multiple criteria. To reach an adequate security status, every security finding requires a response that contributes to achieving this goal while balancing it with other stakeholder interests, like additional software features. Each security finding must be analyzed and tracked throughout its lifecycle to enable and support the decision-making process for security finding responses.

In this chapter, we present the problems necessitating the analysis and tracking of security findings and describe our solution approaches for multiple types of tracking, the analysis of security findings according to various criteria, the prioritization of a finding response, and the decision-making on the finding response. The integration of these solution approaches into our semantic knowledge base, given the previous data quality improvement and its preliminary evaluation, concludes this chapter. The prioritization process for security findings and its evaluation was further published in [196].

5.1 Problem Description

In our preliminary problem analysis, we identified the need for two clusters of activities that must be performed based on the quality improved security data: the analysis and the tracking of security findings. According to Subsection 2.3.3, the tracking of security findings comprises four requirements, including:

1. The secure **Documentation** of security findings and their data
2. The **Status Tracking** of security findings throughout and beyond the security findings management process
3. The **History Tracking** of security findings about their appearance
4. The **Correlation** of security findings appearance

Moreover, Subsection 2.3.3 also identified seven requirements for the analysis of security findings, covering:

1. The **Verification & Validation** of every security finding's applicability
2. The **Classification** of every security finding
3. The **Root Cause Analysis** of security findings
4. The **Clarification** of finding circumstances
5. The **Risk Analysis** of security findings
6. The **Prioritization** of security findings
7. The **Decision** on subsequent actions for each finding

Similar to all previous solution approaches, also the ones for the requirements above must mind the principles arising from modern software development:

- Topic I: Any measuring or tracking activity must collect business relevant indicators
- Topic II: The decision-making on subsequent actions must try to minimize work packages for security improvements
- Topic III: The manual effort for any analysis and tracking activity must be minimized
- Topic IV: The findings tracking and analysis must provide fast results
- Topic V: Any manual activity must be previously supplied by the information necessary to perform the activity
- Topic IX: Any manual activity must mind the cross-disciplinary team structure and enable collaboration
- Topic X: Any decision must be taken with the customer and its goals in focus

In contrast to previous chapters, some principles apply exclusively to specific requirements instead of to every solution approach. The analysis of these problems from the perspective of a security findings management methodology allows to cluster activities with similar prerequisites and outcomes into the same requirement. Hence, the eleven requirements are clustered into seven problems this chapter aims to solve.

History Tracking and History Correlation of Security Findings

The first piece of information that is crucial to decide on subsequent actions for a security finding is its occurrence in security reports. A finding frequently identified by multiple activities has more likely an impact on the security of a software product than a finding that has not been reported recently. Moreover, security findings that reappear after supposedly being fixed can indicate an unsuccessful fix or provide strategies for resolving the newly identified security finding. These conclusions can only be drawn by correlating the finding's occurrence and the type of finding over time. Hence, the history of security findings must be tracked, and their appearance must be correlated over time.

Process Tracking of Security Findings

Each security finding reaches different stages throughout its lifecycle. From its first identification, over the validation of its existence, to its potential mitigation, the status of a security finding continuously changes. To manage security findings effectively, the current status of a finding must be known at any time. Otherwise, security findings that have already been verified as False Positive might be planned for mitigation, or findings that are already solved are still contained in the risk portfolio of a software product. Therefore, the status of security findings must be tracked throughout their lifecycle.

Documentation of Security Findings

Another problem is the information generated while managing security findings in modern industrial software development projects. The reports created by security activities solely present a fraction of the data necessary to address security findings. The findings' applicability, risk towards the software product, and logical consequences concerning other security processes are all part of the information used during the findings management. Moreover, the information is not only necessary for the process itself but might also be crucial for subsequent compliance investigations or audits. Minding the confidential nature of this information, the data must be protected from alteration or deletion, and access to it must be carefully evaluated. Therefore, the documentation of security findings must comprise all information created during the security findings management process and protect the information from malicious actors and inadequate access.

Verification of Security Findings

One of the biggest challenges when managing security findings in industrial software development projects are findings that are either incorrectly identified (False Positives) or have no impact on the software. All subsequent activities that manage security findings without impacting the software product waste scarce development resources. Hence, analyzing unverified security findings can diminish the efficiency of the security findings management process and must, therefore, be avoided. Consequently, security findings must be verified and their effect on software security must be assessed before advancing to later analysis steps.

Investigation of Security Findings

In advance to the mitigation of security findings in industry, details about the findings must be gathered. Even though the security activity already provides information, the data is frequently incorrect or lacks details to decide on subsequent actions. Due to a static perspective of security activities on the product, reporting security findings with an incorrect cause is a common problem in industry. To avoid subsequent process steps working with this inaccurate data and potentially reaching falsified conclusions, the analysis of each finding's root cause is crucial. Whether a root cause analysis is sufficient or further classifications and correlations are necessary to investigate a finding depends on various factors within each project. Hence, the investigation and analysis of security findings must support these particularities inherent to each project.

Risk Analysis and Prioritization of Security Findings

The limited work that can be done is another challenge in industrial software development projects. In practice, security improvements often have to compete with functional enhancements of the software, limiting the amount of effort that can be spent on security findings management. The mitigation of security findings requires the involvement of developers, necessitating the dedication of resources for every security improvement. Therefore, prioritizing security findings is essential to address the most relevant findings first. One indication of the importance of a finding is the risk it poses to the security of the software. However, not only the risk imposed by the security finding determines the significance of the finding to be fixed. Other factors coming from project-specific constraints are often equally relevant to the priority of a finding response. Especially findings that can be solved with little effort are often preferred, as they allow an improvement of the software's security status while using only limited development resources. Therefore, prioritizing security findings must consider the risk imposed by the finding and project-specific constraints.

Decision on Subsequent Actions

Ultimately, some decision has to be taken on how to respond to an identified security finding. The most obvious conclusion is to fix the finding's root cause. However, not every finding can be fixed or is worth to be fixed from a business perspective. These informed decisions on how to proceed with findings can, however, only be taken in collaboration with project team members using evidence-based information. To fulfill this demand, our solution approach must consider all potential responses to a security finding while enabling the necessary collaboration.

5.2 Related Work

Current research proposes various strategies for the assessment and response to security findings. Supporting these strategies, platforms for tracking and correlating security findings have been proposed and often present the state of practice for industrial practitioners.

This section presents research and work related to the three major domains of this chapter: the tracking of security findings, the analysis of security findings, and finally, the prioritization of security findings.

Presumably, the most prominent results related to the tracking of security findings are vulnerability databases. Vulnerability databases uniquely identify and track vulnerabilities throughout their lifecycle. By using vulnerability naming schemes like CVE [178] or component naming schemes like CPE [124], these databases track the occurrence of vulnerabilities in software components. Additional information like patching status provided by references gives further indications on the current status of each vulnerability. An example of a well-known and commonly used vulnerability database is the National Vulnerability Database (NVD) hosted by the U.S. government [125]. However, recent publications also address the tracking of security findings. Cadariu et al. proposed a system for tracking vulnerabilities in proprietary software [29]. In their work, they map software components in proprietary software to known vulnerabilities and raise an alert if a vulnerability is found. This strategy is typically applied in industry by 3rd Party Component Vulnerability Scanning activities during development. A similar linking approach between vulnerability databases and internal project knowledge was also proposed by Alqahtani et al. in the form of an ontological representation [5]. Even though our problem description is closely coupled to the concept of vulnerability occurrence in the software, a solution approach, minding solely the binary identification of findings, is insufficient, as we focus on the entire lifecycle of software instead of a single snapshot. Moreover, our perspective on the lifecycle of each finding necessitates a status tracking that extends the current functionality of vulnerability databases or related platforms. Further research related to the tracking of security findings addresses the lifecycle of vulnerabilities. Typically, the lifecycle starts with identifying a vulnerability and imposes implications on the specific vulnerability, depending on the stage in the vulnerability lifecycle. In their 2011 work, Joh and Malaiya assess the quantitative security risks of vulnerabilities using the vulnerability lifecycle and CVSS metrics [90]. In their work, they propose a simplified vulnerability lifecycle model, where each vulnerability can be in one of six stages between ”not discovered yet” and ”disclosed and already patched”. A similar lifecycle is proposed by Frei et al. [52], which divide the lifecycle of a vulnerability into discovery-, disclosure-, exploit-, and patch-time. Unfortunately, no lifecycle for security findings could be found in recent literature. Therefore, our research will be based on the vulnerability lifecycle but must be tailored to the use case of security findings management in software development projects.

Another prevalent issue when working with security findings is the existence of False Positives. As reported by publications across various domains, False Positives challenge the efficient processing of security findings [15, 122, 157, 3]. Several papers address the potential identification of False Positives and their elimination by data mining [114, 62] or machine learning [58]. Moreover, the current state of practice is acquired in recent research. Alahmadi et al. analyzed the prevalence of False Positive alarms in Security Operations Centers and interviewed practitioners on their subjective perception [3]. Even though they find that a considerable number (99% stated by one subject) of alarms are False Positives, most of them are related to benign triggers so that the terminology of a False Positve is

used quite vaguely in practice. Moreover, they have shown that in the context of Security Operations Centers, the validation of alarms is still commonly performed manually based on information like system logs, the environment of the alarm, or known attack vectors. Sacher presented a discussion on benign triggers resulting in False Positive alarms [157], where such False Positives could be used to refine the company strategy instead of being suppressed. Even though both publications focus on reports from security operations monitoring activities, the insights on alarm validation strategies are coupled to our work. The importance of context information for identifying and eliminating False Positives is also stated by Nadeem et al. [122]. In their work they propose adding contextual information to static code analysis tools to reduce the number of False Positive detections. This further reinforces the value of context information for identifying False Positives in the security findings management process. Research from another perspective was conducted by Choi et al., claiming that the response to certain vulnerabilities like integer overflows is not impacted by whether a False Positive was reported or not [33]. Indications that False Positives might even predict future weaknesses in the software were given by Dimastrogiovanni and Laranjeiro, who identified correlations between False Positives and subsequent vulnerabilities at the same location [40].

One of the last steps in the management of security findings is the response to each security finding. However, only a certain number of tasks can be solved in each iteration due to limited possible progress restricted by available time and project team size. Hence, not every security finding can and should be addressed in each iteration [13], introducing the importance of prioritization when managing security findings. One common factor impacting the relevance of a security finding is the risk inherent to every finding. A common approach to calculate this risk is combining the finding's impact with its probability of getting exploited [129]. The impact of security findings is often represented as a severity or criticality score. While most activities rely on the international Common Vulnerability Scoring System (CVSS) standard [135], whether and how the score looks depends on the reporting source. Calculating a precise and realistic severity score is an ongoing field of research. Maidl et al., e.g., proposed to refine the CVSS base scores with system models to achieve more precise scores [106]. Similarly, Amankwah et al. proposed an automated framework to evaluate existing vulnerabilities in web applications for their severity [9]. These represent just two examples of various papers trying to generate a severity score either by refining CVSS or creating a distinct strategy for prioritization [165, 53, 12, 100]. Even though CVSS is undoubtedly one of the most prevalent techniques, multiple other strategies on how to rate security risks or vulnerabilities exist [66, 109]. In practice, solely the severity of a security finding is not decisive for its priority but represents the basic information used in planning meetings [173] to determine its mitigation schedule. Traditional strategies to prioritize tasks in software development include, e.g., Analytical Hierarchical Processes like Planning Poker [27]. However, these approaches can not be directly applied to all types of prioritization. In contrast to functional requirements, non-functional requirements like security are still prioritized mainly by hand using 'gut-feeling' [173] in industrial practice. This is due to the uncertain nature of non-functional requirements, which limits the effectiveness of classical prioritization methods [69] and the expertise required from

additional stakeholders like technical experts [88] for its calculation. Even though the area of security findings prioritization in itself is underrepresented in literature, the domain of software bug or defect prioritization contains closely related work. For instance, Ahmed et al. developed a framework for the automated categorization and prioritization of bug reports [2] using natural language processing. Another proposal by Gökçeoğlu and Sözer is the usage of Machine Learning for the prioritization of software defects [63], which they investigated as part of a long-term case study. One interesting finding of their research is that the best accuracy can be achieved by using recent data for training, indicating that the prioritization approach or defects evolve over time. This domain contains a variety of publications [209, 181], employing different techniques to improve the prioritization of bugs or defects. However, most of the research focuses on bugs and their reports instead of structured security reports, which apply to the management of security findings.

However, not only the prioritization of a response must be considered, but also the type of response itself. At this point in the management process of security findings, the decision on how to react to a security finding can be reduced to traditional industrial product security risk management. As each security finding potentially imposes a risk on the product's security, we can follow the traditional risk assessment process, resulting in a risk treatment. Across various standards and guidelines, certain key activities can be identified to manage risk, including the identification, analysis, evaluation, and treatment of risk [76, 25, 74, 130]. In the context of security findings management, preconditions like organizational preparedness or risk identification can be treated with reduced importance since the security activities already report potential risk sources. Hence, each security finding must be analyzed for its risk, the risk evaluated against risk criteria, and treated accordingly to adhere to common process requirements in industry.

5.3 Documentation and Tracking of Security Findings

To enable subsequent analysis activities of each security finding, fundamental knowledge of a finding's occurrence and current state must be acquired. The solution approach to collect knowledge about a finding's occurrence history, the tracking of a finding's state, and the documentation and access to the data of security findings are presented in this section.

5.3.1 History Tracking of Security Findings

The first step to track security findings throughout their entire lifecycle is tracking their occurrence. In this context, the term occurrence defines whether a finding was reported by any security activity. For each report, whether or not a finding is identified in a project is binary. Either the finding is referenced by the respective report or not. However, minding that the security finding management typically manages reports from multiple security activities, this exclusively binary information is extended. This adds the source dimension, as each source can report a finding separately. Hence, this changes the reporting format

from a binary value to a list of binary values, where each field represents the respective source. Moreover, the management of security findings is a continuous activity in the software development lifecycle. Similarly, reports are also created continuously. Hence, an additional temporal dimension must be minded, extending the list of binary values to an array. These three states are depicted in Figure 5.1. As this solely represents the occurrence of one finding in the project, the array can be constructed for each finding separately.

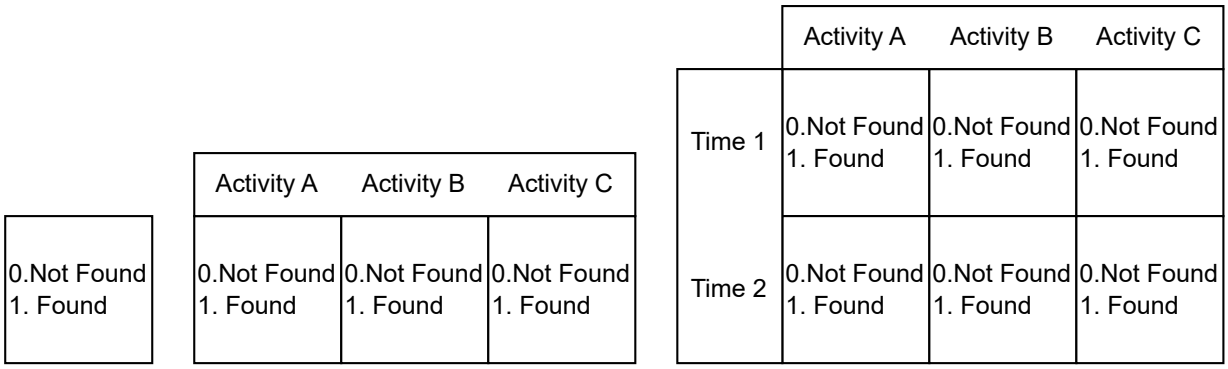

Figure 5.1: Occurrence of Security Findings across multiple Dimensions

However, this introduces the first limitation of its usage in practice. In a real-world scenario, security findings that are never reported by security activities are typically not addressed by the security findings management, as they are unknown. Minding that the National Vulnerability Database already contains more than 200.000 entries, tracking all findings potentially affecting the product is not feasible. Therefore, only those findings reported at least once in the project may be tracked. Another limitation arising from practice is the reliability of data input. While security checks like automated static code analysis present a low-effort activity that can be conducted daily or even more frequently, an in-depth analysis like a Penetration Test typically takes weeks to complete. Therefore, the frequency of reports originating from automated activities is generally higher than from manual activities. Hence, a distribution of values as depicted on the right side of Figure 5.1 is very unlikely. Instead, each timestamp might only inform about a fraction of all activities. This implies that conclusions from tracking a finding's history must be aware of this constraint.

Our solution approach for the history tracking of each finding follows our capability for collecting historical data about findings while minding the practical constraints. For each security finding in a project, a tracking entry is established. The entry tracks all activities that have identified this finding with all timestamps when the occurrences have been reported. An example of these entries is shown below.

```
Finding A -     Activity 1  Activity 2
                01.01.2000  20.01.2000
                05.01.2000
```

```
                07.01.2000
Finding B -     Activity 1  Activity 3
                01.01.2000  02.01.2000
                05.01.2000
```

While *Finding A* was identified by Activities 1 and 2, *Finding B* was reported by Activities 1 and 3. From the timestamps, we see that both findings have only been identified by Activities 2 and 3 once, while Activity 1 reported them at least twice. Moreover, we can see that Activity 1 reported *Finding A* a third time, where *Finding B* was not identified. Hence, each tracking entry solely contains the activities and timestamps where a finding has been reported.

Crucial for the relevance of this information are conclusions that can be drawn from the tracking information. A first indication of what might be relevant is given by the entries in vulnerability databases. Each vulnerability is reported with the date it has been published, the date it was last changed, and a history of all changes to the vulnerability information. Applied to the information available for the history tracking, this would cover the timestamps when the finding was identified first, when anything in the data of the finding was changed, and the entire history of changes to the finding's data. However, only the date it was found first can be derived from the tracking history. Information on data changes necessitates a further investigation of the finding's data. In addition to the fields proposed by vulnerability databases, multiple other conclusions can be drawn from the data. Analogous to the first identification of a finding, its last identification can also give valuable insights. Moreover, the number and list of all activities that have reported the finding can provide helpful information. The number of identifications per activity can further refine this data. Finally, a finding that is fixed should not be reported anymore. Therefore, whether a finding appeared in the most recent report of an activity is also documented. These conclusions are listed with their respective description and data types in Table 5.1. Even though these conclusions can be deducted manually from the raw tracking data, its explicit documentation reduces the effort to access the information. These conclusions extend the information of the tracking entry belonging to each reported security finding.

Field	**Type**	**Description**
First Found	Timestamp	Timestamp, when the finding was identified first by any activity
Last Change	Timestamp	Timestamp, when the finding data was changed last
Change History	List	List of change descriptions with reference to their respective activity and timestamp
Last Found	Timestamp	Timestamp, when the finding was identified last by any activity
Activity Number	Number	Number of activities that reported the finding

Activity Statistic	List	List of all activities that reported the finding with their respective amount of identifications
Recently Reported	List	List of all activities that reported the finding with a binary value stating whether the finding appeared in the most recent report of the activity

Table 5.1: Conclusions based on Findings History

Continuing with *Finding A* from the example above as tracking history, the *First Found* timestamp would be set to `01.01.2000`. The *Last Found* timestamp would be set to `20.01.2000`, while the *Activity Number* would be two. The *Activity Statistics* presents the two activities with their respective timestamps by `{Activity 1: 3 , Activity 2 : 1}`. The remaining three fields, the *Last Change* and *Change History*, and *Recently Reported* cannot be derived solely from the data available in the initial example. However, realistic assumption could lead to `20.01.2000` as date for the *Last Change*, `{Activity 1: No , Activity 2 : Yes}` as the *Recently Reported* data field, and list

```
{01.01.2000 : "Created Finding",
07.01.2000 : "Changed CVSS Score",
20.01.2000 : "Added Description"}
```

as *Change History*. Multiple other conclusions are imaginable in combination with further information about other findings acquired during the security findings management process.

5.3.2 Status Tracking of Security Findings

When managing security findings in industrial software development projects, implications apply to each finding. While these implications are quite limited at the beginning of the management process, more are acquired by investigating and analyzing the findings. Typically, these are defined as states a security finding is currently in. Our solution approach has to support the annotation of security findings with these implications and ensure their persistence and lifecycle throughout the management of each security finding.

To design a solution approach that tracks the current status of each security finding, a list of possible states a security finding may reach must be identified. We define these possible states as *Security Finding State*. The trivial lifecycle includes exclusively the identification of a security finding and the solution of this finding. This lifecycle consists of exactly two states, where each finding can be in either one of those two states. A common strategy for the representation of the vulnerability lifecycle is the usage of state machines. Likewise, these diagrams can also represent the state of security findings. Following the definition for finite state machines, each security finding state is represented by a state in the machine. These states are mutually exclusive, meaning a finding can only have one active state at a time. The starting point of any security finding is its identification, resulting in the status ”Open”. Therefore, our starting state is ”Open”. In contrast to other state machines employed in the vulnerability lifecycle, we disregard all findings that have

not been identified yet. As explained in the last subsection, these findings are irrelevant for security findings management and, therefore, not considered a starting state. Based on the trivial lifecycle mentioned previously, we can also identify a second state representing a solved finding by the state "Fixed". This also symbolizes one of the possible final states a security finding can reach. In the context of security findings management, the final state should allow security findings to persist permanently, according to the goals and intentions of the project team and stakeholders. Accordingly, this state represents a potential conclusion of the management process for the respective security finding. Minding the low reliability of information provided by security activities, final states may be switched again to non-final states. Even if a finding fix is implemented and validated, any software change might re-introduce the same finding, resulting in a status change. Therefore, security findings may leave final states in case of certain events. Whether a state can be considered final depends on the project and its goals. While some projects might consider the state of a finding that is no longer identified by activities as final, others might consider only findings with manually validated responses as final. Hence, the selection and interpretation of a finding's final state depends on the project.

However, the project not only interprets the implications that a finding state imposes on the finding, but it might also define additional finding states. Consequently, our solution approach must provide the capability to extend the finite state machine model by additional states. Expanding the default two-state model before, additional security finding states can be derived from our previous research. As identified in the last subsection, findings may also disappear if they are not reported anymore. This adds an additional state to the default machine model. Minding the challenge of False Positives or not applicable findings, two further states are necessary to cover these cases. Information about potential patches or exploits, as proposed in other literature on the vulnerability lifecycle, is disregarded. Even though this information is valuable to decision-makers for subsequent activities like prioritization, information about potential patches or exploits is not mutually exclusive to other states.

Similar to the definition of possible states for findings, the alphabet defining events for a state change must be specified. Since the project defines the respective states and implications of each state, any event resulting in a status change must also be determined by the project. Considering the fix of a finding, the interpretation when its state can be switched to fixed can highly differentiate. While one project could consider the implementation of a fix as sufficient, others might see the necessity to have it verified under a four-eyes principle. Hence, an exhaustive pre-defined alphabet is neither feasible nor desirable. Instead, we propose a common alphabet to be refined depending on the project's specifications. This results in the following state machine definition:
$Q : \{Open, Fixed, Absent, FalsePositive, Invalid\}$
$q_0 : \{Open\}$
$F : \{Fixed, Absent, FalsePositive, Invalid\}$
$\sum : \{ImplementedFindingFix, Findingnotreportedanymore,$
$DeterminedFindingasFalsePositive, DeterminedFindingasInvalid,$
$IdentifiedincorrectAnalysis, IdentifiedSecurityFinding\}$

The resulting diagram is depicted in Figure 5.2. It shows that each state can be reached from every other state with one exception. The "Absent" state can only be reached from the "Open" state since a finding that is not reported anymore can nevertheless be a False Positive or invalid.

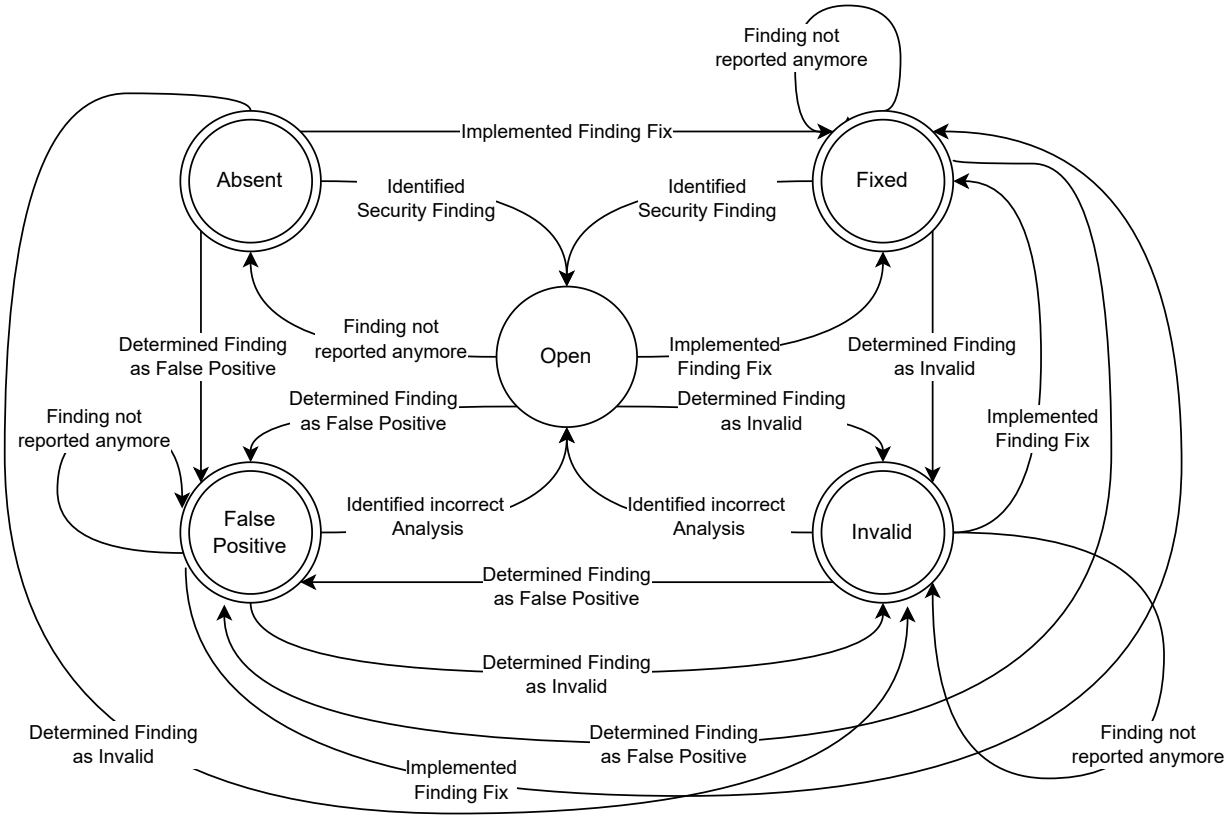

Figure 5.2: Deterministic State Diagram of the Security Findings Lifecycle

A security finding can reach several states throughout its lifecycle based on the occurring events. This creates a history of states that must be tracked as part of our solution approach. Moreover, findings might occur at multiple locations, depending on where they have been reported. The status of a security finding can differ between locations, as one location might be fixed while another has been reported as invalid. Therefore, each location has an individual status with a respective history.

Integration Process

In contrast to the history tracking of security findings, integrating the status tracking into a project requires preceding steps to tailor it to the project goals and scope. This includes the potential introduction of additional states, an explanation of each state's implications on the finding, the definition of all final states, and criteria that define how one state can be reached from others, summarized as state machine definition.

5.3.3 Documentation of Security Findings

During the analysis and tracking of security findings, several pieces of information are acquired. This information is crucial to decide on the response to a security finding, prove compliance with certain norms, or conduct forensic investigations after an incident. Therefore, information documentation is essential for all solution approaches presented in this chapter.

However, its solution approach is straightforward. Any information identified and reported during the management of a security finding must be documented and accessible to the project team and stakeholders. The security findings management platform provides the documentation capability and must be utilized by the respective solution approaches in this chapter. In addition to the documentation, the information must be protected accordingly. This implies the maintenance of the traditional protection goals *Confidentiality*, *Integrity*, and *Availability*.

In our context, maintaining *Confidentiality* ensures that unauthorized actors cannot access the information documented during the security findings management process. Considering our platform, this affects two aspects: The protection of data at rest and the protection of data in motion. The data is in motion whenever new reports or information is sent to the knowledge base or data is retrieved from the knowledge base. To protect against illegitimate data access in these cases, all communication channels with the platform must be secured. Our solution approach uses the TLS protocol with a server certificate that a commonly trusted Central Authority handed out. Most of the time, the data rests on some data storage. This includes data access using interface queries and direct access to the data storage itself. To avoid unauthorized reading access to the information via the data storage, we suggest an encryption of the clear text information. To protect the data from being illegitimately read via the interface, we propose integrating an identity and access management system with a user and role concept.

Analogous, the *Integrity* of the data must be protected. In our context, this implies that unauthorized actors cannot alter or delete the data. To maintain this protection goal, we split the solution approach into the protection of data in motion and at rest. The first solution approach inherently ensures the protection of data in motion. The TLS protocol, combined with a server certificate, not only encrypts the data in motion but also protects its integrity with a message authentication code. For the data at rest on its native storage, we recommend data integrity algorithms traditionally available in any modern data storage. For the integrity protection of the interface, the role concept must be extended, covering the read access to data and any modifications. This ensures that only authenticated and authorized actors may change data via the interface.

Finally, the *Availability* of data refers in our context to the ability to access information whenever necessary. Therefore, teams and stakeholders must be able to read or modify the data when managing security findings. This necessitates proper hardware specifications for the system the platform is running on and protection against any denial-of-service attacks. Since this highly depends on the implementation of the knowledge base and the number of security findings processed on the same hardware, no general solution to the protection

of availability can be given.

Integration Process

To integrate a platform that documents the information gathered during the security findings management process and protect it accordingly, a role concept for accessing and modifying the data must be created. This includes which person is allowed to perform which actions on the dataset. Furthermore, any processes interacting with the data, like uploading reports, should also be included in the role concept as functional accounts. Based on this definition, data protection can be customized to the project's demand.

5.4 Analysis of Security Findings

Analyzing the root cause and background of reported security findings represents one of the fundamental activities in the security findings management process. The results of a finding's analysis provide insights into the relevance of the finding for the project and give additional information on subsequent finding responses. This section presents the solution approach to verifying and validating security findings and the proposal on how to support the analysis of security findings in general.

5.4.1 Verification of Security Findings

To manage security findings effectively, any reported finding not affecting the software must be eliminated from subsequent analysis or solution efforts. A typical example of security findings without effect on the software is False Positives. This type of finding is identified and reported by a security activity, but its existence cannot be verified in practice. Suppose the following stages assume that the said finding affects the software and conduct subsequent activities or responses. In this case, the effort related to these activities has no positive impact on the overall security state of the product, diminishing the efficiency of the overall security findings management process. Hence, the first step in analyzing security findings is the verification of their existence and applicability.

As discussed in the work related to this section, identifying and eliminating False Positives is still a tedious manual task in practice. Every reported finding must be correlated with context information like configuration files, dependent libraries, underlying platform, and infrastructure [3] to judge whether it might be invalid. Considering several hundred findings per security report are not uncommon, only those findings with the most significant potential impact on the software can be manually investigated. This can result in many unreported invalid findings, falsifying any overview of the security status of the project. Minding the scenario of modern industrial software development, with new reports arriving continuously, the effort to identify invalid findings is increased even further.

Therefore, the current state of practice is not desirable for efficiently managing security findings in modern industrial software development projects. Instead, the manual effort should be minimized, and repetitive work should be avoided to conform with modern

development principles. An obvious solution would be the automation of False Positive identification. Scientific literature proposes multiple automated solutions ranging from machine learning to data mining. However, the drawback of all solutions in this domain is the possibility of False Positives when trying to identify False Positives. Whenever a security finding is denoted as invalid, it is considered irrelevant to the project. Typically, this implies that the problem stated in the finding does not affect the software product and is, therefore, not imposing a risk to the overall security state. Consequently, it is disregarded in subsequent stages of the security findings management, regardless of how severely the finding might impact the software. In particular, the finding will neither be mitigated nor shown in the overall security status of the software. Imagining the initial classification of the finding as False Positive is incorrect, a potentially critical security shortcoming is affecting the software and intentionally ignored. Therefore, any security finding incorrectly flagged as False Positive can have a devastating impact on the security of any software product. Hence, preventing False Positives during the identification of False Positives is essential.

Unfortunately, the techniques proposed in scientific literature still have a noteworthy False Positive rate, considering experiments are performed under optimal conditions (e.g., 3% for only one specific group of findings). Moreover, most techniques utilize historical information about security findings. While this is beneficial since existing False Positives will likely be identified again, it requires historical information about False Positives upfront to be effective. To make these techniques usable within our methodology, a set of historical data on False Positives must be known, and the results of the techniques must be manually reviewed to validate each identified False Positive. Since a similar effort is required during the traditional manual review, we decided against automating the security findings verification and validation process. Instead, we divide the process into two steps. During the first step, all security findings are validated and invalid findings are identified. In the second step, the validity information is documented for each finding.

Security Finding Validation

Before any security finding can be classified as invalid, the definition for this property must be specified. Following the underlying challenge, we define any security finding as invalid if it does not negatively impact the product's security. The most prominent case of an invalid security finding is a False Positive. According to binary classification, a False Positive (Type I Error) represents a test result that incorrectly indicates that a particular condition is fulfilled. In the context of security activities, this implies that a check found a security finding at a location containing no security finding. The root cause is always an error in the activity that conducted the check. Exemplary would be a flawed test case that checks for properties that are not correlated with the reported finding. Another case can occur when a finding is reported and can be found, but does not affect the security of the software. An example would be a finding in program code that is never executed. In this case, the finding exists and was correctly identified, but it is irrelevant to the security of the software and, therefore, invalid. Whenever the finding negatively impacts the security

of the software, it is considered valid. This results in the following three validity results:

- False Positive: A finding that was reported, but can not be found
- Not Applicable: A finding that was confirmed, but does not affect the security of the software
- Applicable: A finding that affects the security of the software

To investigate the validity of each finding, project team members have to collect evidence that is contextually related to the security finding. The context information that should be collected highly differentiates, based on factors like finding type, security activity that identified it, or product [122, 3]. While investigating a security finding identified by a static code analysis tool benefits from collecting configurations and code snippets, the analysis of a Penetration Testing finding benefits from aspects like the infrastructure configuration. Typical context information includes the software code, operation environment, configuration files, underlying platforms, third-party components in general, like libraries, and the intended system behavior. Since no clear guidance on the necessary context information can be given, this decision depends on the investigating party. We define this party as one or multiple project team members. Even though one team member is sufficient in most cases, certain findings might necessitate additional domain expertise. Hence, the project team conducts the investigation of security findings and is supported by context information from the project.

Validity Documentation

After the security findings are validated, the results must be documented so that subsequent processes know about potentially invalid findings. Traditionally, all invalid security findings are eliminated from reports so that only valid findings are processed and potentially responded to. However, recent research indicates that False Positives also yield knowledge that might be useful for the project and other stakeholders in the company [40, 3, 157]. As confirmed by our preliminary research on the requirements of the security findings management, the knowledge about False Positives can support the Q&A team in tailoring the security testing strategy, potentially avoiding False Positives in the future. Therefore, our solution approach solely documents the results of the previous investigation instead of removing the findings entirely. This has another remarkable advantage to the security findings management process, as security reports created later can be mapped to the existing decisions. Consequently, a finding that has already been identified as invalid persists in this state throughout newly collected security activity reports, drastically reducing the effort for the validation and verification of security findings. Even though this persistence has effort-wise advantages, it could also threaten security awareness. Findings documented as invalid might only be correct for a limited time period. Even if a finding was determined invalid at one point, this might not be permanent as new attack vectors could expose the finding. Therefore, the results of the validity assessment must be reviewed periodically. The review period depends on the project and is consequently defined upfront. Typical time frames would be a semi-annual or four-monthly review.

The approach for documenting a finding's validity must differentiate between both outcomes. When a security finding is invalid, it has implications on its lifecycle. Since it does not impact the security of the software, further analysis or even responses are not required. Therefore, this represents the end of its lifecycle, indicating a final state in the status tracking, presented in the last section. As "False Positive" and "Not Applicable" indicate different properties of the finding, distinct states must represent them. In contrast, any valid finding has subsequent stages in its lifecycle. The information that a security finding is applicable solely shows that it has already been validated. Therefore, this knowledge must also be documented with an additional finding status. In summary, all results of the security findings validation are documented using individual finding statuses with respective explanations.

Figure 5.3 depicts the entire security findings validation process.

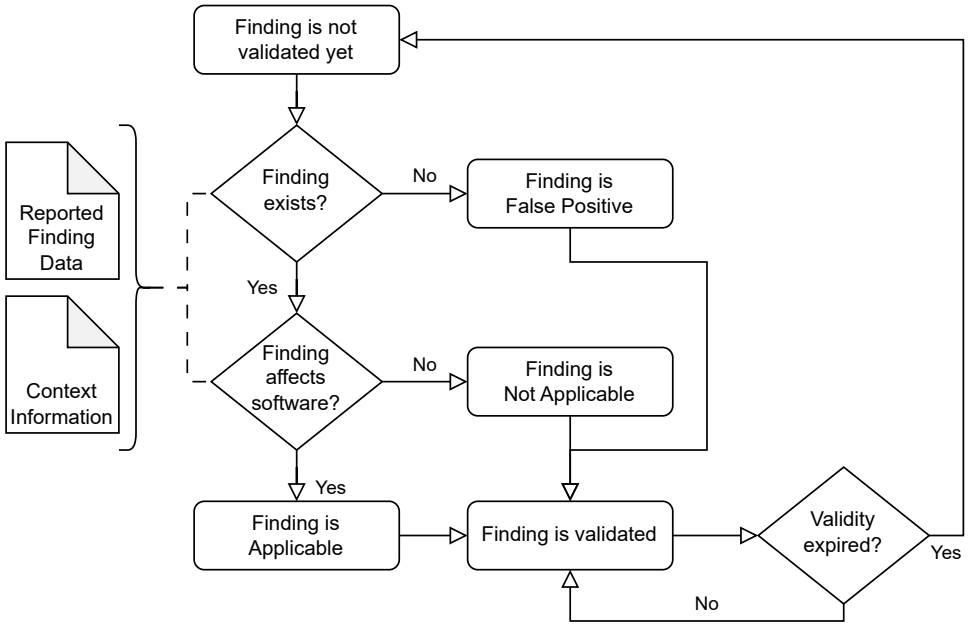

Figure 5.3: Flow Chart of the Security Findings Validation Process

Integration Process

Due to the manual effort of the security findings validation process, its integration into the project benefits from guidance on how to conduct the validation. In particular, this should explain the differences between False Positives and not applicable findings. Moreover, the review period for invalid findings must be defined upfront.

5.4.2 Investigation of Security Findings

During our preliminary assessment of requirements for managing the security findings, several requirements stated the necessity to analyze each security finding. The scope

of the analysis ranges from particular properties of a finding to the impact of a finding response on the project and customer. Even though some factors are commonly analyzed in industrial software development projects, like a root cause analysis, the selection of which analysis is conducted depends on the project. However, not only the project circumstances but also the finding itself impacts the selection of the analysis approach. A finding type that is well known by the project members might not require an in-depth investigation for its mitigation effort, while a yet unknown finding might receive a more detailed analysis. Commonly applied investigation strategies include a root cause analysis, the determination of finding circumstances, an effort estimation for the finding response, or the impact of a finding's mitigation on the product/customer.

As the investigation of security findings is a case-by-case decision, no common strategy can be recommended as part of our solution approach. Instead, we propose that the conclusions of previous stages of the security findings management must be available during any analysis protocol, and the investigation results must be documented for subsequent activities. An elaborate explanation of the presentation of results from previous stages is avoided, as the topic is addressed in Chapter 6. The documentation of analysis results follows the idea of sticky notes. As no exhaustive list of all possible investigations can be collected and the results depend on the scope of each analysis, we solely collect each result as 3-tuple *Title*, *Analysis Description*, and *Analysis Result*. The *Title* is a summary of the conducted activity. This is more precisely elaborated on in *Analysis Description*. Since not every team member might know the procedures inherent to an analysis approach, this can be disclosed in more detail in this field. Finally, the analysis results are documented in *Analysis Result*. This may include the knowledge acquired by the analysis or conclusions that can be drawn from that. An example of this 3-tuple is depicted below:

```
{
"Title": "Effort Estimation",
"Analysis Description": "Check the effort to mitigate or fix the finding",
"Analysis Result": "Finding Fix requires 3 hours
                    of senior development time"
}
```

This structure ensures that arbitrary investigations can be conducted and documented.

5.5 Prioritizing and Deciding on Security Findings

With hundreds of findings commonly reported in industrial software development projects, deciding how and when to respond to an identified security finding is essential to perform impactful actions. This decision depends not exclusively on the severity of a finding but includes multiple other factors. This section presents the solution approach to the prioritization and decisions on subsequent actions for security findings. The prioritization process for security findings is published in [196]. Since the decision-making process requires all data previously constructed, this section is closely coupled with the information communication presented in Chapter 6.

5.5.1 Severity Scoring of Security Findings

The most commonly supplied indicator for the relevance of a security finding is its severity or criticality. This score guides practitioners on how severe a finding impacts the security of software that contains it. This severity score is the first indicator for a findings priority [53, 165]. Hence, it is a crucial aspect in the prioritization of security findings. A typical format for this severity is the Common Vulnerability Scoring System, which utilizes a numeric score between 0.0 and 10.0 to present the relevance of a vulnerability [135]. However, not every security finding is a vulnerability. Therefore, other scoring systems also exist, changing either the calculation approach, the representation of the final score, or both. Examples of the first aspect can be found in the Related Work section of this chapter. Moreover, the representation of the resulting severity score also differentiates between sources, ranging from numeric values to qualitative severity ratings with diverse scales, phrases, and formats (High, Medium, Low; Critical, Severe, Moderate; 1-100; 0.0-10.0).

Minding the scope of the security findings management, practitioners encounter the severity of findings created by various sources. A standard scale or calculation approach across activities cannot be assumed. However, practitioners must be able to compare the severity of findings against each other and depend on reliable severity scores for an accurate rating of the finding's priority. Hence, the first part of our solution approach has to ensure that each security finding contains a severity score on a scale commonly used across all findings. We define this severity score as *Base Score*.

The first step towards this Base Score is creating a common scale with which all security findings are aligned. Due to its broad acceptance, we utilized a modified CVSSv3 scale with numeric values ranging between 0 and 100. This allows a granular ranking of each severity while supposedly being more intuitive than floating point numbers used in CVSS. To cope with the varying scoring approaches by different security activities, we suggest using semantic models to comprehend the content of each security finding and derive a score on the common scale. Since each activity reports data in a different format, these models are tailored to the respective activity. Moreover, the same activity might report different information categories depending on the type of finding. An example would be the same activity reporting differentiating CVSS scores from distinct sources (CVE and Debian Security Advisory). Hence, one model must be constructed for each activity. These models must compensate for each source activity's format, calculation approach, context information, and potential incorrectness. Hence, they must be constructed manually by practitioners with security expertise and experience with the respective source activity to ensure the model can handle the source data appropriately and build a reliable score. To disclose the model's functionality to practitioners utilizing it, each model output is accompanied by an explanation of how the score was calculated and based on which information. Consequently, the execution of a model outputs the Base Score for this model and a corresponding explanation.

Due to the preceding clustering of findings, one security finding might contain severity information from multiple, hardly predictable sources (two static code review tools reporting the same finding). Therefore, our solution approach must be able to aggregate the

severity score from multiple models into one Base Score, representing the respective security finding. Since domain experts manually create the models, we can assume that the resulting scores inherently hold the same degree of accuracy. Hence, significant discrepancies are unlikely. Therefore, the final Base Score is created as the mean value of all model outputs and a list of all associated explanations.

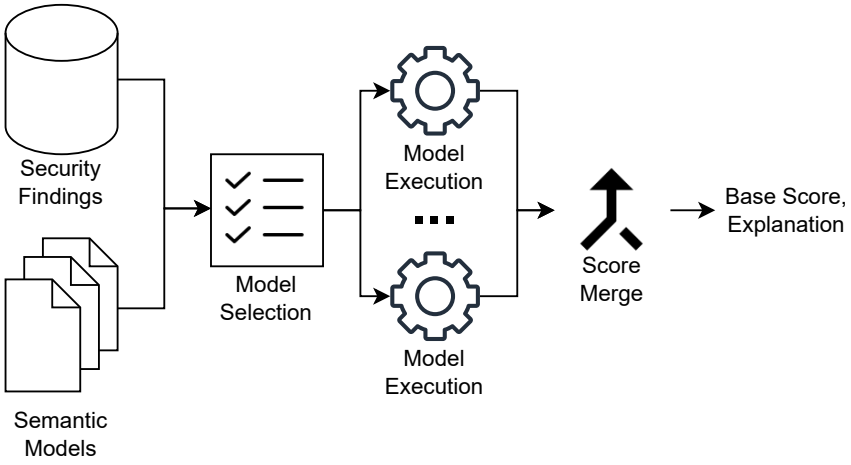

Figure 5.4: Base Score Calculation Process

In Figure 5.4, the entire process is shown. An example calculation of the Base Score would begin with creating models for each source activity. Since the structure of each model highly depends on the respective source and the data it reports, no general modeling approach exists. However, practitioners typically utilize an if-then design to identify available information and map the existing information to the common scale. An example of a simple model could check whether a CVSS score from CVE exists and linearly map the numeric CVSS value to our scale by multiplying it by ten. Otherwise, the finding receives a comparably high static score to avoid it being missed. As an explanation, either the mapping from the CVSS score or the static assignment of a score is added. This model would be connected to the respective source.

```
if exists(security_finding.CVSS_Score) and
   security_finding.CVSS_Score.source == "CVE" then
    base_score = security_finding.CVSS_Score.score * 10
    return base_score, "Linear mapping of the CVSS Score from CVE"
else
    return 80, "Static Score, due to a lack of reliable information"
endif
```

Starting with the exemplary calculation, the process must identify whether an applicable model exists for the existing security findings based on the activity that reported each finding. Afterward, the model is executed on the finding's data, creating a score and an explanation. With multiple models, the mean of all scores would be utilized as Base Score.

Integration Process

To integrate the calculation of the Base Score into software development projects, each employed activity requires a model that can be utilized to calculate the score. Similar to the integration of the parsing strategy, a domain expert has to create one model per security activity reporting findings in the project.

5.5.2 Prioritization of Security Findings

However, the severity calculated in the last step only indicates the relevance of security findings compared to one another. Moreover, it disregards several factors impacting the importance of a finding response, like vulnerability chains resulting in an attack vector. Therefore, the priority of responding to a security finding by, for example, solving or mitigating it exceeds its plain severity.

The traditional prioritization of tasks in software engineering involves various stakeholders and factual aspects to determine the relevance of a task. When responding to a security finding, the impact factors are not less plentiful. The complexity of solving the issue, the resources available in the project, the project planning, and multiple others impact whether, how, and when a security finding is responded to. However, these aspects should not only be used to derive the priority of a security finding but must further be formalized and documented to ensure the transparency of each decision. Therefore, the prioritization of security findings must be a flexible but formalized process to include all relevant factors, resulting in a single reliable prioritization score per security finding.

In practice, two types of entities providing input must be considered: human project stakeholders and automated systems. An example of the latter is a list of critical vulnerabilities containing issues that must be mitigated at all costs, like log4j in 2021. Each input factor further differentiates in their relevance to the project, necessitating a balanced priority. We define this priority as *Refined Score*.

Following the scoring approach of the Base Score, the Refined Score is a numeric score on a scale between 0 and 100. However, solely a numeric value might be interpreted differently based on personal experience by project team members. Since a common understanding is essential [158], the resulting prioritization score must also explain what it implies. Furthermore, humans and IT systems (e.g., ticketing) use the score afterward, so it must be comprehensible and processable for both. Moreover, the Refined Score not only represents the severity or risk of a finding but also its priority in a project. Hence, we extended the result interpretation of the traditional CVSSv3 approach by the concepts of urgency and consequence. Depending on the score, qualitative information on the urgency, differentiating between five levels, and recommendations on the consequence for each level are supplied. In contrast to CVSS, we extended the lowest category up to a priority score of ten since CVSS lacks a score range for existing findings that are entirely negligible. The resulting orientation scale is depicted in Table 5.2.

Our solution approach to formalize and document the influence factors on the priority of security finding are *Finding Prioritization Policies* (FPP). Each FPP consists of five components listed below:

Score	Urgency	Consequence
0-10	None	No consequence to be taken
11-39	Low	To be solved on demand
40-69	Medium	Deep analysis in the upcoming iterations.
70-89	High	Deep analysis in this iteration. Mitigation in upcoming iterations.
>=90	Critical	To be mitigated in this/the next iteration.

Table 5.2: Orientation Scale for Prioritization Score

- Provider: Origin of the Policy
- Policy Explanation: Explanation of the Policy
- Rule: Findings affected by the Policy
- Action: Impact of the Policy on the Priority
- Relevance: Relevance of the Policy

The *Provider* and the *Policy Explanation* fields document who exerts influence on the prioritization score of a security finding and why. This information clarifies the background of each policy and aims to achieve transparency for each decision, as the source of each influence factor is traceable. The *Rule* field defines which security findings are affected by the policy. Specific influences may impact not only a single finding but entire groups, depending on the properties of the findings. Examples include findings at particular locations or that originate from specific sources. The *Action* field contains the impact of the policy on the final score. Since the final score is defined on a scale of 0 to 100, this field contains either an absolute value (0-100) or a mathematical operation to derive the Refined Score from the Base Score. The second case allows adding, deleting, multiplying, or dividing the Base Score by any given numeric value. It is intended for entire classes of findings if they are, e.g., highly exposed and should consequently be treated with a higher priority. To ensure the correct outcome, the information provided in the *Action* field must be derived from the orientation scale (Table 5.2) and hence previously known to the entity providing the input. The last component, the *Relevance*, represents the relevance of the FPPs *Action* compared to other FPPs matching the same finding. This corresponds to the varying influence weights between different entities providing the input. The combination of these five components allows a formalized influence on the prioritization of security findings that is flexible, traceable, and standardized. Process-wise, this input shall be provided by all team members independent of each other.

To construct the Refined Score for a security finding, all influence factors, formalized as FPPs, and the Base Score must be considered. Hence, the first step is the collection of all FPPs affecting the respective finding, resulting in zero to N different applicable

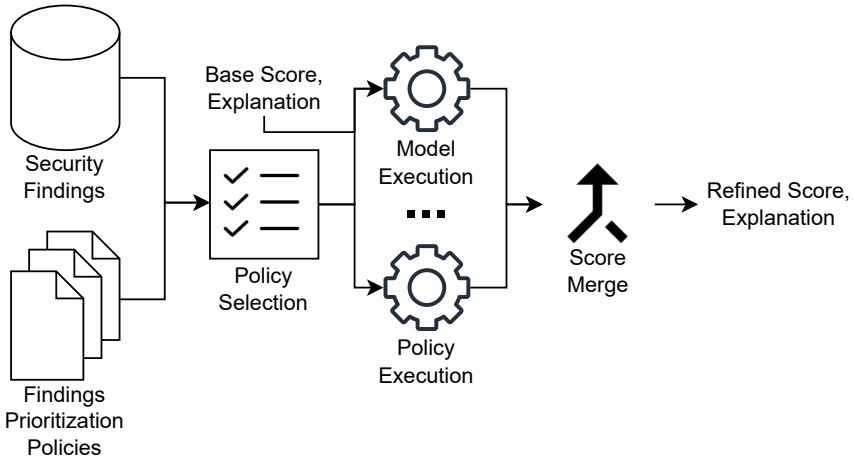

Figure 5.5: Refined Score Calculation Process

policies. If no FPP affects the selected finding, no factors except for the severity impact the priority of this finding. Therefore, the Base Score equals the Refined Score. If FPPs are found, the Action of each policy is applied to the Base Score, resulting in a list of different intermediate scores, which must subsequently be summarized into the Refined Score. This calculation must cope with various challenges. First, an equal distribution of FPPs across security findings is unlikely. Findings with a higher Base Score will likely attract more attention due to the higher potential impact on the software. Moreover, the type of input from certain entities is predefined by its characteristics. Threat Modeling, for example, will never reduce the score of a finding, as the activity focuses on identifying risks and attack vectors instead of low-severity/probability attacks. To compensate for these problems, we cluster all scores according to the orientation scale in Table 5.2 and identify the cluster containing the most values. Depending on the relevance of an FPP, the scores are respectively more often considered. Hence, an FFP with a Relevance of 2 will have the respective score added twice. In case of a tie, the cluster with the higher urgency is chosen. The largest cluster and its adjacent ones are selected, and the upper median of all scores comprised by them is calculated. This approach eliminates outliers that falsify the prioritization score while respecting the input of each entity. To ensure the trustworthiness of the Refined Score for all practitioners utilizing it, it is accompanied by an explanation listing all FPPs considered for calculation. This solution approach can be fully automated, excluding the creation of FPPs, which must be manually added.

Figure 5.5 shows the process for calculating the Refined Score. For an example calculation, let's consider the following Base Score for a vulnerability in a third-party component:

```
(75, "Linear mapping of the CVSS Score from CVE")
```

Based on this Base Score, we further assume that the following three FPPs apply to the findings:

```
{
"Provider" : "Threat Model",
"Policy Explanation" : "Vulnerability in exposed component",
"Action" : "+10", "Relevance" : "1", "Rule": {...}},
{
"Provider" : "Developer",
"Policy Explanation" : "The component will not be addressed in this sprint",
"Action" : "60", "Relevance" : "2", "Rule": {...}},
{
"Provider" : "Stakeholder",
"Policy Explanation" : "Security is currently of no concern",
"Action" : "10", "Relevance" : "2", "Rule": {...}}
```

Applying the Action to the Base Score results in a list of values containing the elements 10, 10, 60, 60, 85. Only the last value is directly derived from the Base Score by adding ten points to the score of 75. The values 10 and 60 are mentioned twice due to the relevance of the respective FPPs. These values are clustered into the buckets [10, 10],[∅],[60,60],[85] and [∅]. The two largest clusters are the "None" and "Medium" urgency clusters. As defined, we select the cluster representing more urgent ratings in case of a tie. In combination with the adjacent clusters ([60, 60, 85]), we derive the upper median value resulting in a Refined Score of 60.

```
{
"Refined Score" : 60,
"Refined Explanation" : {
    "Policy Scores": [60, 85],
    "Considered Scores": [
    (60, "Developer: The component will not be addressed in this sprint", 2),
    (+10, "Threat Model: Vulnerability in exposed component", 1)]}
}
```

The Refined Score is accompanied by an explanation, ensuring transparency about the construction of the score. Therefore, it lists all scores considered during the calculation of the upper median with data from their respective FPPs. This data includes the Provider, the Policy Explanation, the Action, and the Relevance as background information on the validity of the generated score.

Integration Process

In contrast to the calculation of the Base Score, the Refined Score requires no additional models for its integration into projects. However, creating FPPs upfront for known processes or impact factors can be beneficial. Moreover, the orientation scale in Table 5.2 must be distributed to all entities providing input to ensure that security findings are prioritized on the same scale.

5.5.3 Finding Response Planning

Ultimately, the information accumulated during the analysis of each security finding must be summarized in a responding action. While some decisions seem obvious if previous activities identified certain facts, others require further investigations and consultation within the project team. The correct response to a security finding is essential for the successful management of security findings, as otherwise, not managed security risk or an overwhelming amount of security improvement tasks with little impact on the product's security infringe the success of product development.

Therefore, practitioners are challenged with selecting appropriate responses to security findings based on the limited information available. Regardless of whether the project team decides to respond to a finding or not, an inherent decision is taken as security findings pose a risk to the security of the product and all its users. Hence, the planning of a finding response in industry is closely related to the risk management for industrial products. However, this introduces the challenge that risk management and its underlying strategy and activities are industry-, sector-, and company-dependent. Therefore, we cannot assume a particular strategy for our solution approach but must ensure customizability and process independence.

Our solution approach utilizes all insights gained during previous activities of the security findings management and guides the project team to appropriate decisions and finding responses. Fueled by these insights, our strategy resembles a decision tree, proposing response actions depending on the characteristics and circumstances of each finding. As part of our proposed methodology, this approach must be employed for each security finding identified. Figure 5.6 shows a visual representation of this strategy.

During the first stage, we utilize the knowledge acquired during the *History Tracking* and check whether the finding is currently absent. If the finding has not been found recently, we must assume that the underlying problem was either solved or cannot be identified again by any security activity. Hence, we assume that the finding no longer poses a security risk. To verify this assumption, we recommend to confirm the absence of the finding. If its absence can be confirmed, the finding must be considered closed. If the absence cannot be confirmed, it is regarded as a potential security threat.

In the second step, the knowledge acquired during the verification described in Subsection 5.4.1 is employed to analyze the applicability of the security finding. If the finding was tagged as False Positive or Not Applicable, the finding has no impact on the software. Even though the finding will be continuously reported, it either does not affect the software or is incorrectly reported by the security activity. In these cases, no response to the security finding is necessary, and both cases are documented with a respective finding state.

At this point in the decision process we have to assume that the finding exists and impacts the security of the software to some degree. Hence, it poses a threat to the product's security and is, therefore, closely related to the security risk management process for industrial products. To apply the risk assessment process to the decision-making of the security finding response, we map the risk terminology proposed by ISO 27000 to our context [71]. The result and applied examples can be found in Table 5.3.

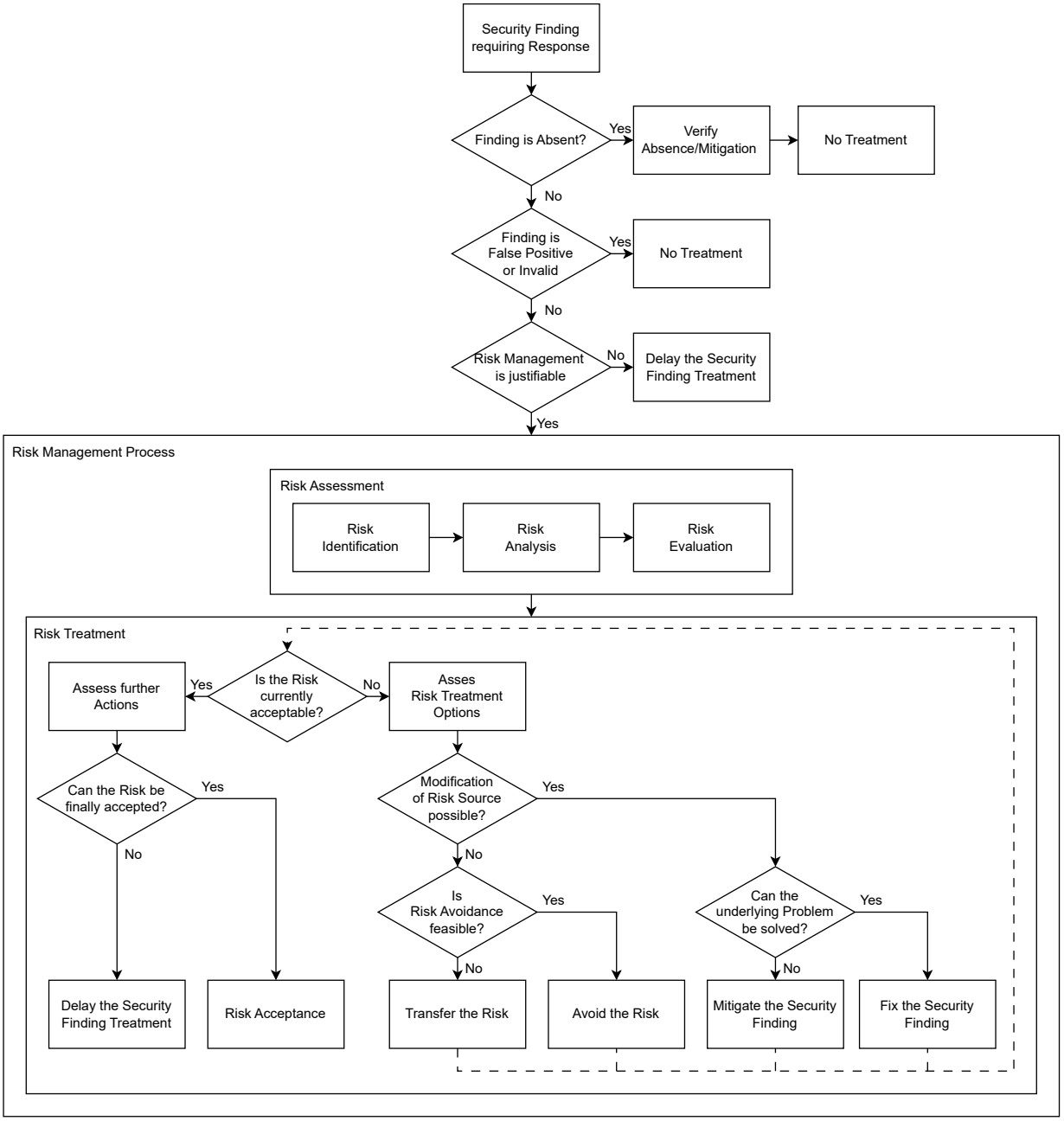

Figure 5.6: Finding Response Planning Strategy as Flow Chart

Term	Definition	Context Example
Consequence	Outcome of an Event, affecting Objectives	Data is disclosed to the public
Control	Measure, modifying Risk	Encrypt data storage
Control Objective	Goal that should be achieved by implementing a Control	Stored data can not be read
Event/Incident	Occurrence of circumstances	Malicious entity can access confidential data
Level of Risk	Magnitude of Risk expressed as a combination of Consequence and its Likelihood	Critical
Likelihood	Chance of something happening	Very likely
Objective	Result to be achieved and typically set by the organization or business	Data may not be disclosed to the public
Residual Risk	Remaining Risk after Risk Treatment	It is unlikely that a limited amount of data might be disclosed to the public
Risk	Effect of uncertainty on Objectives	It is very likely that data might be disclosed to the public
Risk Acceptance	Informed decision to take a Risk	The residual risk is acceptable to the organization
Risk Analysis	Process to comprehend a Risks nature and determine the Level of Risk	Process
Risk Assessment	Process of Risk Identification, Risk Analysis, and Risk Evaluation	Process
Risk Communication and Consultation	Process to provide, share or obtain information	Process
Risk Criteria	Terms of reference against which the significance of Risk is evaluated	Risk Appetite of Business/Organization
Risk Evaluation	Comparing the results of the Risk Analysis with Risk Criteria to determine whether the Risk and is acceptable	Process

Risk Identification	Finding Risk Sources and their Events, recognizing and describing Risks	Process
Risk Management Process	Process to establish the context of Risk, identifying, analysing, evaluating, treating, monitoring and reviewing the Risk	Process
Risk Source	Element that has the potential to rise to a risk [76]	Weak Database Hardening Measures
Risk Treatment	Process to modify Risk	Process

Table 5.3: Glossary of ISO 27000 Terms in the Security Findings Management Context

These terminologies and the proposed processes slightly differentiate between standards and guidelines applicable in industry. However, the fundamental idea of how to manage security risk and its activities in the context of security findings management persists. Moreover, we assume that the organization or business developing a product already obeys some sort of risk management process that our solution approach must follow. The solution approach follows this constraint by adhering to the concepts proposed in industry standards while respecting the need for customization.

The next step in our process is deciding whether the risk will be managed at all or preliminarily discarded. The risk management process requires time, effort, and information from various stakeholders to determine a reliable level of risk and decide on appropriate risk treatments. Since the impact of some security findings on the product might be negligible, the project team could choose to avoid the risk management process and the associated investments. Similar to most other activities, the conclusion to conduct the risk management process depends on the input from the multidisciplinary project team. The discussions and all subsequent activities for planning a finding response are supported by the insights identified during the *Investigation of Security Findings*. As the decision process depends on the project and organization, our solution approach proposes no particular structure but solely supports the documentation and continuous review of the decision. If the risk management process is not justifiable, this decision is documented with a respective validity period.

Otherwise, the risk management process for security findings is started. General guidance recommends to start by collecting information relevant to the risk management process and the identification of risk [74, 25, 130]. However, as part of the security findings management, we already know the risk source: the security finding itself. Therefore, solely the risk and its respective event must be described to complete the *Risk Identification*. Afterward, this newly found risk is analyzed by combining the impact or consequence of the risk with its likelihood to determine the level of risk following the risk process of the organization (*Risk Analysis*). Afterward, the *Risk Evaluation* takes place by comparing the level of risk with the risk criteria to determine whether the risk is acceptable to the business or organization. However, the exact procedures for assessing the security risk

depend on the organization. How risks must be described, the metrics for consequence and likelihood, and the representation of risk criteria all rely on the organizational risk management process. Therefore, we consider this activity as a black box in our solution approach, which is given all information from preceding activities and outputs the decision of whether the current level of risk is acceptable.

After the risk has been assessed, the risk treatment must be conducted, depending on whether the current level of risk is acceptable or not. If the risk is not acceptable, options for the risk treatment must be reviewed. These options include the *Risk Avoidance*, by avoiding the condition that gives rise to the risk, the *Risk Transfer*, by sharing the risk with other parties, and the *Risk Modification*, by introducing or altering controls as defined in ISO 27005. Applied to the management of security findings, this implies that either the software use case or scope must be changed (Risk Avoidance), a third party like an insurance company must take responsibility for the risk (Risk Transfer), or the finding itself must be fixed or mitigated (Risk Modification). In the context of security findings management, the third case differentiates between solving the root cause of a finding and introducing additional controls to reduce the likelihood or consequence of the finding. Minding the case of third-party components, the influence of the project team on certain findings might be limited to an informed decision about replacing components instead of fixing the root cause. Whenever the risk is not entirely eliminated, the remaining risk requires another evaluation against the risk criteria, resulting in another iteration of the risk treatment.

However, if the risk is acceptable, we differentiate between two responses. The straightforward approach is to officially accept the risk and communicate the acceptance to achieve risk transparency. Since this represents a final decision for security risk, it also closes the associated security finding. Another response is to delay the decision for risk acceptance. Even if the risk might be acceptable, treating the security finding by risk avoidance or modification with controls might still be a viable subsequent treatment. Hence, delaying the security finding response and retaining the risk is another outcome of the risk treatment process. The risk is taken in both cases, and no immediate change to the risk source is conducted.

Regardless of the selected finding response, the decision behind it must be communicated, documented, and tracked. As part of the risk management process, the monitoring and periodic review of risks is essential. This affects all risk treatments and both forms of risk acceptance alike. Hence, each decision has a limited applicability that must be reviewed. This validity period differentiates between finding responses, as the remaining risk inherent to certain decisions necessitates a more thorough monitoring. Moreover, all decisions must be communicated to various stakeholders to achieve risk transparency. The communication is addressed in the next chapter.

Integration Process

Certain preconditions must be met to integrate this solution approach for decision-making into modern industrial software development projects. First, the procedures for the established risk assessment process within the organization must be identified to calculate

the risk level in compliance with the guidelines of the company developing the product. Moreover, the risk criteria must be defined by the organization or project to derive when a risk can be accepted without additional measures or controls. Finally, the timeframe for periodically reviewing the finding responses must be specified. This depends again on the corporate risk management strategy and must be determined by the organization's representative in the project.

5.6 Implementation

To ensure that our solution approach can be integrated into actual projects, we must instantiate them in our platform for the management of security findings. We added additional rules and belief classes to the knowledge base presented in Section 3.5 for this instantiation. The steps to implement the proposals of this chapter are described in this section and structured according to the solution approaches.

5.6.1 Implementation of Tracking and Documentation

To track every security finding throughout its lifecycle and document the resulting information, the first solution approach of this chapter tracks each finding's history and current state. This is implemented by two belief classes, one query, and one rule.

A new belief class storing this information is necessary to implement the tracking entry suggested for history tracking. Apart from the regular metastructural details and the reference to the aggregated finding it belongs to, this new belief class contains the entire finding history and the explicit documentation of all conclusions listed in Table 5.1.

```
"Class Name": "Finding History",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Derived",

"Finding Identifier": <>,
"First Found": <Date>,
"Last Found": <Date>,
"Last Change": <Date>,
"Change History": {
                <Date 1>: (<Activity 1>, <ChangeText>),
                <Date 2>: (<Activity 2>, <ChangeText>),
                ...},
"Activity Number": <Number>,
"Activity Statistic": {
                <Activity 1>: <Number>,
                <Activity 2>: <Number>,
                ...}
"Finding History": {
```

```
<Activity 1>: [<Date 1>, <Date 2>],
<Activity 2>: [<Date 2>, <Date 3>],
...}
```

The only exclusion is the "Recently Reported" field. The solution approach proposes that this field contains a list of the last reports that each activity has uploaded. Implementation-wise, this has multiple noteworthy drawbacks. First, having this field in each *Finding History* instance would introduce duplicate data into the knowledge base since each finding reported by the same activity would contain the same information. Second, adding this field would drastically complicate the maintenance of soundness. With each new report of an activity, the history element of all findings identified at some point by this activity would be changed. Therefore, findings that have potentially been solved a long time ago would be recomputed every time a new report is added, thus increasing the workload for its maintenance. Consequently, no information about the latest report of an activity may be stored in the findings history element. Instead, we implemented a new query, named *History Query*, that returns the *Finding History* instance with the latest reports of an activity. Moreover, any project-tailored tracking information is also implemented in this query. This query returns the *Finding History* element for the identifier of a given aggregated security finding and adds the "Recently Reported" field by taking the most recent report date for all activities that reported the finding.

The content of each *Finding History* belief instance is computed by an inference rule, which takes one instance of the "Aggregated Security Finding" class to derive its history. The rule consists of the following 4-tuple:

```
{"Trigger": ["Aggregated Security Finding"],
"Code": "<Reference to the File>",
"Output": "Finding History", "Name": "History Tracking"}
```

The code itself iterates through the list fields of the aggregated security finding and constructs the "Finding History" field. From this field, all other elements can be derived.

To implement the state machine model of the security finding lifecycle, we introduce a new belief class, where project team members can upload a new status. This new belief class is depicted below:

```
"Class Name": "Status Input",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Explicit",

"Finding Identifier": <>,
"Location": <>,
"User": <User Email>,
"Added At": <Date>,
"Status": <Open/Fixed/False Positive/Invalid>,
"Valid Until": <Date>,
"Description": <String>
```

In addition to the meta-structural information and the reference to the aggregated finding it belongs to, this new belief class contains information about the user and timestamp that added the status information. Moreover, it comprises the new status, a timeframe of validity for the status, and a description of how it was achieved. This description maps to the alphabet of the state machine. Since a security finding might be located at different places in the product and the status could differentiate between locations, the affected location is also documented. To add this information to the knowledge base, the */userinput* resource of the RESTful API is employed. In our implementation, we avoided the "Absent" status, as this is intrinsically given by the history tracking in combination with the status history for the purpose of efficient maintenance. Both are again implemented by a query, identifying the most recent and, hence, active status for an aggregated security finding. This *Status Query* collects the aggregated security finding and all instances of the *Status Input* belief class belonging to it. Afterward, it analyzes whether any finding location is "Absent" by checking whether the most recent report of any activity found it and lists all status changes that belong to each location to identify the currently active status. In the following, we describe an exemplary answer to the query:

```
"Finding Identifier": <>,
"Active Status": {
                "Location 1": "Absent",
                "Location 2": "False Positive"},
"Status History": {
    "Location 1": [
        {"User": "system", "Status": "Open", "Added At": 01.01.2020,
        "Valid Until": 01.01.3000,
        "Description": "Findings are Open by default"},
        {"User": "system", "Status": "Absent", "Added At": 07.01.2020,
        "Valid Until": 01.01.3000,
        "Description": "Findings are Absent when not found anymore"}],
    "Location 2": [
        {"User": "system", "Status": "Open", "Added At": 01.01.2020,
        "Valid Until": 01.01.3000,
        "Description": "Findings are Open by default"},
        {"User": "user@team", "Status": "False Positive",
        "Added At": 10.01.2020, "Valid Until": 10.06.2020,
        "Description": "Investigation showed finding does not exist"}]
}
```

Therefore, the information on the currently applicable status of a finding is not stored in a separate belief class but constructed by a query.

Finally, the maintenance of protection goals has already been implemented for the evaluation during the previous chapter. Therefore, no implementation effort was necessary for this solution approach.

5.6.2 Implementation of the Analysis

To supplement and document the analysis of security findings, our solution approach primarily focuses on documenting the analysis results. For our platform to support these requirements, we adapted the existing *Status Input* belief class and added a belief class for analysis input.

To support the documentation of each finding's validity, we adapt the existing *Status Input* class to reflect the conclusions of the solution approach. This implies that the existing state types were extended by the "Not Applicable" and "Applicable" status, while the "Invalid" state was removed. Since the belief class already contained a description field, the explanation of how it has been validated can already be documented. Moreover, the review period is implemented by utilizing the "Valid Until" field. Using this field, the status will only be valid for a certain period until it returns to the "Open" status. This solution approach can be implemented into the security findings management platform by changing the accepted status entries.

However, we require an additional belief class to document the results of any generic security findings analysis. As discussed in the solution approach, the type of investigations that must be supported is arbitrary. Therefore, the belief class to document them must be similarly customizable.

```
"Class Name": "Analysis Input",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Explicit",

"Finding Identifier": <>,
"Title":<>,
"Analysis Description": <>,
"Analysis Result": <>
```

In addition to the meta-structural information, the belief class contains a reference to the investigated finding. Moreover, all three fields of the solution approach are added, covering a short title, a detailed description of the performed investigation and the result of the analysis. The */userinput* resource of the RESTful API is again employed to upload the analysis results.

5.6.3 Implementation of Prioritization and Decision Making

To finally prioritize each security finding and support the subsequent decision-making process, our solution approach constructs the Base Score and Refined Score for every security finding and documents the results of the decision-making process. This is implemented by two new belief classes, one query, and one rule.

The first component of the implementation is an additional belief class, documenting the Base Score and the Refined Score with their respective explanations. Apart from the regular meta-structural information and the reference to the aggregated finding it belongs

to, this new belief class contains the Base and Refined Score as numeric values and the explanations for both.

```
"Class Name": "Prioritization Information",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Derived",

"Finding Identifier": <>,
"Base Score": <Integer>,
"Base Score Explanation": <>,
"Refined Score": <Integer>,
"Refined Score Explanation": <>
```

To document and utilize the FPPs, a second new belief class must be established to collect the prioritization input from knowledge base external entities. In addition to the meta-structural information, each instance of this belief class contains the five components of any FPP, including the Provider, Policy Explanation, Rule, and Action.

```
"Class Name": "Prioritization Input",
"Identifier": <>,
"Content Hash": <>,
"Belief Type": "Explicit",

"Provider": <>,
"Policy Explanation":<>,
"Rule": <>,
"Action": <>,
"Validity": <Date>
```

In our implementation, the Provider and Policy Explanation are both text fields. The Action field can contain a numeric value between 1 and 100 or a mathematical operation to add, delete, multiply, or divide a number from/to the Base Score. The validity can be tested with the following regular expression:

```
^([\+\-\*\/]){0,1}[0-9][0-9]{0,2}(?:,[0-9]{0,3}){0,1}$
```

To define the findings affected by the policy, the Rule field defines the characteristics of all affected findings. This is implemented as an Elasticsearch query that can be executed by the rules calculating the Refined Score. Finally, the Validity field can contain a date until which the FPP is valid. Influence factors that solely temporarily impact the priority of a finding, like sprint planning, can be realized by using this field. To add this external input to the knowledge base, the */userinput* resource of the RESTful API is employed. As per definition, each FPP additionally contains a Relevance field. Based on the insights of the preliminary evaluation in this chapter, it was removed from the implementation.

In addition to the new belief classes, a new inference rule is required to derive the prioritization information from the aggregated security findings and the user input. Similar to

the implementation of the enrichment operation, the rule 4-tuple consists of two different belief classes, triggering the execution of the code. Its trigger is either an aggregated security finding or external prioritization input and results in an instance of the Prioritization Information class belonging to the respective aggregated security finding.

```
{"Trigger": ["Aggregated Security Finding", "Prioritization Input"],
"Code": "<Reference to the File>",
"Output": "Prioritization Information", "Name": "Prioritization"}
```

As the calculation of the Base Score may utilize a different semantic model for each combination of source and version, the code file follows a strategy similar to the parsing approach.

```
if aggregated_finding.Source = "A" and aggregated_finding.Version < "1":
    <Apply Model 1>
elif aggregated_finding.Source = "A" and aggregated_finding.Version >= "1":
    <Apply Model 2>
elif aggregated_finding.Source = "B" and aggregated_finding.Version >= "0":
    <Apply Model 3>
<Calculate Refined Score>
```

Subsection 5.5.1 presents an example of such a model. After the Base Score and its respective explanation are calculated, the Refined Score is constructed from the user input and the Base Score value.

To support our methodology's final decision-making process, the implementation must be able to document the outcome. As discussed in Subsection 5.5.3, the decision-making strongly depends on the processes established at the software engineering organization. Hence, the implementation focuses on documenting and tracking the decided response.

To track the decisions taken, the belief class *Status Input*, described above, must be extended to reflect the additional states. Currently, solely the states *False Positive*, *Not Applicable*, *Closed*, and *Open* are implemented. Moreover, the *Absent* state is implemented using a query. In addition, we define the states *Risk Accepted*, *Risk Transferred*, *Risk Avoided*, *Fixed*, and *Mitigated* as new acceptable states. However, this still lacks the opportunity to document that a finding has been analyzed, but the risk management is either not justifiable or the risk is not accepted yet. Suppose the risk management cannot be considered reasonable. In that case, this information is added as an instance of the *Analysis Input*, presented earlier in this section, and the respective finding state is changed to *Closed*. If the risk to the product is acceptable but not accepted yet, this is represented by the state *Open* with a respective explanation. The review period defined during the methodology integration is implemented using the *Valid Until* field of each instance of the *Status Input*.

Finally, another query must be implemented, supporting the project team during the decision process. This new *Summary* query summarizes all information available for a particular security finding. Hence, for one instance of the *Aggregated Security Finding* it collects all instances of *Analysis Input*, *Enrichment Information*, and *Prioritization Information* belonging to it. Moreover, the existing *History Query* and *Status Query* are

utilized to collect the state and history information. The response to this query from the knowledge base consolidates this information.

5.7 Preliminary Evaluation

Following the research methodology of this thesis, the solution approaches in this chapter must be preliminarily evaluated against their initial requirements. In this section, we discuss the evaluation for each initial requirement and finally present the evaluation of the prioritization approach, published in [196].

5.7.1 Evaluation Scope

The challenge this chapter aims to solve is the analysis and tracking of security findings according to modern industrial software development principles to achieve a satisfactory finding response. This challenge is again split into the activities that must be performed during the security findings management and the principles that must be adhered to in modern industrial software development projects.

We identified eleven functional requirements, summarized into seven clusters of similar problems. The most significant contribution of this chapter is the security findings prioritization approach. Therefore, we focus on this aspect of the solution approach in the preliminary evaluation. For all other areas, we target a functional assessment of the implementation rather than measuring its impact on modern industrial software development projects. Initially, we identified seven principles that the solution approach in this chapter must adhere to. Four of these principles, namely the reduction of manual effort (Topic III), a fast response time (Topic IV), the supply of necessary information (Topic V), and the enablement of cross-disciplinary collaboration (Topic IX) affect all of our solution approaches. The remaining three principles affect particular solution approaches exclusively, including the collection of business-relevant indicators (Topic I), the decision-making with a customer-focused mindset (Topic X), and the minimization work packages by decisions (Topic II). In addition to the functional constraints, the evaluation also aims to assess the fulfillment of these principles.

The first solution approach for the documentation and tracking of security findings utilizes the information from previous stages to document the history and state of each security finding. It exclusively supports practitioners by providing information about the security findings that must otherwise be manually derived and by documenting the results of activities that practitioners conducted. Hence, the evaluation of the proposed solution focuses on the fulfillment of the requirements and the adherence to the development principles. To analyze these factors, data from modern industrial software development must be supplied, and the ability to correctly track the status and history of each finding and its documentation of data must be assessed. Moreover, the fulfillment of principles number I, III, IV, V, and IX are analyzed, based on the supplied data.

The second solution approach proposed in this chapter is the analysis of security findings. As part of our proposed methodology, practitioners assess security findings for their validity, identify the root cause, and classify them. The implementation of our methodology solely supports practitioners during these activities by supplying information about the security findings and documenting the results. Since the investigation of security findings is highly project-dependent and our methodology solely supports practitioners during their work, we focus on the functional aspects of the solution approach once again. Hence, we integrate the solution approach in a modern industrial software development project, and the ability to analyze security findings according to the requirements is analyzed. Moreover, the applicable principles III, IV, V, and IX are assessed for their fulfillment.

The last solution approach proposed in this chapter is the prioritization of and response to security findings. Since the methodology for prioritizing security findings is this chapter's main contribution, the evaluation focuses on this aspect. In addition to assessing the fulfillment of requirements and principles, the approach's potential for industrial practice must be analyzed. This implies that not only the objective fulfillment of requirements and the principles II, III, IV, V, IX, and X must be assessed, but also the subjective perceptions of practitioners must be collected and reviewed.

5.7.2 Evaluation Design

In the evaluation, we want to investigate three aspects: do our solution approaches fulfill its process requirements, do they adhere to the principles of modern industrial software development, and how is the potential of the prioritization approach perceived by industrial practitioners? To conduct the evaluation, we decided in favor of a lightweight qualitative study consisting of interviews and an integration of the solution approaches in modern industrial development projects. To design and conduct this study, we follow the current State-of-the-Art for empirical studies in the software security domain [164, 138, 184] minding known challenges [204].

Based on the goals of the evaluation, we formulate the following research questions:
RQ1*: Is each solution approach covering the respective requirement of the security findings management process?*
RQ2*: Is each solution approach adhering to the applicable principles of modern industrial software development?*
RQ3*: Does our solution approach for security findings prioritization represent the finding prioritization process in industrial development projects correctly and completely?*

We split our study into two separate phases to analyze the impact of our solution approaches on industrial practice and receive additional input on the prioritization of security findings in practice. In the first phase, interviews with professionals for security findings management in modern industrial software development projects are conducted to analyze the theoretical potential of the prioritization approach. During the second phase, the solution approaches proposed in this chapter are integrated into two industrial software development projects, and the findings prioritization is executed in parallel to the findings

prioritization process existing in the project.

For the first phase, we collaborated with our industry partner to select ten practitioners with expertise in software security and experience with modern industrial software development projects. To fulfill these criteria, we solely contacted security professionals who have worked in software development projects following Agile or DevOps principles as security responsible. The one-hour-long interviews were conducted virtually in an isolated environment, with one subject and one interviewer as participants. Each interview consisted of five stages listed below:

1. Introduction to the topic of security feedback management
2. Self-assessment of the subject
3. Preliminary question
4. Presentation of methodology
5. Questionnaire

It started with an introduction to the overall topic and a self-assessment of the subject's skills in Agile, DevOps, and security findings management on a scale from 1-Beginner to 5-Expert. The interview continued with warmup questions to ensure a correct understanding of the subsequent topics and an introduction to the security findings prioritization approach. Finally, feedback on this approach was requested in the last stage using a questionnaire. The interviewer documented and answered potential questions by the subject during the interview. All stages were aided by a slide deck, questionnaire, and interview guide, which can be found in the related material of the published paper [191], accompanied by the documented questions.

In the second evaluation phase, we integrated the implementation of our solution approaches into two ongoing industrial software development projects at our partner. We exclusively selected projects that follow modern software development principles, utilize automated security checks, and have an established strategy for managing security findings. This phase aims to compare the established strategy with our solution approach for security findings management and analyze the functionality of all other solution approaches proposed in this chapter. Using the implementation of the security findings management platform, the evaluation started by establishing the security report collection in each project. This included, in particular, the creation of parsers and semantic models to customize the security findings management platform for each project. As the next step, the development lead of the respective project is introduced to our proposed approach by employing the same slide deck used in the first phase. Afterward, the study protocol is started. Each week, a virtual interview with the development lead is conducted. During these meetings, the development lead presented the results of the existing findings prioritization process in the project and elaborated on all influence factors impacting the findings' priority. Next, the interviewer introduced the results of the automated prioritization approach, and the differences between both processes were identified and explained.

Concluding the interview, the current status, all differences and their root causes are documented by the interviewer in a template. Afterward, the interviewer developed findings prioritization policies based on the influence factors presented by the development lead to support the solution approach under evaluation in covering the current project influences. Moreover, the results provided by each solution approach, including tracking and documentation, analysis, and prioritization and response were analyzed, including the fulfillment of the development principles. Whenever data was incorrect or an activity was not supported, the shortcoming was documented. Whenever actions could not be evaluated due to a lack of data, this circumstance was also noted. In one final interview, the development lead was asked to rate the perceived usefulness of the approach, state identified shortcomings, and inform about any final remarks. The documents and templates can again be found in the related material of the published paper [191].

Between both subject groups, no overlap existed. Moreover, the subjects did not interact with the actual implementation of the system to avoid a falsified response on the findings prioritization process, as the user interface and implementation might introduce bias. While the first two research questions are answered based on the number of shortcomings, the third is assessed based on the subjects' responses.

5.7.3 Evaluation Results

After both phases of the study concluded, two researchers analyzed the results. These results are presented below, structured according to the phases in which they have been acquired.

During the first phase, our subjects provided 41 responses to the preliminary question. These can be summarized into the following list of factors impacting the prioritization of security findings:

- Threat Modeling and Risk Analysis
- Fixture Prerequisites
- Finding Severity
- User Input
- CVSS
- Low Hanging Fruits / Quick Wins
- Others

The responses to interview questions one, two, and three are categorized into positive, neutral, or negative responses. The resulting statistics are depicted in Table 5.4.

Question	Negative Answers	Neutral Answers	Positive Answers
Can all Aspects be covered and is Prioritization correctly represented	0	2	8
Reduction in Work Effort	0	1	9
Understandable for Team	0	2	8

Table 5.4: Answers to the Interview Questions 1-3 of the Prioritization Evaluation

Moreover, the subjects mentioned that the answers are only valid under the assumption that the project team actively uses the system by providing input and that a fundamental security awareness exists in the project. As potential shortcomings and challenges of the prioritization approach, the subjects mentioned the additional effort for adding input, the complexity of the Base Score model, and the dependency on an intuitive user interface.

The study's second phase concluded after seven meetings with the development lead of both projects. During the study timeframe, each activity produced up to six reports per week, containing mainly security findings with a low or informational severity level. In both projects, the result of the prioritization in advance of the first interview iteration partially differentiated from our proposed solution approach. Both projects contained a list of unaddressed findings that were not planned for mitigation. This was due to the release schedule in one project (vulnerability in dependencies) and the affected component that was not used yet in the other. These discrepancies have been resolved in the second meeting by respective findings prioritization policies. During the evaluation timeframe, this phenomenon occurred three times in the first project and 12 times in the second project and has consistently been resolved by additional findings prioritization policies. An example was a new finding being prioritized lower by our approach compared to the project's priority. Based on the protocol, the team prioritized it higher, as it seemed like a quick win and was contained in a component currently under construction. Consequently, the effort to address it was low enough to become a top priority for the team. The final interview results showed that both development leads perceived our prioritization solution approach as useful and wanted to continue the usage beyond the evaluation period. The reasons mentioned by both subjects were the documentation of the prioritization reason and the common Base Score provided across sources. As a shortcoming, the effort and knowledge necessary to create prioritization policies were commonly mentioned. The results of this part of the evaluation can again be found in the related material of the published paper [191].

Moreover, each solution approach has been tested for its functional correctness and principle fulfillment. From a functional point of view, the history tracking, documentation, and state tracking consistently delivered correct results. Moreover, the states suggested by the development lead for specific security findings were covered by our methodology. In particular, the importance of temporary states was reinforced during the interviews. The ability to add results from the analysis of security findings was also verified. It was impossible to

confirm nor deny that this solution approach supports practitioners sufficiently due to the design of the evaluation. Finally, the functionality of the prioritization and response approach has also shown correct functionality. Once again, the importance of time-restricted states and prioritization policies have been identified. Since the risk management in both projects has been conducted inherently by team members without explicit process, the risk analysis could not be assessed. In addition to the functional constraints, the adherence to the principles of modern software development has been assessed. The results of the analysis can be found in Table 5.5, where an "X" indicates verified principles, an "F" stands for an unfulfilled principle, a "N/A" shows that a principle is not applicable and a "I" informs, that the assessment of the principle was insufficient. The description of each topic can be found in Section 5.1.

Solution Approach	**I**	**II**	**III**	**IV**	**V**	**IX**	**X**
Tracking and Documentation	X	N/A	X	X	I	X	N/A
Analysis	N/A	N/A	X	X	I	X	N/A
Prioritization and Response	N/A	X	X	X	X	X	X

Table 5.5: Development Principles Fulfillment for the Analysis and Tracking

5.7.4 Discussion

Finally, we discuss the evaluation results and derive answers to our research questions. The discussions are structured according to the research questions in the following paragraphs.

RQ1 - Requirements Fulfillment

The first research question addresses the fulfillment of the requirements for each solution approach. During the study with data from industrial practice, all solution approaches correctly functioned and supported the security findings management process as defined by our research. Therefore, they fulfill the fundamental requirements. However, this result cannot be generalized due to the study design. Since no project team members interacted with the actual implementation, it cannot be assessed whether it supports all activities in practice and is, therefore, beneficial to modern industrial software development. Hence, the solution approaches cover our interpretation of the requirement, which might differ from the process within a particular project or industrial practice in general.

RQ2 - Principle Compliance

The second research question aims to assess compliance with modern software development principles. The results of our evaluation showed that the principles III and IV are fulfilled for all solution approaches due to the automation of the methodology in a semantic knowledge base. Even though certain tasks must still be conducted manually, repetitive tasks like documentation are fully automated. Moreover, receiving information about security findings can also be considered as fast, as queries to the knowledge base are answered consistently in under one second, with the processing of new reports within

2 minutes. As each manual activity has a set of preceding queries that supply all information available about a particular finding, we further consider principle V as fulfilled. As each solution approach requires input from project team members, the cross-disciplinary team structure and need for collaboration must be minded, as mentioned in principle IX. During the evaluation, particularly, the prioritization minded these principles. For both other solution approaches, we can currently neither confirm nor deny that they encourage cross-disciplinary collaboration, as this requires a cross-disciplinary project team working with the solution approach to measure the impact. According to principle X, each decision must be taken with the customer in mind, making it applicable to the solution approach addressing the finding response. Even though this strongly depends on the project team, the adherence to common risk management standards ensures that the customer is considered as a factor. Similarly, principle II, which requests that work packages for each iteration should be minimized, is also included when following a risk management approach. During the assessment of risk treatment, the size of work packages for security can also be considered, fulfilling this principle in practice. The last assessed principle that only affects the tracking of security findings is principle I. In the context of security findings management, it implies that to provide information to stakeholders from the business, relevant information must be collected. As the solution approach documents each security finding with all related information and tracks the occurrence and status history of each finding, we consider this principle fulfilled.

RQ3 - Correct and Complete Security Findings Prioritization

Finally, we want to discuss the potential of the security findings prioritization process. Based on the influence factors on the prioritization of security findings mentioned by our subjects, we believe that all of them can be realized with findings prioritization policies, indicating that the overall strategy is valid. Moreover, most subjects in both phases were optimistic about the correctness of the prioritization, the reduction of effort in the project, and the transparency of the prioritization score. This applied in particular to the approach's usefulness for both projects in the study's second phase. However, a major drawback of the process mentioned in both phases is the additional effort necessary to create findings prioritization policies. Subjects mentioned concerns about the acceptance of this new approach in projects if it implies additional work. This acceptance is crucial for the success of the process in industrial software development projects. Hence, the automated creation of findings prioritization policies, as recommended by the development leads during the study, is a potential future work in this domain. Another lesson learned from the implementation is the low importance of the *Relevance* field in findings prioritization policies. Neither project saw the necessity to adapt this to represent the team's influence distribution correctly. While expected to be important in practice, this proved incorrect and is therefore removed from the implementation. We conclude that our modular approach of adding formalized findings prioritization policies to prioritize security findings covers the necessary input factors assuming that a project team contributes to the process. The process is considered complete, correct, and beneficial by the interviewed subjects.

Threats to Validity

Even though the results of this preliminary evaluation indicate the correctness of the solution approaches and adherence to modern software development principles, their validity is threatened by multiple concerns. The validity of the overall evaluation is constrained by its circumstances. We cannot state any significant claims about its usefulness based on the limited number of participants and the short timeframe of just seven weeks. Moreover, the evaluation of the requirements and development principles only provides weak evidence for the usefulness of most solution approaches. As the evaluation focused on the prioritization process, the remaining solution approaches have only been covered superficially. In particular, the evaluation criteria represent a threat to the construct validity, as the researcher who designed the metric to measure the fulfillment of requirements also designed the solution approach based on the same requirements. Hence, another researcher might consider other criteria as relevant. However, this solely represents a preliminary evaluation of the solution approaches. Minding that these are evaluated later in comprehensive manner in combination with other solution approaches, we consider the current evaluation as sufficient.

5.8 Conclusion

In this chapter, we presented our process for the analysis and tracking of security findings and its integration into the existing security findings management platform. The proposed approach tracks and documents security findings throughout the lifecycle, supports the analysis of security findings, and prioritizes and guides the finding response. This chapter comprises mostly manual activities of the security findings management process, which are supported and guided by our methodology.

This chapter makes three contributions to the current state of the art. First, a state model for the security findings lifecycle is presented as part of the status tracking. Second, a guidance for responding to security findings aligned with common risk management standards has been developed. Finally, the main contribution of this chapter is the methodology for prioritizing security findings in modern industrial software engineering. Moreover, multiple smaller contributions towards a security findings management methodology have been made. All these contributions have been preliminarily evaluated and partially published.

We conclude that the solution approaches for the analysis and tracking of security findings proposed in this chapter support the security findings management activities demanded by the current state of practice while conforming to modern industrial software development principles. Due to the number of manual actions to be performed during this part of our methodology, its usefulness and benefit to modern industrial software development projects requires a subsequent evaluation with an adapted scope.

Chapter 6

Security Findings Feedback Communication

The last objective of the methodology is the communication of knowledge acquired during the management of security findings to all project stakeholders. The communication is bi-directional, implying that stakeholders receive information from and supply information to the methodology. Since all preceding solution approaches proposed in this thesis require interfaces to retrieve or provide information, the strategy proposed in this chapter extends across all preceding solution approaches. The communication itself has to supply data to various stakeholders, differentiating in their demand for information, interest in the project, and organizational hierarchies. Examples include project team members, externals like Penetration Testers, or processes like a top N finding list maintained at an enterprise level.

This chapter presents the challenges for communicating security findings data and describes our proposed communication strategy. Moreover, we demonstrate the implementation of this strategy and its integration with the remaining solution approaches. This chapter contains no preliminary evaluation as the foreseeable threats to the internal validity by solely looking at the communication itself diminishes the impact of an evaluation. Instead, the communication strategy is evaluated in the next chapter with the remaining solution approaches as a comprehensive system.

6.1 Problem Description

Our preliminary problem analysis identified that multiple processes or activities rely on information provided by the security findings management process while not directly belonging to it. Moreover, all preceding solution approaches require interfaces to communicate and consume data from external entities. Subsection 2.3.3 defines that the requirements on "Feedback Processing" and "Process Response" are comprised by our last objective and hence addressed in this chapter. The processing of feedback from the security findings management process comprises the following 16 requirements:

1. Feedback Loop to the responsible for decision making (8)
2. Report risks and findings securely to the appropriate party/stakeholder (9)
3. Report the findings management results to enable the disclosure of vulnerabilities (12)
4. Provide data to the finding response process (13)
5. Share the security findings with other projects (17)
6. Update elements of the software development lifecycle (18)
7. Report known security findings in the system documentation (21)
8. Monitor the success of the security findings management process (25)
9. Report the history of the finding to derive a training curriculum (26)
10. Report security findings to improve secure design patterns (27)
11. Loop security findings to validate assurance cases (30)
12. Share security findings from operations stages with the Q&A team (31)
13. Provide known security findings to the Pentesting team (32)
14. Identify hotspots in the security finding landscape (33)
15. Refine governance policies from security findings information (34)
16. Maintain a list of the most important findings for the company (35)

Moreover, some requirements affect the security findings management process after it has concluded for one particular finding. These include:

1. Solve all findings in a timely manner (4)
2. Review code for instances of similar findings (10)
3. Verify the security finding fix (19)
4. Review all unsolved/unfixed findings (24)

The numbers depicted after each entry represent the identifier of the overall requirements list in Table 2.1.

While all requirements mentioned above are related to or require the communication of information acquired during the security findings management process, their goals, targets, and required information differ. Moreover, any solution approach for the requirements above must mind the principles arising from modern software development:

- Topic I: Any measuring or tracking activity must collect business relevant indicators
- Topic III: The manual effort for the communication must be minimized
- Topic IV: The communication must provide fast results
- Topic V: The communication must supply the actionable information
- Topic IX: Any manual activity must mind the cross-disciplinary team structure and enable collaboration
- Topic X: Any decision must be taken with the customer and its goals in focus

To address the requirements above, the target of this chapter is an aligned communication strategy. This strategy must adhere to the abovementioned principles as a holistic concept rather than have single components of the communication strategy adhering to certain principles.

Organizational Improvement

The first cluster of requirements aims to improve aspects on an organizational level, including the software development lifecycle, the training curriculum, other ongoing projects, secure design patterns, governance policies, and the construction of a common list of the most important findings for the enterprise. These improvement activities are de-coupled from the security findings management process and conducted on a higher organizational level than the original project.

Internal Subsequent Processes

Another cluster comprises all requirements that represent activities in the project, aiming to contribute, improve, or refine elements of the development lifecycle that are not directly related to the security findings management process. These requirements affect the disclosure of security findings to customers, feedback to the Pentesting and Q&A team, the documentation of known system issues, the creation of a security hotspot map for the project, and the ability to answer assurance cases based on the outcome of the security findings management. While these activities utilize the information provided by the security findings management and are conducted on a project level, they are mostly disjoint from one another.

Security Findings Management Processes

The last cluster comprises all requirements addressing activities in the security findings management process and the communication of information to them. The requirements of this cluster include the general demand to report findings and risks to all appropriate stakeholders, including, in particular, decision-makers and the security finding response process. Moreover, the success of the security findings management shall be monitored,

covering the verification of fixed findings and the finding response time. Finally, the software should be monitored for similar findings, and the currently open findings should be reviewed regularly. The requirements in this cluster either request the reporting of information to activities in the security findings management process or propose additional tasks for the respective project.

6.2 Related Work

The presentation of information by computer systems is a domain-independent challenge. While each domain has its particularities related to underlying data transported or the scope of the data usage, certain challenges persist. Isenberg et al. discussed the area of collaborative visualization, in which the same presentation is interacted with by multiple targets to achieve certain goals [84]. They present challenges and examples for collaborative visualization approaches but lack specific principles that must be followed. Hence, their relevance to our approach is limited. Some of these specific principles are presented by Barth et al. in their publication on the aesthetics of visualization [23]. They discuss the importance of carefully selected visual presentations, contribute fundamental design principles, and present case studies on the key features of visualization. The extension of visualization into the domain of the recipient is suggested by Ebert [43]. In the paper, the author recommends the term perceptualization instead of visualization to ensure the concise transportation of information by employing perceptual human inputs. Even though both papers present valuable insights into the topic of data visualization, they lack the particularities of the security domain to create a reliable communication of security findings.

The importance of visualization in the security domain is not only mentioned in academic literature [56] but reinforced by a conference focusing exclusively on visualization for Cybersecurity, called VizSec [189]. While earlier years of the conference have shown a strong focus on visualizations in the network security domain, publications in recent years have diversified their scope. Amongst others, this trend includes the areas of privacy risks [20], incident reports [57], or Malware [32]. A closely related publication is presented by Schreiber et al. [162], proposing a visualization dashboard for presenting results from multiple static code analysis tools across several projects. They utilize three diagrams, including a timeline of identified findings, a depiction of the severity, and a hierarchy of all projects correlated with the location of findings. Their design provides suggestions for the visual representation of security findings. However, their limited scope (static code analysis) and lack of evaluation necessitate further investigation for our use case. A proposal for the visualization of another security findings type is given by Dennig et al. [39]. They present a tool to audit software development organizations based on their exposure to publicly known vulnerabilities in open-source software. They engineered one visualization diagram of all applicable open-source vulnerabilities with the ability to switch between a repository-centered, library-centered, and bug-centered view. Even though their work seems valuable for exploring vulnerabilities, its scope strongly differs from our use case.

6.3 Communication Strategy

The problem description of this chapter provided insights into the fundamental issue that must be addressed when developing a strategy for communicating security findings: distinct entities require differentiating information from the security findings management process to achieve various goals.

6.3.1 Communication Concept

To develop a common strategy for the communication of security findings data, the different use cases and their particular demands must be identified and met. A common strategy can be established based on their similarities and differences. Based on the applicable requirements, we can deduce that each use case consists of three components using the applicable requirements. The *Data* defines what information must be communicated, ranging from specific information like the accepted security risk to all data available for a particular finding. This is communicated in a second step to a specific *Target* and used to achieve a certain *Goal*. Depending on the requirement, the definition of these components ranges from being well-defined to quite ambiguous.

Starting our analysis with the *Target*, we can fundamentally differentiate between human entities and automated processes requiring information. This introduces the first constraint to our communication strategy, as the information transported must be interpretable by humans and systems alike. While automated systems typically require a well-defined interface providing structured information, humans prefer a more visually oriented approach to communication. Hence, our communication strategy must support efficient communication with other systems and humans. Within these two categories, we can further differentiate between the communication preferences of various recipients. The initial list of requirements specifies 16 different human targets, covering:

- Customer
- Decision Maker
- Project Manager
- Project Team
- Peer Project Teams
- Project Stakeholders
- Documentation Responsible
- Security Architects
- Sales Team
- Quality Assurance Team
- Security Testing Responsible
- Operations Team
- Quality Assurance Responsible
- Software Development Process Responsible
- Education and Enablement Responsible
- Performance Measurement

Depending on the organizational structure and maturity of the project, these targets

can either be part of the project team, an entity on the organizational level, or not existent at all. Moreover, this list just represents a fraction of all stakeholders interacting with the project.

Each of the *Targets* discussed previously requires the information to achieve a certain *Goal*. This *Goal* indicates how each recipient utilizes the information from the security findings management process. As presented in the Problem Description of this chapter, the goals are related to organizational improvement, the management of security findings, or other project-internal processes that are not part of the security findings management. Each of these *Goals* has an inherent need for certain information. This implies the *Data* communicated is coupled to the *Goal* that must be achieved. Our requirements indicate that all information acquired during the security findings management process must be communicated at some point. However, for certain goals, a subset of information suffices to achieve it. Example data includes all open or applicable findings, the history of each finding, or the currently accepted and applicable security risk imposed by the security findings. In theory, this could be achieved by always presenting all information available. However, this necessitates considerable effort from the target to derive the exact information necessary to conduct the task, making it an inefficient communication strategy. Instead, we propose tailored communication presenting solely the data necessary to complete a task. This abstracts the concepts of *Goals*, *Targets*, and *Data* into *Presentations*. A *Presentation* supports a *Target* in achieving a *Goal* by presenting the necessary *Data* in an efficient way that simplifies the activity that is conducted for achieving the goal.

6.3.2 Text-Based Communication Interface

The first group of targets identified were automated processes and software-typed systems that require data. Since these do not benefit from a visual representation of the data, we utilize an API as the underlying solution. Using a RESTful approach for this interface further ensures well-defined constraints for fetching data from the security findings management process.

To fulfill our initial requirements for communicating security findings information, the API must be able to retrieve and supply the information necessary for each target to achieve its goals. Since human targets might also employ automated processes, offering any data *Presentation* exclusively via the visual communication interface is insufficient. Instead, the text-based communication interface must also supply each *Presentation* provided by the visual communication. Table 6.1 lists all information that must be retrieved from the security findings management process and its presentation in text-based and visual form. These are derived from the requirements and structured according to their *Presentation*.

Requirement	**Data**	**Text-Based Presentation**	**Visual Presentation**
8, 9, 13, 17, 18, 32, 34	All Findings	List of all Findings	Common Listing

4, 25, 26, 30	Status and Occurrence History of each Finding	Statistics on the Status and Occurrence History of all Findings	Chart of Statistics
12, 24, 21	All unsolved/accepted open Findings	Filtered List of all Findings	Common Listing with Filter Function
26, 35	Most prevalent Findings	Sorted List of all Findings	Common Listing with Sort Function
9	Existing Risk	Statistics on Risk imposed by Findings	Chart of Findings Risk
10	All verified Findings	Filtered List of all Findings	Common Listing with Filter Function
19	All solved/mitigated Findings	Filtered List of all Findings	Common Listing with Filter Function
27	All Findings from specific Activities	Filtered List of all Findings	Common Listing with Filter Function
31	All Findings from Operations Phase	Filtered List of all Findings	Common Listing with Filter Function
33	Most prevalent affected Locations	Sorted List of Locations and Prevalence	Location Heatmap

Table 6.1: List of Presentation in Visual and Text-based Form

From the ten presentations identified in Table 6.1, seven can be realized in a text-based form by a list of all security findings that can be filtered and sorted. In addition, a list of all currently imposed risks by all applicable findings, a list of the most prevalent locations, and a presentation of all findings histories, including the common averages and min/max values, are necessary to cover the Presentations. Especially in organizational improvement, the requirements often lack precise definitions of what data exactly is required. In these cases, they have been assigned to the presentation of all security findings. This shows the necessity to customize the communication strategy over time, as introducing new processes or specifying existing ones could refine how data must be presented. Furthermore, each project might have different interpretations of what is considered relevant information for particular tasks, necessitating requirements engineering for the sophisticated presentation of information [39]. In summary, this results in four textual presentations.

6.3.3 Visual Communication Interface

In addition to the textual presentation of data, our second target group requires visualization as a preferred communication channel. Hence, the previously identified presentations must be transformed into an efficient visual presentation. As each visual presentation must

also be available in a textual form, the visual communication interface utilizes the text-based communication interface to retrieve data. This ensures that each visual presentation is available in textual form and each presentation contains the same information, regardless of whether it is presented visually or textually.

The first visual presentation is an interactive list of all security findings with the ability to sort, filter, and search. With this presentation, the first seven requirements are covered. In addition to the listing approach, we see the necessity for a visualization, just showing one finding in particular to reduce the effort for providing user input on each security finding. It presents all information of this finding in a structured form with the ability to supply input. The second commonly requested presentation is the depiction of the statistics on the history of findings. It is split into two diagrams to avoid an overly dense visualization of this information. The first shows the currently active state of all security findings, while the second provides insights into the recently changed and newly occurred states. The second visualization further requires an interactive approach to select the timeframe of what is perceived as "recently" [162]. The fifth visual presentation is a bar chart of all currently active findings, grouped by severity (Critical, High, Medium, Low, Info). This presentation must also be interactive to switch between currently imposed risk and risk that has not been addressed yet (Open vs. Accepted Security Findings vs. Both). The last visual presentation shows the most prevalent locations as a heatmap, colored according to the highest severity at each location, with the number of findings per location noted on each element. Analogous to the text-based communication, these visual presentations might be added or adapted based on each project.

Recommendation System

During the analysis of the problem space, we have further identified a potential for using a recommendation system as part of the visual interface. The evidence-based identification of a training curriculum for developers or recommendations for the next tasks gives examples of its applicability. In an initial effort, a recommendation system was designed, and initial recommendations were established. This system is depicted in Figure 6.1. The system retrieves data from the security findings management platform daily and processes it in a flat tabular format. Next, the recommendation engine utilizes policies that define how to derive recommendations and generates a list of daily recommendations for each project. An API can access these daily recommendations. Users of the system can employ the same API to return feedback on the quality of the recommendations to, e.g., exclude certain recommendations from being presented anymore. Our initial recommendations reported potential improvements in the process (Secure Coding Practices, Hardening, etc.) to the project team and the recommendation of efficient next tasks based on what has already been solved by the user.

Even though this initial investigation has shown potential, it also requires significant additional research and multiple iterations of feedback with industrial practitioners to be beneficial. Since this exceeds the initial scope of the thesis, we decided against further investigation. Instead, we conclude that a recommendation system for communicating

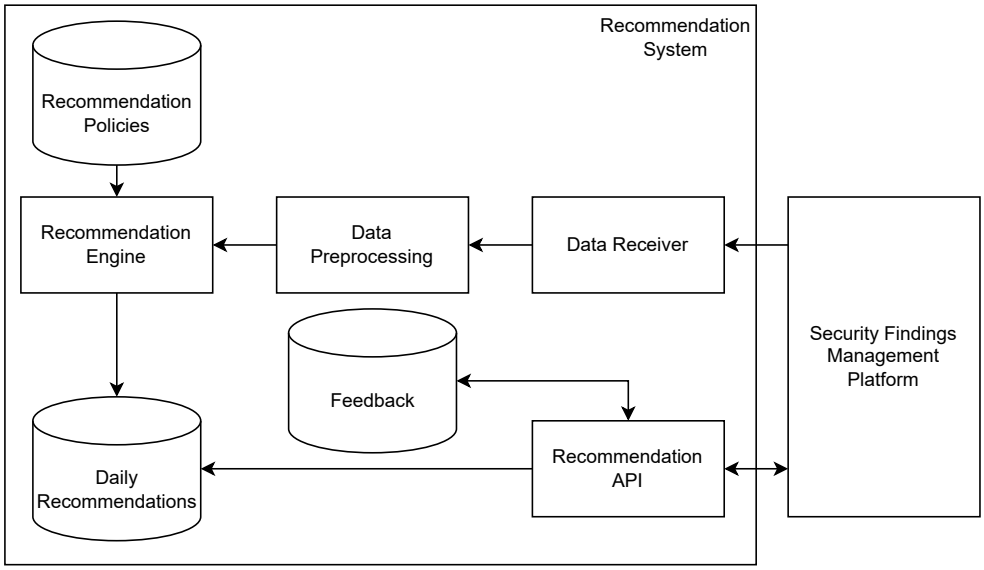

Figure 6.1: Recommendation System Design for Security Findings Management

security findings appears to be a valuable domain for future work.

6.4 Implementation

To employ the communication strategy in practice and utilize the information acquired during the security findings management, the solution approach of this chapter must be implemented and integrated into the existing platform. This implies that the implementation of this chapter builds upon the implementation of all previous chapters.

6.4.1 Restful API

The first part of the implementation focuses on the text-based communication interface. Towards this goal, the existing implementation of the platform is re-used and extended to support all demands. First, all suggested presentations can be implemented using *Queries* of the knowledge base. This implies that six new queries are created. The presentation of all security findings as a sort- and filter-able list is technically implemented by Elastic query searching for all security findings fulfilling the constraints. The results are listed and returned. For the presentation of a single security finding with all respective information, the *Summary* query implemented in the last chapter is re-used. To present the prevalence of locations, the amount of locations in each *Aggregated Security Finding* is counted, and a sorted list is returned. The number of entries in this list is determined by the initial request so that a *Location Prevalence* query with the result number set to two would respond:

```
[{"location": "file1", "prevalence": "10"},
{"location": "file2", "prevalence": "8"}]
```

The statistics on the risk imposed by security findings are presented by another new query, which takes the ”Base Score” of all security findings that are either in the ”Open”, ”Delayed” or ”Risk Accepted” state. These scores are grouped according to the orientation scale of Table 5.2 into ”Critical”, ”High”, ”Medium”, ”Low”, and ”Info” and the returned as list:

```
[{"severity": "Critical", "number": "2"},
{"severity": "High", "number": "6"},
{"severity": "Medium", "number": "34"},
{"severity": "Low", "number": "27"},
{"severity": "Info", "number": "11"}]
```

Finally, the presentation of the occurrence and state history of findings is implemented by multiple queries. Even though this could be implemented by one single query, it was split across multiple ones to reduce the load on the knowledge base and adhere to the concept of knowledge base queries in which each query answers one particular question instead of combining multiple ones. The first query presents all statistics on the number of findings in a specific state. The second query gives all state changes achieved in a particular timeframe. This includes, e.g., all new findings or findings that have been fixed within a specific time. Similar to the *Location Prevalence* query, this query allows to specify a timeframe for the response. The last new query shows the amount of security findings that have been reported. For this query, the timeframe of the presentation, as well as the bucket size, can be defined. The bucket size determines the aggregation of finding occurrences, e.g., daily, weekly, or monthly. A combination of a one-week timeframe with a daily bucket size could result in the following response:

```
{"Day1": 0, "Day2": 34, "Day3": 35, "Day4": 39,
"Day5": 31, "Day6": 28, "Day7": 0}
```

In summary, the API comprises three endpoints to interact with the knowledge base. The */data* resource allows adding new belief or reading unprocessed belief instances. The */userinput* resource enables platform users to add belief instances related to external user input. Finally, the newly introduced */query* resource can execute all queries described in this thesis. Moreover, multiple endpoints for authentication, authorization, logging, or debugging exist, which are not related to the communication strategy and, therefore, are not described in more detail.

6.4.2 Webinterface

During the second part of the implementation, the previously defined queries must be interpreted and presented in visual form. Therefore, a webinterface that is decoupled from the initial platform was developed. In Figure 6.2, the integration of the communication strategy with the components of the security findings management platform (Figure 3.6) is depicted. The webinterface itself is implemented in the NextJS framework. Since the visual interface uses the RESTful API of the platform, any visual presentation is also available in its text-based form.

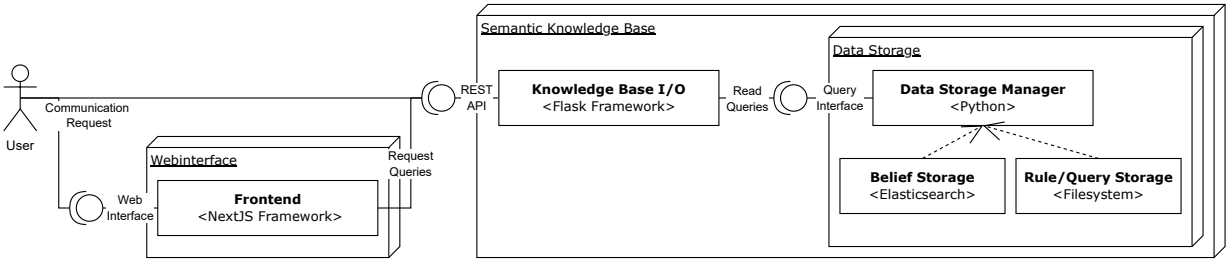

Figure 6.2: Implementation of the Communication Strategy

The fundamental pages of the webinterface itself are depicted in the Appendix (C). The common starting point for each user of the webinterface is shown in Figure C.1. It consists of five *Presentations* including the number of security findings currently in a particular state, the list of most critical security findings, a list of all team members, the state changes in the last week, and a bar chart on the severity of all currently applicable security findings. The "Overview" page solely provides all *Presentations* that are relevant to the respective user. This implies that another role within the project would receive another selection of *Presentations*. The navigation bar on the left allows users to access other webinterface pages, including the list of all security findings ("Findings"), a list of all team members ("Team"), or all *Presentations* that exist, regardless of the user role ("Diagrams"). Listing all security findings is implemented as a table with the ability to utilize search-, sort-, and filter- functionalities as depicted in Figure C.2. When one particular security finding is selected, its data is presented, and the functionality of the */userinput* resource for this finding is supplied (Figure C.3). This implies that users can provide input on priority, analysis, state, or response using this page. In addition, the webinterface has several functionalities not related to the communication of security findings like account management. These are not presented in more detail.

6.5 Conclusion

This chapter presented the communication strategy for the security findings management process. We introduced an approach for communicating security findings and information derived during their management to humans and systems. The developed presentations in visual and text-based form are intended to support all targets of the communication in achieving the related goals. This strategy was implemented as part of the security findings management platform and integrated with all previous solution approaches. In addition to contributing a security findings communication strategy, this chapter identified recommender systems for security findings management as a promising domain for future work. No preliminary evaluation has been conducted for this solution approach. Instead, the communication strategy is evaluated as part of the final solution evaluation presented in the next chapter.

Chapter 7

Instantiation and Evaluation in the Industry

The last chapters presented the problems and potential solution approaches for the management of security findings in modern industrial software engineering. Following the iterative Design Science Research approach (Figure 1.2) selected for this thesis, the last step is an empirical validation of all solution approaches to solve the overarching problem instance. Reinforced by the conclusions of each previous chapter, all proposed solutions must be evaluated as one methodology for the management of security findings in modern industrial software development projects.

In this chapter, the instantiation of our methodology comprising all solution approaches is presented and evaluated utilizing a case study at our industrial partner. First, the instantiation of all solution approaches and their merging into one platform is described. Second, this platform is used to evaluate the proposed methodology in ongoing software development projects. The results of this evaluation are discussed and final conclusions are drawn. The case study conducted in this chapter is further published in [193].

7.1 Implementation and Integration of the Methodology

In this section, we present the instantiation of our solution design to allow an evaluation of our proposed methodology in practice. We present the implementation of the current security findings management platform, show adjustments that improve its usability in practice, and describe how the platform should be integrated into projects.

7.1.1 Implementation of the Knowledge Base

For the instantiation of the final solution approach, we utilize the platform for managing security findings, described in Chapter 3 and include the implementation of Chapter 4 and Chapter 5. Moreover, the interfaces to the platform have been adapted and extended as

described in Chapter 6. This results in a semantic knowledge base with a web interface containing eleven belief classes, five rules, and nine queries. These are depicted in Figure 7.1, structured according to the chapter they have been introduced in and containing the relation between each other either through logical inference or usage within a query.

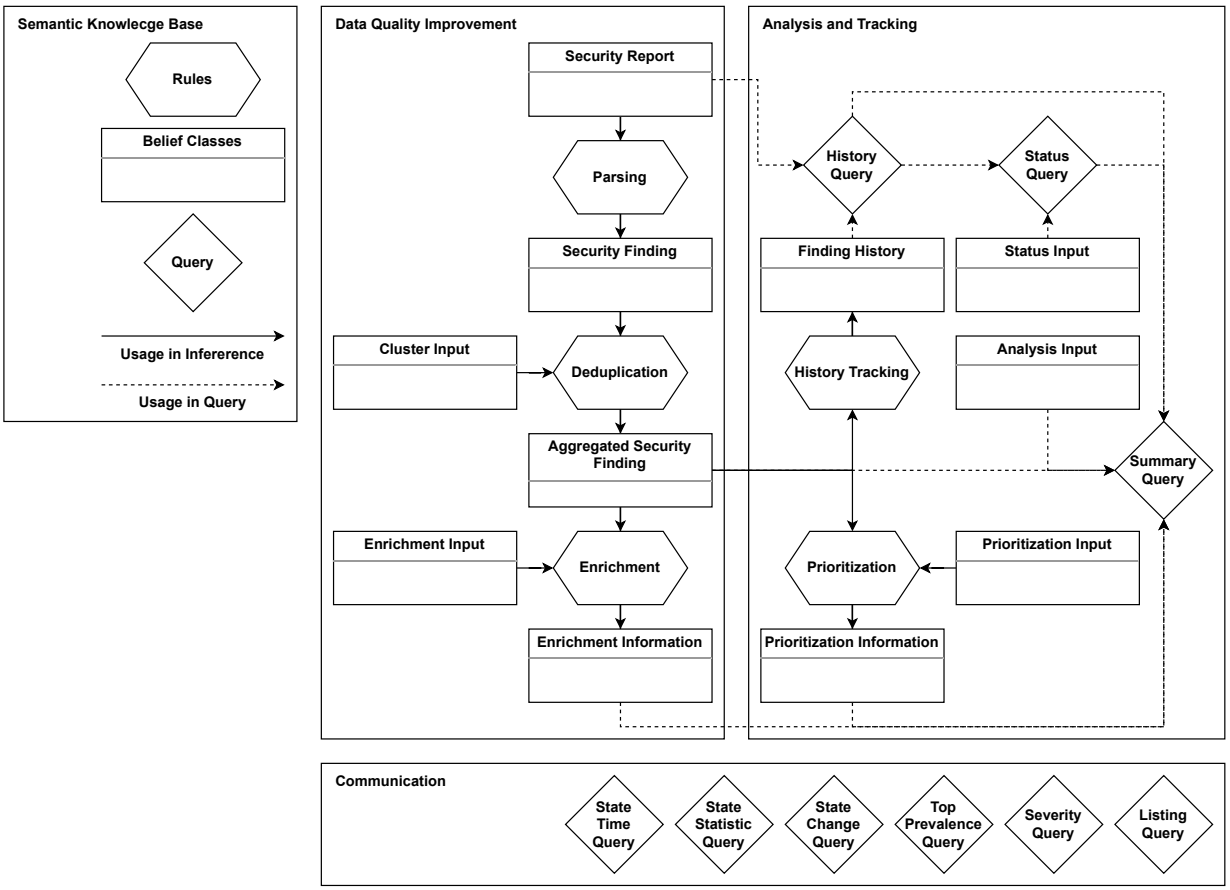

Figure 7.1: Belief Classes, Rules and Queries implemented in the Platform

During Chapter 5, the lifecycle of a security finding was proposed as a state machine, implemented as part of the *Status Input* belief class, and extended throughout the chapter. In summary, each security finding can reach eleven different states. Nine states further represent final states, where the security finding might persist permanently. This excludes the states "Open" and "Applicable" as they require subsequent activities before the management of the respective finding is considered finished. Each state can be reached from any other state with a respective transition. This implies that a respective transition exists for each of the eleven states. An example would be the "Fixed" state, which can be reached from any other state with the transition "Implemented Finding Fix". The only exception is the "Absent" state, as the finding state persists regardless of whether the finding is still reported. The "Absent" state can only be reached from the states "Open" and "Applicable". In all other states, a no longer reported finding represented by the transition

"Finding no longer Reported" is a self-transition. In addition to these eleven transitions, each state has another self-transition if a review confirms the current state, expressed by the transition "Confirmed Current State". The resulting state machine is defined below:
$Q : \{Open,\ Fixed,\ Absent,\ False\ Positive,\ Closed,\ Not\ Applicable,\ Applicable,$
$Risk\ Accepted,\ Risk\ Transferred,\ Risk\ Avoided,\ Mitigated\}$
$q_0 : \{Open\}$
$F : \{Fixed,\ Absent,\ False\ Positive,\ Closed,\ Not\ Applicable,\ Risk\ Accepted,$
$Risk\ Transferred,\ Risk\ Avoided,\ Mitigated\}$
$\sum : \{Re-/Identified\ Security\ Finding,\ Confirmed\ Current\ State$
$Implemented\ Finding\ Fix,\ Implemented\ Finding\ Mitigation,$
$Confirmed\ Applicability,\ Disproved\ Applicability,$
$Avoided\ Risk\ Treament,\ Determined\ False\ Positive,$
$Accepted\ Finding\ Risk,\ Transferred\ Finding\ Risk,$
$Avoided\ Finding\ Risk,\ Finding\ no\ longer\ Reported\}$
The visual representation of this state machine is avoided due to its complexity.

The knowledge base implementation is extended by a webinterface and RESTful API, representing the communication strategy. Both utilize authentication and authorization mechanisms to protect the confidentiality and integrity of the data. A PostgreSQL database is used to store the respective user accounts and rights. The different components are implemented as Docker containers and orchestrated by Docker Compose.

This implementation of the semantic knowledge base, including its interfaces is called the **S**ecurity **F**eedback **L**oop **A**nalysis and **M**anagement **A**pplication (*Security Flama*). This terminology is used in the remaining chapter when referring to the implementation of the methodology. The implementation of the Security Flama was conducted in collaboration with multiple students over the course of three years. Colin Wilk and Florian Angermeir collaborated in the development of the operational environment, while Philipp Kamilli and Sandip Sah supported the implementation of the visual interface.

7.1.2 Tweaking the Knowledge Base

Following the secondary results of the preliminary evaluations presented in the previous chapters and system tests, multiple changes to the implementation of the methodology have been conducted. These changes aim to improve the existing platform's functionality or usability.

The first change to the implementation of the knowledge base is the summary of certain inferences. The more belief classes and logical inferences are used in the semantic knowledge base, the more maintenance cycles are required to retain the soundness of the knowledge base. Hence, the first tweak to the knowledge base is summarizing the enrichment and history tracking inference. Since these rules are executed on the same source data, namely an instance of the aggregated security finding class, and experience showed that enrichment input is seldom changed or added, summarizing these inferences saves one maintenance cycle.

Another change to the knowledge base is outsourcing the logical core into temporary

storage instead of utilizing a traditional database or file storage. Since the logical core is continuously read and rewritten, the usage of volatile memory improved the knowledge base's performance by more than 30 percent. To cope with blackouts, the current logical core is backed up after each finished maintenance operation.

This leads to the third change of the platform, which is the usage of versioning for maintenance operations. While the maintenance of soundness can theoretically be performed in parallel, meaning that subsequent changes to the knowledge base can be maintained alternatingly, we assigned each maintenance operation with a version number to execute it consecutively. The parallel execution complicated the maintenance of soundness, as there was no partially sound status of the knowledge base reached. By executing maintenance operations consecutively, intermediate states representing a sound knowledge base at a certain time are reached. Hence, answers to queries can be provided faster, even if they report a currently outdated version of the knowledge base since they do not wait until all maintenance operations are finished. Experience-wise, this is preferred in industrial practice, especially during times with multiple changes to the knowledge base.

This demand for fast feedback also necessitated the fourth change to the implementation, as a caching system was required for the queries. Minding that multiple platform users request the same query, a re-calculation of the query, regardless of whether changes occurred or not, is inefficient. Hence, each query is cached and only re-calculated after changing data.

The fifth change affected the list of security finding states. As the first change, the "Absent" state was renamed to "Disappeared" since the adjective absent was perceived as if a test was successfully completed instead of a security finding that was identified but has not been reported again. Another change was the new finding states "In-Work" and "Delayed", which have been introduced based on preliminary feedback.

With these changes, the knowledge base is evaluated in this chapter.

7.1.3 Integration of the Methodology

Before evaluating the methodology, it must be integrated into the respective projects under test. Some proposed solution approaches require preparation and customization before being used in practice. The necessary steps and supplementary material are listed in this paragraph. In addition to the steps below, the methodology should also be introduced to the project team using hands-on training to educate them on using the methodology.

Before the methodology can be integrated into a project, information must be collected from the project and its overarching organization. First, all security activities reporting security findings must be identified. For each source, the collection strategy must be derived based on the type of reporting provided by the activity and the project demand on how and when to collect reports. Since most security findings arise from security tests conducted in CI/CD pipeline, we supplied the *Haystack*, a containerized application to upload findings to the Security Flama securely. A pipeline runner can execute this container to collect all security report artifacts of the pipeline and upload them with the necessary context information. Next, the demand for enrichment must be acquired from the project

team and the security findings state model, including the validity period for the states reviewed. Potentially necessary changes to the existing approach must be documented. All known influence factors on the prioritization of security findings are collected to prepare the prioritization of security findings. Similarly, the established risk management strategies are acquired, and the risk acceptance criteria and the validity period for risk reviews are collected. Finally, the users of the system and their respective rights are determined.

Based on the sources of security findings in the project, the preconditions for the collection must be implemented, and one parser model and base score model must be written for each source. In particular, this necessitated the development of a lightweight Docker container that can be used as a pipeline runner to upload reports from CI/CD pipelines securely. Next, additional rules for the internal and external enrichment of security findings and any modifications to the state model are implemented. If influence factors on the prioritization are known upfront, these are realized by writing findings prioritization policies. The risk review period is defined for the respective states and the user and role concept implemented.

In addition to the technical changes, further guidance to comprehend the security findings management process is supplied, e.g., in the form of a wiki. This includes explanations on the implications of each finding state, clarification for the security findings priority scale, and guidance on the security findings response. Moreover, the previously acquired data on conducting the risk assessment of security findings in the organizational context is documented. After finishing these integration steps, the methodology can be used in modern industrial software development projects.

7.2 Evaluation Planning

This section discusses and presents our decision process on the evaluation design and the research strategies we employed. We discuss the selected research strategy, the selection of data collection and analysis, and finally present our decision.

The goal of the evaluation is to analyze the impact of our methodology on industrial software development projects. Since our methodology is designed for industrial software development projects, a meaningful evaluation can solely be performed with this scope. As there is no greenfield approach for research and evaluation in empirical software engineering [115, 172], we require a precisely defined evaluation approach customized to our methodology and evaluation goals.

7.2.1 Evaluation Strategy in Industry

The first step towards evaluating our methodology is selecting an empirical research method. However, there is no such thing as a silver bullet for software engineering research [172]. Consequently, the informed selection of an evaluation method based on our particular setup is essential for solid claims. In contrast to the former chapters, which are based upon solution-seeking research, we want to evaluate the impact of our methodology in

this chapter utilizing knowledge-seeking research approaches [172]. This already limits the commonly accepted research methodologies, which could be relevant for our evaluation. According to the *ABC Framework* by Stol and Fitzgerald [172], we can cluster research strategies into four quadrants, each impacting the generalization of results, control over measurement/behavior, and the realism of the context. Figure 7.2 depicts a simplified version of the framework, including solely the quadrants and their research strategies. In the following, we discuss each quadrant with its respective research strategies.

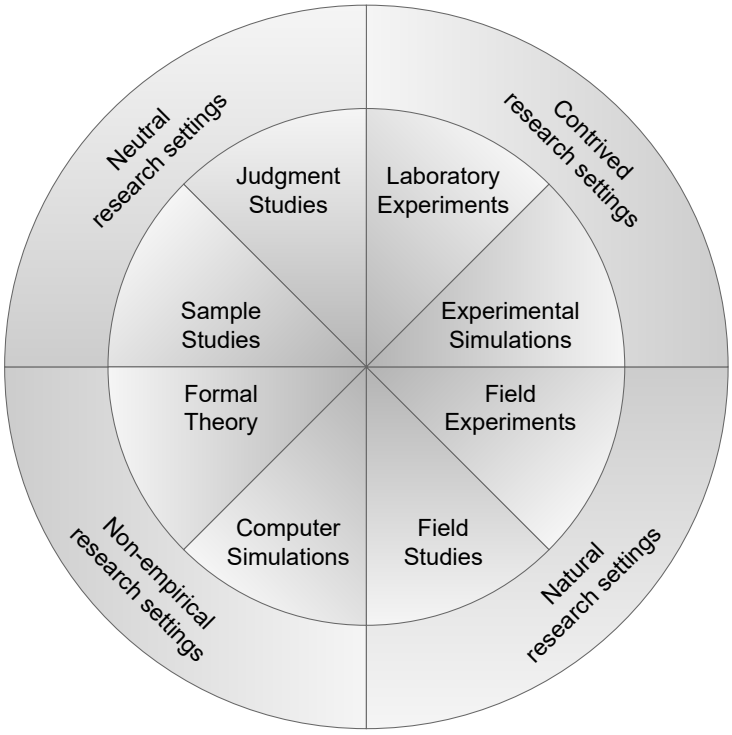

Figure 7.2: Simplified ABC Framework

For the purpose of our evaluation, we can instantly remove one quadrant from our scope as it addresses *non-empirical research settings*. With our methodology targeting actual software development projects, any claim resulting from computer simulations or formal theory is of limited relevance to the initial challenge of this thesis.

The second quadrant that does not provide the necessary characteristics for our evaluation addresses *neutral research settings*. These include, e.g., sample studies (surveys) or judgment studies. While surveys might be helpful to identify the need for such a methodology upfront or collect ideas for potential solutions, their usability for evaluating an artifact is extremely limited. Judgment studies, on the other hand, might be beneficial for exactly this case by presenting the methodology to systematically selected domain experts and asking for their input. While this can provide valuable insights into potential benefits or shortcomings, it cannot answer whether this it also applies to practice. Consequently, it could instead be utilized as a preliminary evaluation to eliminate flaws upfront before

evaluating it in a laborious and costly sophisticated evaluation. Furthermore, this entire class of research strategies misses the concept of a particular research setting that we fundamentally require to evaluate a methodology intended to be used in industrial software development projects.

The remaining two quadrants, covering contrived and natural research settings, could be potentially used for our evaluation. Consequently, we will discuss the advantages and disadvantages of all four research strategies comprised by the quadrants with respect to our industrial setting. The first strategy we discuss is the laboratory experiment. Applied to our use case, we could evaluate the methodology by, e.g., dividing our participants into two groups and asking them to manage security findings. While one group would receive the methodology, the other would be asked to complete the tasks without it. This strategy might even include existing industrial data or teams of professionals as participants to improve realism. But even these amplified properties cannot cope with this strategy's overall low level of realism. Since we are particularly interested in the impact on practice and are willing to sacrifice measurement precision for a higher degree of reality, it is less suitable for our use case than other research strategies. The second strategy we want to address is the field study. Here, we would investigate industrial software development projects and document their approaches and processes for managing security findings. The main problem of the field study is the level of control we can impose on the project. As we cannot manipulate any project parameter, it is impossible to introduce our methodology. Furthermore, no projects actively use the methodology without us interacting with them upfront. Consequently, we cannot evaluate it, and the strategy is not relevant to our use case.

The third strategy we want to examine is experimental simulations. The most crucial difference to more realistic strategies is that the research environment does not exist before or after the study. This implies the creation of an artificial software development project just for the evaluation procedure. Furthermore, the participants of experimental simulations are recruited for the study. The selection of participants for the study is a crucial aspect to the generalizability of any claims arising from it. The drawbacks of using non-professionals are extensively discussed by Feldt et al. [50], including limitations towards claims formulated when using students as participants and general threats to the research's internal, external, and construct validity. To avoid this controversy in our evaluation, we decided against utilizing students as participants. However, encountering real-world professionals willing to provide their time and effort for a scientific evaluation while, in parallel, trying to cope with their regular work effort is challenging. This challenge is further amplified if an interconnected group of subjects, like an entire software development team, is needed for research instead of single team roles (Developer, Architect, etc.). Graziotin et al. contacted randomly selected developers for a survey, achieving a response rate of approximately 4% [59] (Wagner et al. in a similar setup only 2% [198]). Consequently, we must balance the number of participants in our evaluation with the level of realism we can achieve when recruiting them. Both sides affect the generalizability of the evaluation. Using a professional software development team for the evaluation provides a high level of realism, but finding them is complex and using their time is expensive. Hence, this

might come with the drawback of having only a few development teams available during a shorter period. With a team of mixed subjects, we might be able to form more artificial development teams that could work with us for a more extended period of time. Even though this allows us to receive more data from diverse groups, the relevance of the data is questionable. While experimental simulation comes at the cost of a reduced level of realism, the opportunity to introduce events or ask for specific tasks is a major advantage when compared to more realistic strategies. Certain events like a security incident or a high number of security findings are events that might occur once or potentially never during a real-world development project. This capability allows us to test for such edge cases without waiting for them to happen naturally. In summary, the usage of toy projects in simulations is time and cost-effective but is solely helpful for illustrating the methodology while, however, providing no insights into its practical relevance [119]. Furthermore, we agree with de França and Ali that simulations and models do not represent an equal alternative to field studies or experiments, especially for our industrial setting [36].

Finally, this leaves us with the last strategy: field experiments. Field experiments are conducted in natural settings, but in contrast to field studies, they allow a certain degree of manipulation. This already sets the scope for the evaluation, as we would be using ongoing industrial software development projects with actual professionals as participants. Working within the allowed degree of manipulation, we could introduce our methodology and observe any changes. Therefore, this strategy provides the highest level of realism while still being suitable for an evaluation. The disadvantages of this level of realism are just as obvious. Similar to a simulation with a high-realism participants group, finding subjects and performing an evaluation might result in low participation, high cost, and a short evaluation period. This challenge is further reinforced since the research environment is an actual project. Hence, recruiting subjects does not exclusively involve finding human participants but rather finding ongoing software development projects with an existing development team. Therefore, the focus during recruitment shifts towards locating project leads or responsibles willing to support the evaluation, which manage a project team and stakeholders that also agree to contribute to the evaluation. In this case, a higher commitment to the research is necessary, as we tamper with potentially critical data in an integral business-oriented project. The last statement especially initiates the question of realism in industrial projects. Can we consider a project consisting of one developer already as realistic? Or do we focus on the project's customer, implying that any software written for a customer outside the project team is realistic? Industrial software often consists of multiple components, all written by different project teams. Can each sub-project be considered a separate subject group, or can an evaluation only be grounded on the umbrella project? To avoid an extension of this discussion, we defined our own constraints for realistic industrial software development projects. We consider a project to achieve the necessary level of realism if it is

(1) following State-of-the-Art project management practices,
(2) developing software for a customer in an industrial sector, and
(3) consisting of a project team including developers.

Unfortunately, the control advantages of experimental simulations do not apply to field

experiments. We can, e.g., not control the amount of security findings the project has to deal with. Consequently, we might be unable to state any claims if a project has no applicable security findings during the evaluation. Another drawback of this approach is introduced by the type of artifact that is evaluated. The demand for protecting this data arises as we work with confidential data (security findings). Hence, a significant effort must be put into securing the application and the respective infrastructure by the researchers before the evaluation can be started.

7.2.2 Quantitative vs. Qualitative Evaluation

After the research strategy is selected, the next decision deals with the data collection and analysis. This decision is often reduced to quantitative vs. qualitative data collection processes. The respective quantitative or qualitative analysis methods used are based on the collection approaches. In most cases, the selected research strategy is already closely coupled to the research process. Field experiments and surveys are traditionally bound to quantitative research, while interviews or observations are more associated with qualitative approaches [49]. However, this distinction between quantitative and qualitative research is not only connected to the type of data collected but also to the objectives [61]. In the following, we discuss the advantages and disadvantages of quantitative and qualitative research approaches for evaluating artifacts like our methodology in industrial settings.

Quantitative Research Approach

Quantitative Research Approaches are fundamentally data-driven approaches aiming to identify relationships between different aspects of the research environment. This allows researchers to identify phenomenons based on measurable variables and mathematical/statistical models for data analysis [61]. An example of a quantitative approach for our use case would be the impact of using our methodology on the security of a software product. The first variable, our methodology's usage, can be true if it is employed in the project or false if not. For the second aspect, the product's security, we would need to define a set of variables that capture this aspect of the product. This model would be simple as the first variable can solely take binary values.

The first challenge arising from quantitative approaches is the variable selection for meaningful claims. In our example, identifying variables that cover the "security of a product" is challenging. Is "security" represented by the amount of post-release software security issues? Or are there more aspects that influence the "security"? Most likely, we would need to specify the claims and check the impact of our methodology on the post-release security issues, thus weakening our claim. Even with adequately defined variables, we would encounter the challenge of internal validity. Since we measure the influence of one set of variables on another, it must be ensured that there is no third group of variables falsifying our claims. Following our example, we would need to ensure that the usage of our methodology causes a lower amount of post-release software security issues. However, software development teams might have improved their skills over time, resulting in fewer security issues in the code before they could even be identified. This either results

in threats to the research's internal validity or substantial effort upfront for the creation and testing of models that cover the phenomenon. However, even in the second case, a potential threat to the internal validity can never be excluded. Finally, we can mathematically measure particular phenomena and derive conclusions from them. This relies on the assumption that the aspects we are interested in are mathematically measurable. Software development projects, however, take place under complex conditions, often hardly expressible by mathematical terms [115]. The usage of methodologies by project teams often depends on whether it is perceived as beneficial. This highly subjective perception can hardly be captured in a variable. Consequently, we might not be able to capture all positive and negative impacts of the methodology when solely relying on quantitative research approaches. Finally, the amount of data collected to derive representative claims is relatively high. However, drawing these large sample sizes in software engineering is difficult [115]. Furthermore, identifying the necessary sample size to achieve representativeness underlies the challenge of first identifying the entire population. However, there is no commonly accepted number of all industrial software development teams employing modern software development practices. Therefore, taking advantage of the representativeness provided by quantitative research approaches is often not possible in industrial software development projects.

Qualitative Research Approach

Qualitative Research Approaches, on the other hand, focus on understanding the reasons behind a phenomenon [61]. It focuses less on mathematically measurable indicators to identify a phenomenon and instead more on reasons and mechanisms that lead to the phenomenon. An example of a qualitative approach for our use case could be understanding our methodology's positive or negative impact on the development team. This could be realized by conducting interviews with the project teams using the methodology.

One challenge when using qualitative research approaches is the necessary intervention in the project. The origins of relevant data are primarily human. Hence, the data collection requires time and effort to retrieve this data, e.g., by using interviews with the project team. Researchers and the project team alike invest this effort. Consequently, a high commitment to the evaluation by the project team is required upfront. This type of data acquisition further comes at the disadvantage of potentially influencing the retrieved information. This can include the introduction of a bias in the interaction with the subjects (e.g., biased questions) or threats towards the construct validity if the interaction is differently interpreted between researcher and subject. Finally, whether to use qualitative or quantitative research approaches is derived from the research question asked [115]. However, the challenges arising from the selection accompany the remaining design and implementation of the research approaches. Hence, it is to be considered during the decision process.

7.2.3 Evaluation Approach

The discussion in the last paragraphs showed that the evaluation must be precisely planned. Furthermore, they indicate that a field experiment or an experimental simulation possesses the characteristics to conduct our evaluation. We accepted that the high level of realism could limit the statistical generalizability of our claims, as our sample size might be too small to represent the entire population of all industrial software development projects [172]. Regardless, we believe that only a realistic setting can provide valuable insights into the impact of our methodology.

Moreover, we decided against an initial hypothesis claiming the usefulness of our methodology. Instead, we want to explore the benefits and challenges of the methodology, decreasing the risk of potential bias [155]. However, we assume that the usage of our methodology impacts the software development project to some degree. Our target is to understand how this impact manifests. Consequently, we want to explore the impact instead of explaining how our methodology impacts the project, traditionally coupled to qualitative research [205]. However, we utilize quantitative approaches to check for mathematically measurable impact while employing qualitative techniques to identify subjective influences. This approach allows us to use methodological triangulation [155], increases the precision of our research, and avoids known challenges [204].

7.3 Evaluation Design

For the design of our study, we follow the state-of-the-art empirical software engineering and case study research [155, 115, 110, 164] minding challenges and domain experience [138]. In this section, we present the objectives of the evaluation and the outline for conducting it.

Since the evaluation was performed in an industrial company, not all data can be ethically and legally disclosed. It is marked in the respective section whenever data was collected but not disclosed on purpose. Consent agreements with the respective industry partner have been established upfront for information disclosure.

7.3.1 Evaluation Objectives and Research Questions

The objective of the evaluation is to identify the impact of using our methodology in industrial software development projects that utilize modern software development approaches. In the following, we define each aspect of the objective and associate them with the respective actions. First, the impact to be explored is quantified by the benefits and limitations of applying our methodology in the project. Since the objective of the methodology impacts the project, the data points to obtain relevant information from are the project team and measurable properties of the development process. Finally, we limit the evaluation to software development projects performed in the industry that utilize modern software development approaches.

From these objectives, we derived our research questions:

1. **RQ1** How does the usage of the methodology affect the development process indicators?
2. **RQ2** Is the methodology perceived as useful by the project team for managing security findings?
3. **RQ3** Which benefits does the using the methodology bring?
4. **RQ4** Which limitations exist when using the methodology?

The first question aims to collect non-subjective data to identify whether there are any changes in project performance indicators. With the second question, we want to focus on the personal perception of project team members working with the methodology. The third research question focuses on the advantages of using the methodology, while the fourth one addresses its shortcomings and potential disadvantages. Overall, we avoided testing for a hypothesis with our research questions and instead solely assumed that there is some degree of change when using our methodology, which we want to explore in our evaluation.

Based on these research questions, a case study in real-world projects employing our methodology seems to be the most promising research approach. In the following sections, we discuss the characteristics of our case study.

7.3.2 Subjects and Cases

The first step towards our case study is the selection of cases and subjects. As discussed in the planning, we want to achieve a high-realism evaluation. Consequently, we decided against any simulation-based approach and instead conducted a case study with ongoing software development projects. In the software engineering domain, the straightforward selection for a case is a software development team. Following our evaluation scope, we restrict this selection further to industrial software development projects that utilize modern software development approaches. To narrow the case selection to active and mature software development projects, the project has to run for at least one year and follow established security practices, including automation of security tests and processes for managing security findings. Moreover, the project has to be developed actively, necessitating at least one code change per week for the last month at the time of selection.

These restrictions result in a list of requirements depicted in Table 7.1. The project is unsuitable for our evaluation unless all questions can be answered with yes.

Question	Answer
Team develops software in an industrial setting	
Team follows DevOps or Agile principles	
Project runs for at least one year	
Codebase is changed at least weekly for the last month	

Property	Project A	Project B
Project Domain	Research Project	Research Project
Development Approach	Adapted Agile/DevOps	DevOps
Author Involvement	None	None
Amount Developer	3	2
Project Team Size	4	3

Table 7.2: Subject Project Properties

Project has automated security tests	
Project has a active process for managing security findings	
The project is an actual software development project	

Table 7.1: Recruitment Requirements for Projects

To ensure all projects originate from an industrial source, we contact the industry partner of this thesis for potential evaluation projects. To follow the selection criteria, the project owner or manager of each project proposed by our industry partner was contacted, and the employed development principles and project duration were assessed. Moreover, we determined the level of realism for each project in collaboration with the project lead based on the principles presented in the evaluation planning. Due to the differences between documented and applied processes in projects, we requested to witness the established findings management approach upfront. Similarly, we asked to see the automated security testing in practice. For the changes in software code, we requested access to the code repository to ensure the fulfillment of this requirement.

Finally, we identified two projects for our evaluation, fulfilling all requirements defined above. The properties of both projects are listed in Table 7.2. Both projects utilize the Gitlab CI environment to automate security tests and manage the resulting findings as artifacts in the Gitlab environment.

7.3.3 Data Collection and Storage

To answer our research questions in collaboration with the selected subjects, we collect and analyze data from the project. In this subsection, we present the data collection and storage strategy. The actual protocol conducted for data acquisition can be found in Subsection 7.3.4.

As discussed in the evaluation planning, multiple strategies for data collection could be applied in our case study setting. For our evaluation, we seek information in two distinct areas of data points. The first area is the project team, where we want to collect first-degree data by conducting interviews. The second area is process performance indicators, where we obtain data by second-degree collection, using passive observation of project statistics without any project interaction.

Throughout the case study, any data acquired is stored electronically and, due to the confidential nature of security findings, protected against illegitimate access.

7.3.4 Case Study Protocol

To conduct the study, our research follows a defined case study protocol. This protocol is based on the data acquisition and storage strategy of Subsection 7.3.3 and is described in this section. In advance to the actual case study, we decided to perform a preliminary pilot evaluation to avoid confounding factors and improve our construct validity. The unit of analysis for this pilot is the project team that develops the Security Flama itself. As most project team members have a limited understanding of the scientific background of the implemented artifact, it seems to be a promising piloting environment. However, we limit the bi-weekly interviews to a single occurrence. Otherwise, we strictly follow the protocol defined below. The results of this initial study are discussed with the subjects to ensure that the questions were correctly understood.

After the pilot, the case study is executed in parallel on the different units of analysis. We decided to follow a common procedure to collect data for each project. The six steps of this protocol are listed below and will be discussed in the following paragraphs.

1. Integration of the Flama Haystack according to Subsection 7.1.3
2. Passive, continuous collection of statistics from the project
3. Introduction of the Security Flama to the project team
4. Data collection with multiple iterations of questionnaires
5. Conclusion of data collection with interview
6. Ramp-Down

(1) During the first phase of the procedure, Haystack is integrated into the projects. As described in the implementation section of this chapter, this enables the collection of security reports from the security tools established in the project. At this point, the project team is aware that data is gathered from the project, but the Security Flama has not been introduced yet. Starting from this point, the researchers have to perform regular reviews of the security activities employed by the projects. As tools might change their report format or content, aspects like report parsing or the severity calculation for the prioritization must be adapted accordingly. However, this is solely a passive interaction with the project without any active intervention, as the awareness about changes is sufficient to adapt the implementation of the artifact.

(2) Based on the data provided by the integrated Flama Haystack, we collect the observational statistics from the Security Flama. This collection is performed weekly on Sunday at midnight UTC throughout the entire timeframe of our evaluation. Details on which

data is collected and how the collection is performed can be found in Subsection 7.3.5.

(3) After collecting data passively for four weeks, we introduce the Security Flama to the project team. From a technical perspective, this includes creating user accounts for all project members and ensuring access to the web interface since the collection of findings data is already in place. On an explanatory level, we introduce the project team to the functionality of the Security Flama and all potential user interactions. Towards this goal, one researcher presents the Security Flama according to the guidance presented in Subsection 7.3.5. Furthermore, we introduce the case study and its scope of analyzing the benefits and limitations of applying the methodology for industrial software development projects.

(4) After introducing the Security Flama, we perform bi-weekly interviews with the project team. Instead of performing just one interview, we are particularly interested in the change over time, as the perception of the methodology might evolve [198]. These interviews are conducted per the defined interview protocol, employing the same questionnaire each time. The questionnaire and interview guide can be found in Subsection 7.3.5. During this time, the researchers are not involved in the projects, excluding the bi-weekly interviews.

(5) After five iterations of the interviews, the case study is concluded by a final interview with each participant, requesting a retrospective of the last weeks using the Security Flama. The final interview is less structured than the predecessor, solely requesting information on the perception of using our methodology and potential benefits and challenges. The material for this final interview can be found in Subsection 7.3.5.

(6) After gathering all data, the study is ramped-down. This is performed by removing the team member's access to the web interface and removing Flama Haystack from the project's development environment. Furthermore, the project team is informed that the study concluded. If requested, the data acquired and presented by the Security Flama is handed over to the project team, and the remaining information is securely deleted.

Concluding the protocol, we acquired a dataset of passively observed statistics and the questionnaires obtained during the interviews. Afterward, the data is analyzed according to the approach defined in Subsection 7.3.6.

7.3.5 Case Study Instruments

The evaluation execution relies on multiple instruments to define the interaction with the research environment and our subjects. Below, we list the material for introducing the Security Flama to subjects (Introduction of the Security Flama), the reoccurring interviews with their questionnaire (Reoccurring Interview), the final questionnaire conducted during the last part of the study (Final Interview), and the observation of project indicator values (Observation Statistics).

Property	Reoccurring Interview	Final Interview
Duration	1 hour	30 minutes
Frequency	bi-weekly	once
Subject	One Team Member	One Team Member
Conductor	Single Researcher	Single Researcher
Documentation	Recording, Live Transcription	Recording, Live Transcription
Environment	Virtual with Screen Sharing	Virtual with Screen Sharing
Interview Type	(Semi-) Structured	Semi-Structured
Interview Scope	Since last interview (2 weeks)	Entire study time
Language	English	English

Table 7.3: Interview Characteristics

Introduction of the Security Flama
During the evaluation, the Security Flama is introduced to the project team during a one-hour presentation. During this meeting, one researcher presents the functionality of the Security Flama while aided by a slide deck. The slide deck can be found in the supplementary material [192]. The entire project team, excluding stakeholders, is present in this meeting. Any questions by the audience aiming towards clarifying functionality or use of the application are answered immediately. Deciding whether a question is answered during the presentation is the researcher's discretion. However, any question that can not be directly linked to the system's functionality is documented. During this stage, we further identify the interview subjects, present the scope and targets of the evaluation, and align regular appointments with them.

Reoccurring Interview
For the collection of qualitative data, we decided to conduct recurring interviews with the members of every project. Due to the similarities of structured interviews with questionnaire-based surveys [155], we additionally consider the work of Wagner et al. for the creation of the questionnaire [198]. The interviews are conducted during a one-hour-long meeting every two weeks. The researcher performing the interviews is assisted by an interview guide and the list of questions to be answered by the subject. Concerning the questionnaire, we decided to conduct semi-structured interviews with our project teams, mixing quantifiable and open-ended questions. These properties are listed in Table 7.3.

The interviews are started by welcoming the subject and clarifying, that the interview will be recorded. The interview environment is virtual appointments using the communication software Microsoft Teams. To avoid distractions, the interviewer ensures a proper environment upfront using the subject's camera. Afterward, both cameras are turned off to avoid visual bias. Next, a short introduction to the evaluation is presented, reminding the subjects of scopes and targets, followed by the actual questionnaire. This is initialized by the researcher sharing the questions through the screen share feature. Even though all

questions are written in English, subjects are allowed to answer in their preferred language to avoid confusion [198]. These are translated on the spot and re-read to them, ensuring a correct translation of their intention. The answers to all questions are digitally documented by the researcher.

The questionnaire consists of 22 questions asked subsequently. It consists of three categories. It starts with organizational questions, proceeds with questions to answer research questions RQ2, RQ3, and RQ4, and concludes with general feedback. The organizational questions address the subject and its project environment to identify the answer context. Since this information might change throughout our evaluation, we request this information every time. Each closed question is formulated to be answered either binarily or on a Likert Scale to state their agreement to the question. For the binary questions, the answers "Yes" and "No" are expected. The Likert scale is defined from 1 - "Strong Disagree" to 6 - "Strong Agree", avoiding any neutral answer. Since all questions are about personal perception, no default selection for rejecting a question is given. The remaining questions aim towards answering our research questions RQ2, RQ3, and RQ4. For RQ2, we want to identify the usefulness of our methodology from the different project team members, realized by various questions affecting the usefulness of the methodology in software development projects. The research questions RQ3 and RQ4 are summarized in our questionnaire to identify the benefits and limitations of using our methodology. Finally, we added two questions allowing comments by the subject that are not directly related to any research question. Since the interaction with the methodology likely changes over time, subjects must answer according to their experience during the last two weeks. Hence, all questions are formulated accordingly when this property is relevant. Table 7.4 depicts all questions with their respective category, our expectations on the answer format, and the timeframe they address. During the interview, the subject will solely see the question itself and be informed about the Likert scale, if applicable, for the respective question.

Category	Type	Timeframe	Question
Organizational	Open	Overall	How would you call your role in the project?
Organizational	Open	Overall	How many hours did you approximately spend on the project in the last 2 weeks?
Organizational	Binary	Overall	Did you interact with security findings and in particular the Security Flama last week?
Organizational	Open	2 weeks	How frequently did you interact with the Security Flama in the last 2 weeks?
RQ2	Likert	2 weeks	I perceived the methodology overall as useful for my work in the last 2 weeks.
RQ2	Open	2 weeks	What did you perceive as notably useful, when using the methodology during the last 2 weeks?
RQ2	Likert	Overall	I think that the methodology covers all aspects of the Security Findings Management.

RQ2	Open	Overall	Which aspects of the management of security findings are missing in/not covered by the methodology?
RQ2	Likert	Overall	I currently feel able to manage security findings.
RQ2	Open	2 weeks	How was the Security Flama affecting your ability to manage findings in the last 2 weeks?
RQ2	Likert	Overall	The methodology is simple to use.
RQ2	Likert	Overall	The methodology is transparent to me, hence I know how data is computed.
RQ2	Likert	Overall	The methodology is effective, hence it provides me with the desired and correct results.
RQ2	Likert	Overall	The methodology is efficient, hence it seems to minimize the time and effort I have to invest.
RQ2	Likert	Overall	The methodology is flexible, hence it allows me to perform necessary customization so that it fits the project.
RQ3/RQ4	Likert	2 weeks	Using the methodology was beneficial to my work during the last 2 weeks within the project.
RQ3/RQ4	Likert	2 weeks	Using the methodology restricted me in my work during the last 2 weeks within the project.
RQ3/RQ4	Open	2 weeks	Which benefits of the methodology did you encounter in the last 2 weeks?
RQ3/RQ4	Open	2 weeks	Which limitations of the methodology did you encounter in the last 2 weeks?
RQ3/RQ4	Open	2 weeks	Which areas of the project did the usage of the methodology affect in the last 2 weeks?
General	Open	Overall	What could be improved within the methodology to support your work?
General	Open	Overall	Do you have any further comments?

Table 7.4: Reoccurring Questionnaire

The interview guide, supporting the researcher during each meeting, is shown below.

1. Welcome the subject and thank for the collaboration. Explain that the recording is started next
2. Start the recording
3. Check the subject's interview environment and turn off both cameras

4. Explain that we want to identify the benefits and limitations of using the methodology
5. Share the screen showing the questionnaire
6. Conduct the questionnaire
7. Thank the subject and conclude the interview

This guide was solely available to the interviewer for the meeting period. After the meeting, the recording is compared to the interview documentation, and potential discrepancies are corrected so that the recording can be deleted in accordance with data retention policies. The data storage is described in Subsection 7.3.3.

Observation Statistics

For quantitative data collection, we collected values from development performance indicators passively. This automated collection happens weekly during the evaluation period and addresses our first research question by identifying our methodology's impact on development process indicators. Minding the primary target of assessing the impact on product security, we focus on indicators related to the software's security and the methodology's usage. The most valuable information to collect would be the methodology's impact on the product's security. However, this induces the question of how to measure the security of a product. Instead, we focus on measuring values that are associated with our methodology. Hence, we look into the data consumed by our methodology, the data produced by it, and the overall interaction of the project team with it. Table 7.5 lists all observed indicators.

Category	**Indicator**
Data Input	Amount of security activities producing findings in the project.
Data Input	Amount of security tools providing data to our methodology.
Data Input	Amount of reports created per week.
Data Input	Amount of raw findings created per week.
Data Output	Amount of correlated findings per week.
Data Output	Amount of new findings per week.
Data Output	Amount of findings with tag X (Open, False Positive, ...).
Data Output	Amount of findings with severity X (Critical, High, ...)
Data Output	Response time per finding.
Methodology Usage	Amount user input per week (Tags, Prioritization).
Methodology Usage	Amount of interface requests.

Table 7.5: Development Indicators

For each category, the collection is implemented passively by relying on the data acquired or produced by the Security Flama. A finding is considered to be new, if the last collection has not identified it. The interaction is monitored by accessing the logs written by the Security Flama. The data from each week is stored separately and finally analyzed

according to Subsection 7.3.6.

Final Interview
For the final interview, the guidance is equal to the reoccurring interviews. We restrict the interview time to 30 minutes with our subject due to the drastically reduced number of questions. In contrast to the reoccurring interview, we diminished the abstraction of questions from our research questions and addressed them directly. Using this approach, we expect an open discussion and a reflection across the entire study period. The properties of the final interview are listed in Table 7.3. The three questions discussed are in Table 7.6.

Category	Type	Timeframe	Question
Final	Open	Overall	What do you think about the usefulness of the methodology?
Final	Open	Overall	What is your opinion on the benefits of the methodology?
Final	Open	Overall	What is your opinion on the limitations of the methodology?

Table 7.6: Final Questionnaire

7.3.6 Data Analysis

Finally, the results from the methodology evaluation are analyzed to derive conclusions about the research questions. The analysis strategies differentiate between the two data acquisition strategies. In this section, we describe the analysis approach for the quantitative observation data and the qualitative interview data.

The data available from observing the development indicators provide the fields listed in Table 7.5 for 16 data points in summary. The primary strategy for analyzing this data set is the identification of outliers and correlations between the values. Moreover, the relation between the reported observation data, the interview data, and activities in the project might provide valuable insights or explanations for anomalies. This results in the following four data analysis strategies for the quantitative data:

- Identify anomalies within the same reported indicator
- Identify anomalies between different indicators
- Identify correlations between indicators
- Identify correlations between indicators and other data points like interviews or project information

In addition to the quantitative data, the interview results report additional qualitative insights from project team members. This supposedly provides a data set of five interviews with the reoccurring questionnaire and one final interview with each team member of both

projects. Minding that the reoccurring questionnaire consists of 22 questions and the final one of three, a substantial amount of data is accumulated during the interviews. The analysis of this highly subjective information is closely coupled to the type of question.

All answers to the *Organizational* category of Table 7.4 represent the factual baseline for interpreting all consecutive questions. Therefore, this context information is solely reported. The analysis of the answers to the ten closed questions is treated similarly to the quantitative data. Since the investigation's key target is identifying the benefits and limitations of the methodology, these quantifiable answers are used to derive the general perception of the team members towards the methodology. In contrast to the quantitative analysis, the design of the interviews adds the additional dimension of different team members reporting the data instead of solely having data points being documented for varying dates and projects. Therefore, not only outliers and correlations must be analyzed, but descriptive statistics such as mean values or standard deviations must be acquired. In summary, the following strategies for analyzing the closed questions are applied:

- Identify common answers for the same question
- Identify outliers for the same question across different subjects and dates
- Identify outliers between questions
- Identify correlations between questions
- Map outlier with quantitative results and organizational data

Finally, eleven questions did not restrict the format in which they should be answered. Five of these closed questions give additional insights into previously answered closed questions. The analysis of these questions supplies the discussion on the results of the closed questions. In summary, three questions focus on the benefits of the methodology, four focus on the limitations, and four are general questions to acquire additional insights. During the analysis, each answer to an open question is assigned the tag "benefit", "limitation" or "context" to summarize the results. The responses in each of the three groups are summarized to identify the benefits and limitations of the methodology.

7.4 Evaluation Results

This section summarizes the evaluation results following the collaboration agreement with our industry partner. Consequently, the results are partially anonymous. First, any particularities that occurred during the evaluation procedure itself are introduced. Next, the data acquired through the quantitative analysis is presented. Finally, the results of both interview types are documented.

7.4.1 Evaluation Procedure

The initial step during the evaluation procedure was the piloting of the study. This preliminary study resulted in two improvements to the subsequent case study with the actual subjects. First, the infrastructure that hosted the Security Flama was adjusted. Previously, the machines were shut down overnight to reduce energy consumption. However, the pilot subjects mentioned this as problematic since the results of nightly tests could not be imported. Therefore, the shutdown mechanism was deactivated for the evaluation period. The second change affected the slide deck used to introduce the Security Flama to the project team. The subjects of the piloting phase had problems understanding the meaning of the data fields presented for each security finding. Therefore, one slide describing each field's content was added to the slide deck. All other aspects of the evaluation, including the protocol, structure, and study instruments, remained unchanged.

The study itself was conducted between January 9 and April 24, 2023. During the first four weeks solely the quantitative data was collected. In the week of January 30th, the Security Flama was introduced to both project teams. Due to the sickness of one team member, another session in the same week solely for this subject was necessary. The reoccurring interviews started on the week of February 13th and concluded with the final interview on the week of April 24th. Since one developer left Project B in March, this resulted in completing just two instead of six interviews for this subject. Moreover, one subject hesitated to answer the final interview due to a lack of direct interaction with the Security Flama.

Changes to the implementation of the Security Flama are part of its intended usage in practice. Examples include a change in the list of security tools generating reports that necessitated the addition of a new parser. Similarly, changes to the web interface, including bug fixes or additional visual representations, are acceptable, as customization is a crucial requirement for the communication strategy. However, all changes affecting the validity of the evaluation were rejected, including, e.g., changes to the methodology or knowledge base itself. Every shift in the implementation of the Security Flama was tracked and can be found in Table D.1.

7.4.2 Quantitative Results

Eleven distinct development indicators were collected during the evaluation on 16 occasions. In the following, the resulting data is presented for both projects. Since all security activities reporting security findings have been automated in both projects, the number of security activities equals the number of security tools. In Project A, the activities included one Secret Scanning tool, one 3rd Party Component Vulnerability Scanning and SCA tool, and one Code Review tool, summarizing to three security activities in the project. In Project B, the same tools have been employed. However, an additional tool for 3rd Party Component Vulnerability Scanning was added on February 15th after a gap in the testing coverage was identified while reviewing existing security findings in the Security Flama. Therefore, Project B initially utilized three and later four security activities during the

evaluation period.

Regarding the amount of uploaded security reports, parsed security findings, and aggregated security findings, Figure 7.3 depicts the distribution over time. In both projects, the number of security reports added to the knowledge base summed up to 6082 in Project A and 5508 in Project B. Both projects started with more than 1000 security findings initially. The security aggregation reduced this number during the data quality improvement to approximately 32% and 36% of the original amount in both projects. This trend is visible throughout the entire evaluation timeframe.

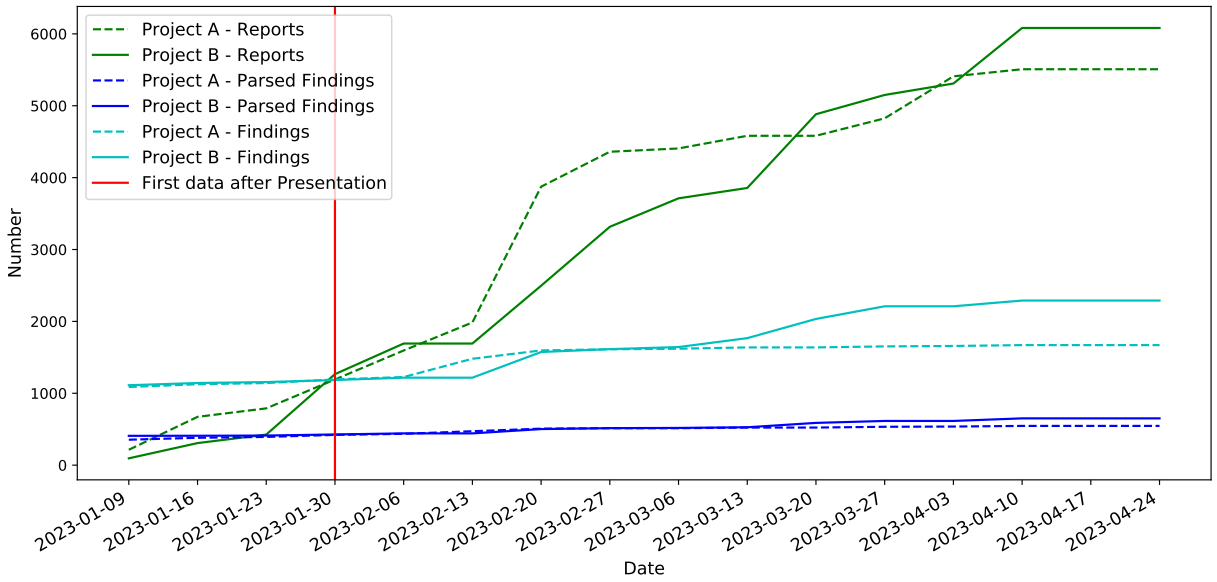

Figure 7.3: Finding Statistics of both Projects

During the evaluation, the observed security findings reached different states in their lifecycle, including ”Disappeared”, ”Open”, ”False Positive”, and ”Accepted”. The latter two states have only been assigned to one project on a single occasion. Therefore, no detailed reporting is reasonable. Figure 7.4 presents the total number of aggregated security findings compared to findings with the ”Open” and ”Disappeared” states in Project A. These statistics are shown throughout the evaluation period. The same information is presented in Figure 7.5 for Project B.

Another measurement was the severity of security findings in both projects. In both projects, only the severity of open findings is reported to list the currently applicable findings instead of already resolved findings. Each finding is clustered into the severity categories ”Critical”, ”High”, ”Medium”, ”Low”, and ”Info”. Figure 7.6 reports this information for Project A and Figure 7.7 for Project B, including the overall number of open findings.

In addition to all the diagrams above, the values collected each week are further reported in the Appendix in Table E.1 for Project A and Table E.2 for Project B. For each date in 2023, the tables contain the number of reports, findings, and aggregated findings in

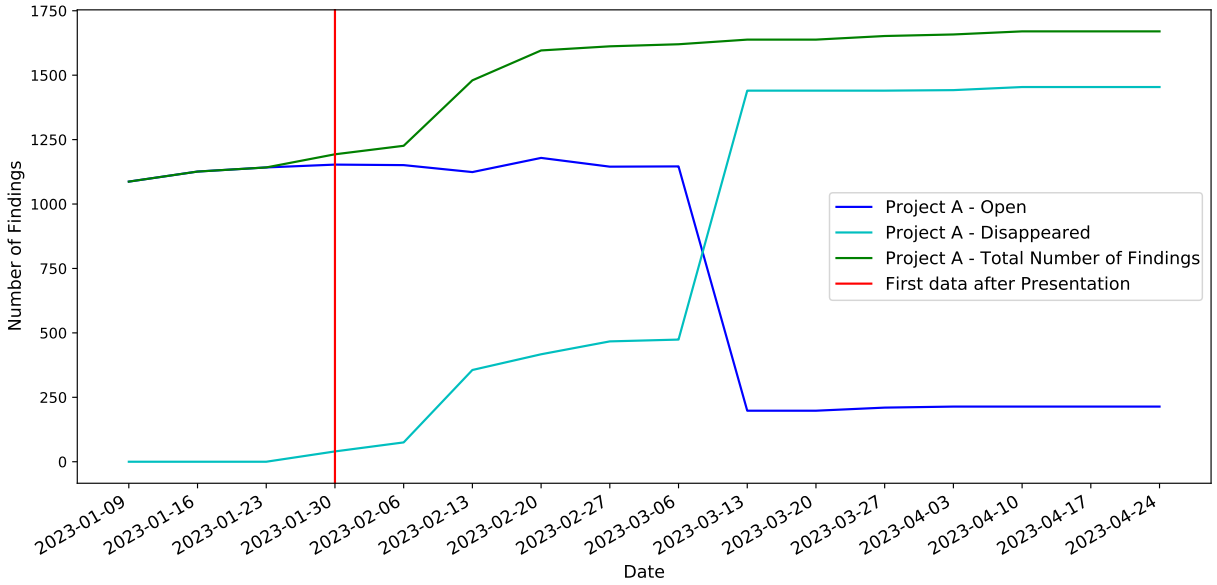

Figure 7.4: Findings Status of Project A

the knowledge base. Moreover, it lists the number of new findings that have not been previously reported and the number of user inputs for the prioritization score. Finally, the number for all findings with a specific severity (Critical, High, Medium, Low, Info) and states (Open, Disappeared, False Positive, Accepted) are presented.

The summary of the finding statistics can also be found in Table 7.7, including the minimal, maximal, and average value for each indicator in both projects. The presentation of new findings in this table excluded the first data point, as every initial finding would be considered new.

Finally, the response time per finding, the user input per week, and the number of interface requests have been collected. However, none of these values provide representative

Table 7.7: Quantitative Statistics per Week

Indicator	**Project A**			**Project B**		
	Min	**Max**	**Avg**	**Min**	**Max**	**Avg**
#Security Activities	3	3	3	3	4	4
#Security Tools	3	3	3	3	4	4
#Reports	0	1889	344	0	1026	380
#Raw Findings	0	1087	104	0	1112	143
#Aggregated Findings	0	354	34	0	408	40
#New Findings	0	36	12.8	0	61	17.2
#User Input	0	2	0.25	0	1	0.125

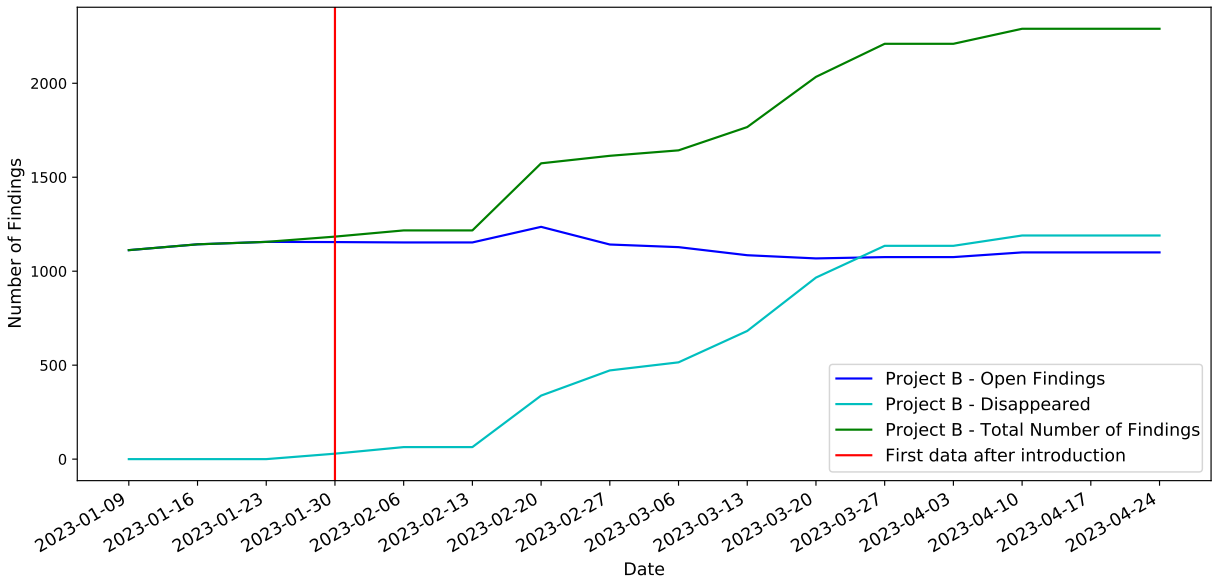

Figure 7.5: Findings Status of Project B

insights into the project. The interface requests were impacted by sessions kept open over a long time and frequent reloads of the interface when new test results were added. This falsifies any conclusion that can be derived from this information. A similar issue affected the recorded response time per finding. The response time did not provide any insights, as the low user input gave no insights on whether a finding was purposefully ignored or overlooked. Therefore, the response time provides no relevant information as well. As depicted in Tables E.1 and E.2, the user input was limited to one priority input added to Project B and two states added to Project A.

7.4.3 Qualitative Results

During the evaluation, five iterations of the reoccurring interview and one final interview were conducted for every team member. Considering that one team member left during the evaluation, the dataset of the qualitative analysis consisted of 38 answered questionnaires. Especially during the first interview sessions, the subjects hesitated to respond to the questionnaire based on the limited time they spent using the Security Flama.

The results of the organizational questions identified our subjects as two project managers, five developers, and one product owner. One project manager simultaneously contributed to the code base, resulting in this double role. The project contribution varied between 70 hours and two hours. The contribution of two hours was only once reported by the product owner, who solely participated in the weekly project team meetings. All participants interacted with the Security Flama at least weekly. During the first week, the interaction in both projects was limited to one access per subject. Afterward, most subjects accessed the Security Flama twice a week in both projects, while one developer in

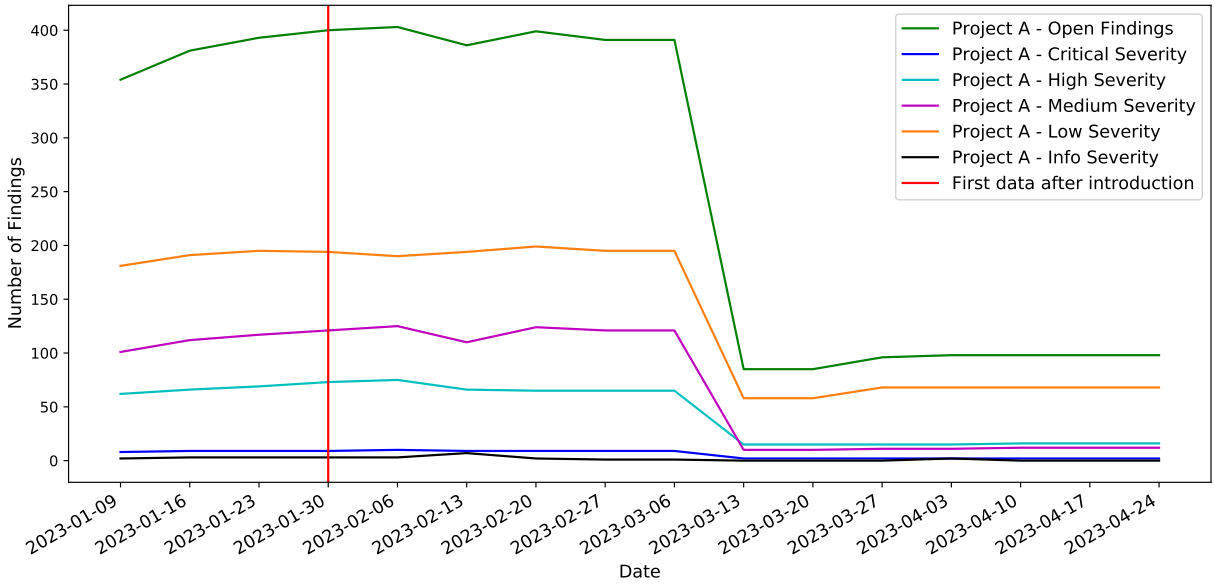

Figure 7.6: Severity of Findings in Project A

each project accessed it more than five times per week. Only the subject with the product owner role did not access the Security Flama weekly but interacted with it passively through other colleagues during the weekly team meetings.

Most closed questions in the reoccurring questionnaire have been consistently answered with ”5-Agree” or ”6-Strong Agree”. The only exceptions included the question referring to the restriction imposed by the methodology and to the transparency of the methodology. All subjects stated that they did not feel restricted by the methodology in their work by responding consistently with ”1-Strong Disagree”. Initially, most subjects also stated that the methodology and its data processing were not transparent and understandable, which improved over time, resulting in general agreement during the last interview.

As part of the data analysis, each answer to an open question was assigned to one of the groups ”benefit”, ”limitation” or ”context”. In this first paragraph, we present the results of the recurring interview. The main benefit the subjects reported was the overview of the currently existing findings in the software product provided by the web interface (mentioned 31 times). In particular, the simplicity of identifying new findings (mentioned 22 times), the aggregation of findings from different locations (17 times), and the common severity scale across various security activities (nine times) were mentioned. Those two developers with the highest interactivity with the Security Flama exclusively gave the last answer. Moreover, the subjects reported that they prepare for the weekly team meetings with the data shown in the Security Flama or utilize it during team meetings to discuss the current status of software security (mentioned 24 times). This was also perceived as notably useful. However, our subjects also mentioned multiple limitations of the methodology. All of them have been related to the current implementation of the methodology. This included 16 bug reports, which have been summarized into five bugs. Moreover, eight features were

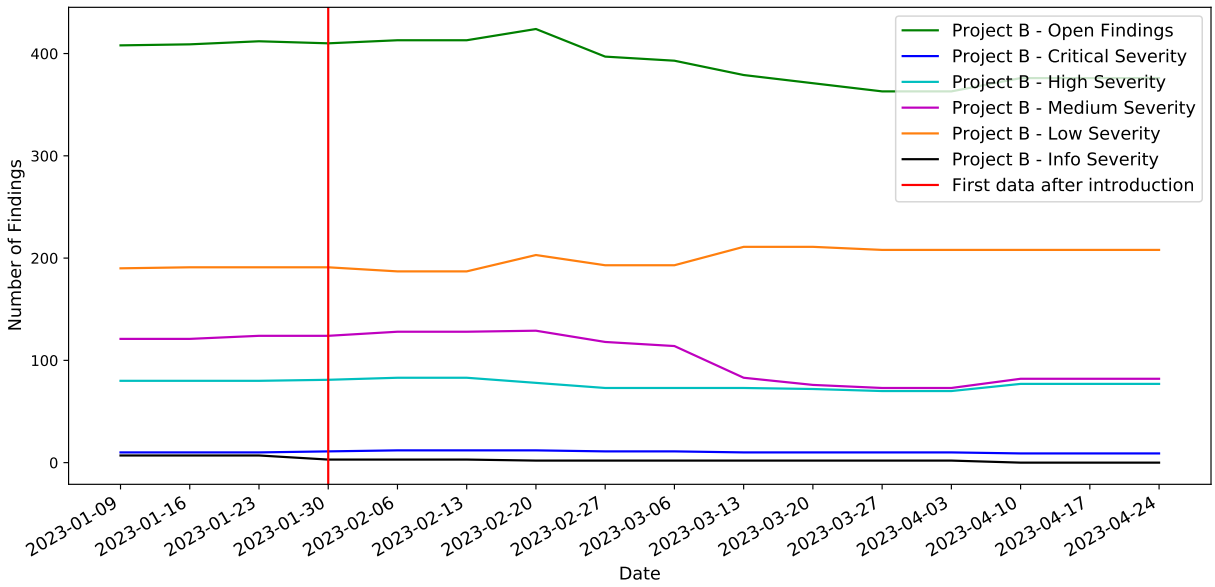

Figure 7.7: Severity of Findings in Project B

requested by our subjects to improve the existing implementation. Table D.1 lists these changes. One feature has been rejected due to the necessary refactoring of the Security Flama. One subject requested to change the state of all security findings with a severity lower than ”High” to ”Delayed”.

Next, we report the final interview results with a focus on the usability, the benefits and limitations of the methodology, and its instance. During the final interview, all subjects claimed they perceived the Security Flama as useful. Various formulations have been used, ranging from ”extremely useful” or ”highly beneficial” to ”essential for the current management of security findings”. Similar to the results of the reoccurring interview, the subjects predominantly mentioned the summary of the current security state based on the security findings as the key aspect for its usability (mentioned five times). The stated reasons include:

- ”It gives an up-to-date insight on the security state”
- ”Newly introduced findings can be easily identified after changes to the code base have been made”
- ”I can check whether my measures fixed the security finding”
- ”Diagrams can be screenshotted into reports”
- ”You can get in insights [that are] otherwise impossible”

The second aspect mentioned in favor of the methodology’s usability was the simple access to the system when compared to the previously existing approach (mentioned three times).

Subjects noted the approach of a central web interface with straightforward navigation as crucial for the system's usability. No subject mentioned aspects with a negative impact on the methodology's usability.

The most commonly mentioned benefit was again the evidence-based overview of the security status of the project given by the Security Flama (mentioned five times). Another aspect also mentioned by all subjects is the ability to impact the security findings management process according to the project's needs. This included integrating reports from additional security activities on demand and adding feedback (user input) to security findings. Moreover, certain procedures have been mentioned as particularly beneficial, including the common data model, the aggregation of security findings, and a common severity scale across all security findings regardless of the source. The various perspectives on the data provided by the web interface were also mentioned as another benefit.

However, limitations about the Security Flama were also reported during the final interviews. These can be clustered into three areas. The first cluster contains all potential improvements to the implementation of the web interface given by the subjects. Six improvements have been suggested, including a dark mode, additional diagrams, a different color scheme with higher contrast, and comfort functions when clicking on diagrams. In addition to improvements to the web interface, subjects also mentioned limitations in the methodology and procedural shortcomings, represented by our second cluster. The first limitation in this cluster is the poor visibility of the data processing. The subjects claimed that it is not immediately obvious how certain data is computed, infringing the trust in the results. Even though this improved over time, subjects mentioned that it should be apparent from the beginning. Another subject stated that communication about the long-term success based on performance indicators was missing. According to the feedback, comparing how many findings have been solved, the number of new findings, and the sum of all still existing findings over the last month would be detrimental to the team's motivation. Another subject supposedly identified a bug in the finding state system, where findings did not reach the "Disappeared" state. This represented a limitation, as developers have been using this function to identify the impact of their mitigation. However, the root cause of this limitation was that no new report had been uploaded as no re-test had been conducted, further confirming the poor transparency of the methodology's functionality. Two subjects also considered the approach of using numbers for adding priority scores as too complex and, therefore, a limitation of the methodology. The last cluster deals with limitations affecting the project when utilizing the Security Flama. One subject described the effort to manage security findings as "exhaustive". The subject further mentioned that the interaction and management of security findings were often forgotten and proposed that a proactive communication strategy with emails for high-severity findings might be beneficial. Finally, two subjects identified a problem with managing tasks related to mitigating security findings. Since this project managed the security findings in the backlog to prioritize them against other tasks, one subject perceived the additional prioritization as "duplicate housekeeping".

7.5 Discussion

The evaluation aims to identify the impact of using the methodology on industrial software development projects that utilize modern software development approaches. In this section, the results of the evaluation are discussed. We start with general insights acquired during the evaluation, continue by answering each research question, and derive conclusions in the end.

7.5.1 General Insights

In addition to answering our research questions, the evaluation gave us multiple insights into security findings management in industrial practice. In this subsection, we present the additional insights and context information necessary to interpret the evaluation results.

Firstly, each project had one subject that accessed the Security Flama more frequently than all other subjects combined, indicating that each project team had one member with a particular security responsibility or interest. Another insight showed that not every subject accessed the methodology using the web interface. Instead, one team member utilized other subjects during appointments to discuss security findings and task planning. This subject represents a currently unconsidered type of user, either missing the time or interest to work with security findings directly in the web interface.

Moreover, the evaluation has shown that both projects validate security findings after a preliminary prioritization. In contrast to the initially designed process, findings are prioritized by severity before they are investigated for False Positives or not-applicable findings. Hence, a manual process step currently not considered by our methodology exists that could be implemented as part of the security finding analysis.

To provide context to the data presented in Tables E.1 and E.2, readers must be aware of public holidays in Germany. Due to the Easter Holidays, neither team conducted code changes between ”2023-04-10” and ”2023-04-24”. Therefore, the reported numbers persisted during that timeframe since no code changes resulted in no additional or mitigated findings. A lack of changes in the numbers can also be observed whenever code changes and tests are exclusively performed on development branches, as only the main branch is regarded as input for the Security Flama.

Figure 7.3 must be interpreted, minding the fact that the Security Flama solely collects data but never deletes it proactively. Therefore, the number of reports and security findings can only increase since the history of each finding is tracked, but findings that are not found anymore are not eliminated from the statistics.

To understand the relation between a finding and its state, it is important to mention that each location a finding is found at has its separate state. Due to the aggregation, each finding can be located at multiple places and, therefore, may contain a different state at each location. Thus, the sum of all status values is larger than the number of aggregated security findings.

7.5.2 RQ1: How does the usage of the methodology reflect in development process indicators?

The evaluation results suggest different impacts of the methodology usage on the development indicators.

First, the results of Figure 7.3 show that the aggregation and correlation of security findings reduce the overall number of findings by approximately two-thirds in both projects. Hence, the methodology successfully reduced the number of finding. Second, Figure 7.4 and Figure 7.5 both indicate that the overall number of open findings was reduced in both projects. However, only the results of Project A showed a considerable reduction of security findings. In the week of March 6, this team responded to multiple security findings based on the insights provided by the Security Flama, resulting in a 78% reduction of security findings. Moreover, the number of findings that disappeared increased in both projects over time, indicating that security findings have either been consistently fixed or the location of findings changed. Based on the input provided by the subjects, both factors impacted the number of disappeared findings. Therefore, the increasing number of findings is due to the refactoring of components, resulting in new finding locations and consistently new findings being reported by the security activities.

Figure 7.6 and Figure 7.7 show that no impact on the severity distribution of findings can be identified. Therefore, there was no apparent prioritization of exclusively critical findings. However, in some instances, mitigating one finding affected multiple others as well. An example could be an update of third-party components, where the same component can be affected by various findings with different severities. Hence, we cannot derive conclusions about the project prioritization approach.

Unfortunately, no conclusions can be drawn from the data reported about the security findings state due to the low numbers of states assigned. Similarly, the response time, the user input, and the number of interface requests did not provide reliable data for any inferences. We conclude that the usage of the methodology positively correlates with the number of security findings. This reduction was visible in Figure 7.4 for one of the projects.

7.5.3 RQ2: Is the methodology perceived as useful by the project team for managing security findings?

For the second research question, we analyzed any response related to the methodology's usefulness to the project team. During the interviews, two aspects have been mentioned in favor of our methodology's usefulness. Subjects mentioned that the simple access to information about security findings was perceived as useful. Considering that both projects previously used the CI environment to analyze formatted reports, we can conclude that a central and easy-to-access dashboard is preferred in practice. The second aspect perceived as particularly useful was the overview provided by the Security Flama. However, the reasons differentiated between roles. While developers mentioned the ability to identify new and disappeared findings instantly, the project managers preferred the high-level overview. Therefore, the role-tailored presentation of security findings has shown to be useful in

practice.

Even though no subject reported any aspect negatively impacting the usefulness, the quantitative data on the amount of user input indicates that there might be disrupting factors. With three instances of user input being added in both projects, the amount of user input is remarkably low. Explanations for this circumstance can be found in the results of the interview sessions. The absence of user input for prioritizing security findings can be explained by the effort and complexity of adding input to the Security Flama using numeric values and the strategy in both projects, in which only high and critical findings are considered for a response. The second explanation made prioritizing security findings unnecessary and is highly project-specific, as this binary decision is uncommon in most projects. However, it also shows that the usability of the methodology in this domain cannot be considered entirely beneficial for both projects. On the other hand, the low number of user inputs for assigning security finding states was explained by the need to confirm that a finding was not reported again. Whenever a finding reached the "Disappeared" state, subjects did not see the necessity of adding a new state. Furthermore, the assignment of finding states was affected by the manual effort to upload them using the webinterface. One subject wished for a bulk-update feature of findings based on the severity. This wish is rooted in the binary prioritization strategy of the projects and would have allowed the project to move all findings with medium or lower severity to the state "Accepted". As discussed previously, this change was rejected during the evaluation period but would have affected 325 aggregated findings. Therefore, redesigning the implementation of prioritization and status tracking seems beneficial to improve project acceptance.

We conclude that our subjects perceived the methodology as highly useful. However, we believe that restrictions apply to the usefulness of all manual interactions with the Security Flama. As mainly the automated aspects of the methodology were employed, we assume that the perception of usefulness relies on the ability to avoid interaction with inconvenient parts of the methodology like manual prioritization.

7.5.4 RQ3: Which benefits does the using the methodology bring?

To answer the third research question, we collected all benefits mentioned during the interviews and correlated them with quantitatively collected data. The foremost benefit identified by all subjects was the evidence-based visibility provided by the methodology. However, details on what exactly was perceived as beneficial differentiated between roles in each project. Especially those two subjects that accessed the Security Flama most reported benefits, including the common severity scale, the identification of new findings, and the aggregation of security findings. As part of this overview, one project identified a coverage gap in their testing strategy, which was also perceived as beneficial and attributed to the visibility of the current security status. However, the gap could only be determined by combining the knowledge of one developer who knew that certain findings existed with the overview supplied by Security Flama. Hence, one benefit is communicating relevant information for its manual interpretation, combination with context information, and collaboration between project team members.

Another benefit stated during the interviews is the support for the entire security findings management process. In both projects, all team members agreed that the Security Flama covers all aspects of the security findings management process. However, this statement is only valid under two constraints. First, the subjects could only identify whether the methodology supports the process activities they were aware of. Hence, the claim can not be generalized as subjects with another background might perceive different aspects as relevant. Second, the relevant parts of the security findings management process are also project-specific. For example, both projects did not require the prioritization of security findings, as their binary strategy for selecting which findings should be addressed was sufficient for their context. Hence, we can only claim that it addressed the aspects relevant to the respective projects.

Even though almost no user input was given to the methodology due to the project-specific constraints and the shortcomings in the implementation of the web interface, both projects reduced their number of open findings. Therefore, the methodology impacts this indicator, regardless of how mature the project or the implementation is. Hence, we consider the ability to customize the methodology according to the project's needs and still conduct a successful security findings management as a benefit of the methodology. We conclude that the key benefits of the methodology are the visibility of the current security status of the software, the ability to customize the methodology according to the project's demand, and the enablement of the project team to collaborate on security findings.

7.5.5 RQ4: Which limitations exist when using the methodology?

The fourth research question was answered identically to the third one by collecting all limitations mentioned during interviews and combining them with quantitative data. The most mentioned limitations of the methodology were related to the implementation of the web interface. Especially in the final interview, most shortcomings have been associated with the implementation of the methodology instead of the methodology itself. This includes the design decision to use numerical values for the prioritization, the lack of proactive communication, some visual and functional improvements for the web interface, and the ability to bulk-assign finding states. Hence, the current implementation of the methodology must be improved based on practitioner feedback to mitigate the current shortcomings.

Another frequently mentioned limitation is the transparency of how the methodology and its implementation process the security reports. Even though our subjects reported that their knowledge about how the Security Flama processes data improved over time, the lack of transparency is not acceptable to achieve the goals of our methodology. This absence of insight could manifest in users of the methodology as trust issues. Even though no subject showed a lack of trust during the interviews, this could have materialized in acceptance issues or supplementary activities, like analyzing the plain reports. A hint that this might have been the case can be found in the low rates of user input due to a lack of knowledge of how the user input is applied in practice. Therefore, the communication

strategy of the methodology must be refined to present how information is processed, and users may impact the processing.

Another limitation of the communication strategy is the lack of long-term success diagrams. During the final interview, one subject wished for a visual representation of the long-term security findings management success, including all mitigated findings to motivate team members. As we did not consider the importance of team member motivation during the design of the solution approach, this represents a gap in the communication strategy. Consequently, the motivation of team members must be considered when refining the communication approach.

One shortcoming mentioned by subjects of both projects was that the prioritization was not beneficial to them, as their security strategy solely focused on high and critical findings. Since this strategy is uncommon in industry, we acknowledge the feedback but believe the limitation cannot be generalized. However, another limitation derived from the feedback on the prioritization was the lack of backlog integration. The subjects mentioned that they prioritize the finding response in their regular backlog against other functional requirements and hence perceive the prioritization in the Security Flama as "duplicate housekeeping". Thus, the methodology must integrate better with the backlog to close the feedback loop for security findings. The subjects attributed the scarce usage of user input during the evaluation period to the complexity of interacting with the user interface. Moreover, the subjects mentioned the effort to regularly access the web interface as exhausting and recommended a proactive strategy instead. Therefore, we see the potential to further reduce the effort for managing security findings in practice. We conclude that especially the communication strategy and its implementation are the limiting factors of the methodology.

7.5.6 Threats to Validity

Similar to all case studies, our results are also constrained by threats to their validity [155]. In this subsection, we present and discuss all threats to the validity of the evaluation, structured according to the type of infringed validity.

First, the case study was conducted with the Security Flama, representing an instance of the methodology. Therefore, any implementation shortcomings might have been attributed to the methodology itself. Even though this infringes the internal validity of the results, automation is a central requirement for utilizing the methodology in modern software development, making it impossible to evaluate it without instantiation. Another threat to the internal validity is the previous knowledge of our subjects in security findings management. During the interviews, some subjects indicated they were inexperienced with managing security findings. One consequence this lack of knowledge might have introduced is the incorrect attribution of benefits related to the source data being instead associated with the methodology. An example would be that information originating from high-quality security testing is being reported as a benefit of the Security Flama instead of being attributed to the testing strategy. Conversely, limitations introduced by a suboptimal security testing strategy might be interpreted as limitations of the methodology.

This threat affects not only the internal validity but the construct validity as well. Subjects with different backgrounds might interpret the questions differently based on their experience with security findings management in modern industrial software development. Regardless, this represents industrial practice correctly, as not every project member can be considered a security expert. Therefore, we accept this threat to achieve a realistic case study. A similar threat related to the implementation of the methodology is the impact of factors that are currently not considered. An example is the design of the web interface. Whether the web interface follows the corporate design guidelines might impact the perceived usefulness, especially during the initial onboarding when features are not known yet. However, these aspects can only hardly be measured in practice.

In addition to threats to the internal and construct validity, the case study design introduces additional threats to the external validity. Since we conducted the evaluation in an industrial enterprise, the ability to transfer our results to other companies and projects might be infringed. The first factor affecting the external validity is the number of projects and subjects interviewed. By assessing only two projects with a limited number of project team members, the results might not be transferable to other projects. This affects the type and amount of findings in particular. Security findings in software development projects located in different domains or markets could vary in amount, type, severity, or relevance. Therefore, we cannot derive that the behavior reported in this case study represents projects in other domains equally. However, this is a common threat when conducting case studies in industrial practice. Moreover, the target of the evaluation was an analysis of our methodology for industrial projects, which can be derived from the results nevertheless. Another threat is the time frame of the evaluation. Long-term data on the usage of the methodology and its impact on the project could provide further insights into the benefits and limitations in the long run. However, this would complicate maintaining a static case for the study, as the methodology is intended to be customized and continuously improved based on ongoing research. Anyway, we believe that a long-term evaluation with a less strict case study design can provide valuable insights in the future.

7.5.7 Summary of Conclusions

Finally, we present the summary of conclusions derived from the discussions in this section:

1. The impact of the Security Flama depends on the acceptance of the platform in the team and the importance of security in the project in general

2. One project drastically reduced the number of findings based on insights provided by the Security Flama

3. Projects conducted a preliminary prioritization of security findings based on severity before analyzing them

4. Even though perceived as useful, the current effort for user input and methodology interaction diminishes the usefulness of the methodology

5. Evidence-based overview, collaboration encouragement, and project-specific customization are key benefits of the methodology

6. The communication strategy and implementation are the biggest drawbacks of the methodology

The validity of these conclusions is constrained by the threats discussed in Subsection 7.5.6.

Even though the evaluation has not shown that our methodology is the optimal solution for security findings management in modern industrial software development, it can be considered a feasible solution as it is field-tested and grounded [154]. The results showed that it is perceived as useful and beneficial for managing security findings in the analyzed modern industrial software development projects, with limitations affecting the communication strategy and implementation.

Chapter 8

Conclusion and Future Work

The importance of managing potential security shortcomings of software appears obvious when building products in an industrial domain with high security demands. This circumstance is confirmed by various norms, standards, models, and best practices, requesting that the management of security findings is an obligatory process in software development projects. This process consists of multiple activities that must be conducted to manage security findings and their inherent security risk successfully. The execution of these activities is constrained by the principles and strategies of the employed software development methodology. With trends like DevOps or Agile software development being commonly followed in modern industrial software engineering, the management of security findings in such projects has to address the challenges of both domains: security and software engineering.

Summary of the Thesis

This thesis explored the problems of and solutions to the security findings management in modern industrial software development projects and presented a methodology addressing the challenges of conducting traditional findings management procedures in a modernized development setting. Following the principles of Design Science Research, this thesis splits the problem of managing security findings in modern industrial software development projects into multiple sub-problems by conceptualizing the overall problem instance and structuring the identified challenges according to their occurrence in the security findings management process. Initially, the fundamental problems are conceptualized by collecting requirements for the security findings management from industrial norms, standards, and models. From 269 analyzed documents, 21 contributed 35 distinct requirements clustered into four problem instance areas to be addressed. These problem instances were constrained by 12 principles, derived from value propositions and principles of modern software development.

The first problem instance area addressed the challenges of following modern software development principles when performing security findings management. As a solution approach, a semantic knowledge base consisting of belief, rules, and queries was proposed to serve as a platform for all consecutive solution approaches supporting automation, fast feed-

back, and continuous modular data processing. The second problem instance area covered the shortcomings of the source data provided by security activities during the development lifecycle. To cope with multiple challenges in this area, the solution approach includes a data model for security findings, a multi-staged deduplication and aggregation scheme, and the project-dependent context enrichment of security findings. The third problem instance area addresses the challenges related to analyzing security findings, their tracking, and responses to them. The proposed solution approach includes history and state tracking, documentation support for manually conducted analysis, prioritization using formalized project input, and guidance for responding to security findings aligned with common risk management standards. The fourth problem instance area includes the communication between the security findings management process and the project's stakeholders. The solution approach to ensure an efficient communication of data arising from previously proposed solution approaches relies on a communication strategy providing a text-based and visual interface for project stakeholders. Each interface consists of multiple presentations customized to the goals of each stakeholder.

Finally, the solution approaches proposed in this thesis were implemented as overarching solution instance. This solution instance was empirically evaluated in ongoing industrial software development projects that follow modern software development principles. The evaluation results indicated the positive impact of the proposed methodology and confirmed it as a feasible solution instance for the underlying problem instance, thus concluding the Design Science Research approach.

Directions for Future Work

Throughout the thesis, additional potential for future research has been discovered that could not be explored as part of this thesis. Even though most topics sparked interest and indicated their relevance for industrial practice, they could not be included in our research due to the scope of the thesis or relevance to the investigated problem instance. In this paragraph, we want to present the directions for potential future research.

The first potential future research is the automated identification of False Positives in security reports. During our investigation, we found multiple proposals for determining False Positives automatically. However, they have been discarded due to the unacceptable False Positive rates. Therefore, individual research focusing on False Positive identification by employing the data available throughout the security findings management process might yield better results. The impact of an automated False Positive identification would be tremendous on industrial practice, making this a worthwhile future research.

Another direction is the construction of a recommender system for the security findings management process and subsequent activities. This area was preliminarily explored in this thesis, but further research was discarded due to the presumably lower impact on industrial practice than other areas. However, the evaluation has proven the importance of an understandable and straightforward communication strategy, making this an interesting topic for further exploration.

The structured investigation of security findings communication in academia and in-

dustrial practice represents the third direction for future research. During the evaluation, we discovered the importance of a distinguished security findings communication and challenges if an implementation lacks these attributes. Hence, further research on principles for creating an appropriate presentation of security findings data could further improve the methodology and its instantiation.

Finally, additional evaluations could provide insights into the impact of the methodology under distinct circumstances. First, the effect of the methodology on projects not following modern software development principles could deliver information about the particularities of our design. Even though the results could infringe on our initial claim, they could also broaden the area of application for our methodology. Other promising evaluations would be stricter or less strict evaluation designs. By eliminating the regular interviews with the project team and instead entirely focusing on quantitative data, we could achieve a more realistic setting to observe the long-term impact of the methodology on industrial practice. Switching directions, we could tighten the conditions for the security findings management process for the evaluation subjects, necessitating, e.g., that each finding must be reviewed and closed after disappearing. Using quantitative and qualitative data collection approaches, we could derive whether the methodology's customizability is beneficial or damaging to the software product's security.

Even though this thesis has contributed a feasible solution for the management of security findings in modern industrial software development projects, thus fulfilling its initial target, its domain yields further challenges worth exploring in the future.

Appendix A

Analyzed Documents for State of Industrial Practice

Table A list all documents considered during the document review of Section 2.2. It presents the name of the document, whether it was included into our analysis, why it was excluded, which of the three sources it originated from and at which point it was added to the list of analyzed documents.

Document Name	**Incl.**	**Excl. Criteria**	**[147]**	**[175]**	**Partner**	**Addition**
OWASP SAMM v2	Yes	N/A		True	True	Initial
BSIMM 12	Yes	N/A		True	True	Initial
SAFECode 2018	Yes	N/A		True	True	Initial
IEC 62443-4-1	Yes	N/A	True	True	True	Initial
NIST SP 800-218	Yes	N/A	True		True	Initial
NIST SP 800-53 Rev. 5	Yes	N/A	True	True	True	Initial
COBIT 2019 Framework: Governance and Management Objectives	Yes	N/A	True			Initial
NIST SP 800-160 Vol. 1 Rev. 1	Yes	N/A	True		True	Initial
ISO/IEC 15408-3:2008	Yes	N/A		True		Initial
NIST SP 800-181 Rev. 1	Yes	N/A	True		True	Initial
ISO/IEC 27002:2013	Yes	N/A	True	True	True	Initial
BSI IT Grundschutz Kompendium 2022	Yes	N/A	True		True	Initial
NIST CSF v1.1	Yes	N/A		True	True	Initial
IEC 62443-2-4	Yes	N/A	True		True	Initial
IEC 62443-3-2	Yes	N/A	True		True	Initial
ISO/IEC 27005:2018	Yes	N/A	True	True	True	Initial
IEC 62443-2-1	Yes	N/A	True		True	Initial
ISO/IEC 15026-4:2021	Yes	N/A		True		Initial
NIST SP 800-37 Rev. 2	Yes	N/A	True	True	True	Initial
IEC 62443-1-1	Yes	N/A	True		True	Initial
IEC 62443-2-3	Yes	N/A	True		True	Initial
IEC 62443-3-1	Yes	N/A	True		True	Initial
IEC 62443-3-3	Yes	N/A	True		True	Initial
IEC 62443-4-2	Yes	N/A	True		True	Initial

FIPS 140-3	Yes	N/A	True			Initial
OWASP Application Security Verification Standard 4.0.3	Yes	N/A		True		Initial
Common Criteria Part1 v3.1r5	Yes	N/A		True		Initial
Common Criteria Part2 v3.1r5	Yes	N/A		True		Initial
Common Criteria Part3 v3.1r5	Yes	N/A		True		Initial
The TOGAF Standard, 10th Edition (Part 0-5)	Yes	N/A		True		Initial
COBIT 2019 Framework: Introduction and Methodology	Yes	N/A	True			Initial
COBIT 2019 Design Guide: Designing an Information and Technology Governance Solution	Yes	N/A	True			Initial
COBIT 2019 Implementation Guide: Implementing and Optimizing an Information and Technology Governance Solution	Yes	N/A	True			Initial
ISO/IEC 15026-1:2019	Yes	N/A		True		Initial
ISO/IEC 15026-2:2011	Yes	N/A		True		Initial
ISO/IEC 15026-3:2015	Yes	N/A		True		Initial
ISO/IEC/IEEE 41062:2019	Yes	N/A		True		Initial
ISO/IEC 15408-1:2009	Yes	N/A		True		Initial
ISO/IEC 15408-2:2008	Yes	N/A		True		Initial
SANS (Institute)	Yes	N/A		True		Initial
IEC/TR 61508-0:2005-10	Yes	N/A		True		Initial
IEC 61508-1:2010	Yes	N/A		True		Initial
IEC 61508-2:2010	Yes	N/A		True		Initial
IEC 61508-3:2010	Yes	N/A		True		Initial
IEC 61508-4:2010	Yes	N/A		True		Initial
IEC 61508-5:2010	Yes	N/A		True		Initial
IEC 61508-6:2010	Yes	N/A		True		Initial
IEC 61508-7:2010	Yes	N/A		True		Initial
ISO/IEC 27000:2018	Yes	N/A	True		True	Initial
ISO/IEC 27001:2013	Yes	N/A	True	True	True	Initial
ISO/IEC 27003:2017	Yes	N/A	True		True	Initial
ISO/IEC 27004:2016	Yes	N/A	True		True	Initial
ISO/IEC 27006:2015	Yes	N/A	True	True	True	Initial
ISO/IEC 27007:2020	Yes	N/A	True	True	True	Initial
ISO/IEC TS 27008:2019	Yes	N/A	True		True	Initial
ISO/IEC 27009:2020	Yes	N/A	True		True	Initial
ISO/IEC 27010:2015	Yes	N/A	True		True	Initial
ISO/IEC 27013:2021	Yes	N/A	True		True	Initial
ISO/IEC 27014:2020	Yes	N/A	True		True	Initial
ISO/IEC TR 27016:2014	Yes	N/A	True		True	Initial
ISO/IEC 27017:2015	Yes	N/A	True		True	Initial
ISO/IEC 27018:2019	Yes	N/A	True		True	Initial
ISO/IEC 27021:2017	Yes	N/A	True		True	Initial
ISO/IEC TS 27022:2021	Yes	N/A	True		True	Initial
ISO/IEC 27031:2011	Yes	N/A	True		True	Initial
ISO/IEC 27032:2012	Yes	N/A	True		True	Initial
ISO/IEC 27033-1:2015	Yes	N/A	True		True	Initial

ISO/IEC 27033-2:2012	Yes	N/A	True		True	Initial
ISO/IEC 27033-3:2010	Yes	N/A	True		True	Initial
ISO/IEC 27033-4:2014	Yes	N/A	True		True	Initial
ISO/IEC 27033-5:2013	Yes	N/A	True		True	Initial
ISO/IEC 27033-6:2016	Yes	N/A	True		True	Initial
ISO/IEC 27034-1:2011	Yes	N/A	True		True	Initial
ISO/IEC 27034-2:2015	Yes	N/A	True		True	Initial
ISO/IEC 27034-3:2018	Yes	N/A	True		True	Initial
ISO/IEC 27034-5:2017	Yes	N/A	True		True	Initial
ISO/IEC TS 27034-5-1:2018	Yes	N/A	True		True	Initial
ISO/IEC 27034-6:2016	Yes	N/A	True		True	Initial
ISO/IEC 27034-7:2018	Yes	N/A	True		True	Initial
ISO/IEC 27035-1:2016	Yes	N/A	True		True	Initial
ISO/IEC 27035-2:2016	Yes	N/A	True		True	Initial
ISO/IEC 27035-3:2020	Yes	N/A	True		True	Initial
ISO/IEC 27036-1:2021	Yes	N/A	True		True	Initial
ISO/IEC 27036-2:2014	Yes	N/A	True		True	Initial
ISO/IEC 27036-3:2013	Yes	N/A	True		True	Initial
ISO/IEC 27036-4:2016	Yes	N/A	True		True	Initial
ISO/IEC 27037:2012	Yes	N/A	True		True	Initial
ISO/IEC 27038:2014	Yes	N/A	True		True	Initial
ISO/IEC 27039:2015	Yes	N/A	True		True	Initial
ISO/IEC 27040:2015	Yes	N/A	True		True	Initial
ISO/IEC 27041:2015	Yes	N/A	True		True	Initial
ISO/IEC 27042:2015	Yes	N/A	True		True	Initial
ISO/IEC 27043:2015	Yes	N/A	True		True	Initial
ISO/IEC 27050-1:2019	Yes	N/A	True		True	Initial
ISO/IEC 27050-2:2018	Yes	N/A	True		True	Initial
ISO/IEC 27050-3:2020	Yes	N/A	True		True	Initial
ISO/IEC 27050-4:2021	Yes	N/A	True		True	Initial
ISO/IEC TS 27110:2021	Yes	N/A	True		True	Initial
ISO/IEC 27701:2019	Yes	N/A	True		True	Initial
NIST SP 800-12 Rev. 1	Yes	N/A	True	True	True	Initial
NIST SP 800-16	Yes	N/A	True		True	Initial
NIST SP 800-18 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-22 Rev. 1a	Yes	N/A	True		True	Initial
NIST SP 800-28 Version 2	Yes	N/A	True		True	Initial
NIST SP 800-30 Rev. 1	Yes	N/A	True	True	True	Initial
NIST SP 800-34 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-35	Yes	N/A	True		True	Initial
NIST SP 800-38A	Yes	N/A	True		True	Initial
NIST SP 800-38B	Yes	N/A	True		True	Initial
NIST SP 800-38C	Yes	N/A	True		True	Initial
NIST SP 800-38D	Yes	N/A	True		True	Initial
NIST SP 800-38E	Yes	N/A	True		True	Initial
NIST SP 800-38F	Yes	N/A	True		True	Initial
NIST SP 800-38G	Yes	N/A	True		True	Initial
NIST SP 800-39	Yes	N/A	True	True	True	Initial
NIST SP 800-40 Rev. 4	Yes	N/A	True		True	Initial
NIST SP 800-41 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-44 Version 2	Yes	N/A	True		True	Initial

NIST SP 800-45 Version 2	Yes	N/A	True		True	Initial
NIST SP 800-46 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-47 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-49	Yes	N/A	True		True	Initial
NIST SP 800-50	Yes	N/A	True		True	Initial
NIST SP 800-51 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-52 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-53A Rev. 5	Yes	N/A	True	True	True	Initial
NIST SP 800-53B	Yes	N/A	True	True	True	Initial
NIST SP 800-55 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-56A Rev. 3	Yes	N/A	True		True	Initial
NIST SP 800-56B Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-56C Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-57 Part 1 Rev. 5	Yes	N/A	True		True	Initial
NIST SP 800-57 Part 2 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-57 Part 3 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-58	Yes	N/A	True		True	Initial
NIST SP 800-59	Yes	N/A	True		True	Initial
NIST SP 800-60 Vol. 1 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-60 Vol. 2 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-61 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-63A	Yes	N/A	True		True	Initial
NIST SP 800-63B	Yes	N/A	True		True	Initial
NIST SP 800-63C	Yes	N/A	True		True	Initial
NIST SP 800-63-3	Yes	N/A	True		True	Initial
NIST SP 800-66 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-67 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-70 Rev. 4	Yes	N/A	True		True	Initial
NIST SP 800-72	Yes	N/A	True		True	Initial
NIST SP 800-73-4	Yes	N/A	True		True	Initial
NIST SP 800-76-2	Yes	N/A	True		True	Initial
NIST SP 800-77 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-78-4	Yes	N/A	True		True	Initial
NIST SP 800-79-2	Yes	N/A	True		True	Initial
NIST SP 800-81-2	Yes	N/A	True		True	Initial
NIST SP 800-82 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-83 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-84	Yes	N/A	True		True	Initial
NIST SP 800-85A-4	Yes	N/A	True		True	Initial
NIST SP 800-85B	Yes	N/A	True		True	Initial
NIST SP 800-86	Yes	N/A	True		True	Initial
NIST SP 800-87 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-88 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-89	Yes	N/A	True		True	Initial
NIST SP 800-90A Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-90B	Yes	N/A	True		True	Initial
NIST SP 800-92	Yes	N/A	True		True	Initial
NIST SP 800-94	Yes	N/A	True		True	Initial
NIST SP 800-95	Yes	N/A	True		True	Initial
NIST SP 800-96	Yes	N/A	True		True	Initial
NIST SP 800-97	Yes	N/A	True		True	Initial

NIST SP 800-98	Yes	N/A	True		True	Initial
NIST SP 800-100	Yes	N/A	True		True	Initial
NIST SP 800-101 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-102	Yes	N/A	True		True	Initial
NIST SP 800-107 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-108 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-111	Yes	N/A	True		True	Initial
NIST SP 800-113	Yes	N/A	True		True	Initial
NIST SP 800-114 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-115	Yes	N/A	True		True	Initial
NIST SP 800-116 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-119	Yes	N/A	True		True	Initial
NIST SP 800-121 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-122	Yes	N/A	True		True	Initial
NIST SP 800-123	Yes	N/A	True		True	Initial
NIST SP 800-124 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-125	Yes	N/A	True		True	Initial
NIST SP 800-125A Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-125B	Yes	N/A	True		True	Initial
NIST SP 800-126 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-126A	Yes	N/A	True		True	Initial
NIST SP 800-126 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-126 Rev. 3	Yes	N/A	True		True	Initial
NIST SP 800-128	Yes	N/A	True		True	Initial
NIST SP 800-130	Yes	N/A	True		True	Initial
NIST SP 800-131A Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-132	Yes	N/A	True		True	Initial
NIST SP 800-133 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-135 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-137	Yes	N/A	True		True	Initial
NIST SP 800-137A	Yes	N/A	True		True	Initial
NIST SP 800-140	Yes	N/A	True		True	Initial
NIST SP 800-140A	Yes	N/A	True		True	Initial
NIST SP 800-140B	Yes	N/A	True		True	Initial
NIST SP 800-140C Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-140D Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-140E	Yes	N/A	True		True	Initial
NIST SP 800-140F	Yes	N/A	True		True	Initial
NIST SP 800-142	Yes	N/A	True		True	Initial
NIST SP 800-144	Yes	N/A	True		True	Initial
NIST SP 800-145	Yes	N/A	True		True	Initial
NIST SP 800-146	Yes	N/A	True		True	Initial
NIST SP 800-147	Yes	N/A	True		True	Initial
NIST SP 800-147B	Yes	N/A	True		True	Initial
NIST SP 800-150	Yes	N/A	True		True	Initial
NIST SP 800-152	Yes	N/A	True		True	Initial
NIST SP 800-153	Yes	N/A	True		True	Initial
NIST SP 800-156	Yes	N/A	True		True	Initial
NIST SP 800-157	Yes	N/A	True		True	Initial
NIST SP 800-160 Vol. 2 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-161 Rev. 1	Yes	N/A	True		True	Initial

NIST SP 800-162	Yes	N/A	True		True	Initial
NIST SP 800-163 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-166	Yes	N/A	True		True	Initial
NIST SP 800-167	Yes	N/A	True		True	Initial
NIST SP 800-168	Yes	N/A	True		True	Initial
NIST SP 800-171A	Yes	N/A	True		True	Initial
NIST SP 800-171 Rev. 2	Yes	N/A	True		True	Initial
NIST SP 800-172	Yes	N/A	True		True	Initial
NIST SP 800-172A	Yes	N/A	True		True	Initial
NIST SP 800-175A	Yes	N/A	True		True	Initial
NIST SP 800-175B Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-177 Rev. 1	Yes	N/A	True		True	Initial
NIST SP 800-178	Yes	N/A	True		True	Initial
NIST SP 800-183	Yes	N/A	True		True	Initial
NIST SP 800-184	Yes	N/A	True		True	Initial
NIST SP 800-185	Yes	N/A	True		True	Initial
NIST SP 800-187	Yes	N/A	True		True	Initial
NIST SP 800-189	Yes	N/A	True		True	Initial
NIST SP 800-190	Yes	N/A	True		True	Initial
NIST SP 800-192	Yes	N/A	True		True	Initial
NIST SP 800-193	Yes	N/A	True		True	Initial
NIST SP 800-202	Yes	N/A	True		True	Initial
NIST SP 800-204	Yes	N/A	True		True	Initial
NIST SP 800-204A	Yes	N/A	True		True	Initial
NIST SP 800-204B	Yes	N/A	True		True	Initial
NIST SP 800-204C	Yes	N/A	True		True	Initial
NIST SP 800-205	Yes	N/A	True		True	Initial
NIST SP 800-207	Yes	N/A	True		True	Initial
NIST SP 800-208	Yes	N/A	True		True	Initial
NIST SP 800-209	Yes	N/A	True		True	Initial
NIST SP 800-210	Yes	N/A	True		True	Initial
NIST SP 800-213	Yes	N/A	True		True	Initial
NIST SP 800-213A	Yes	N/A	True		True	Initial
NIST SP 800-215	Yes	N/A	True		True	Initial
NIST SP 800-219	Yes	N/A	True		True	Initial
NIST SP 1800-10	Yes	N/A	True		True	Initial
Standard of good Practice for Information Security 2020 (SOGP 2020)	No	Access missing	True		True	Initial
ISO 27799:2016	No	Domain specific: Health systems	True			Initial
ISO/IEC 27019:2017	No	Domain specific: Energy Industry	True			Initial
ISO/IEC 27011:2016	No	Domain specific: Telecommunication	True			Initial
ETSI EN 303 645	No	Domain specific: Consumer IoT security	True			Initial

ISO/SAE 21434	No	Domain specific: Road vehicles – Cybersecurity engineering	True			Initial
ISO 26262	No	Domain specific: Road vehicles — Functional safety		True		Initial
IEC 62304	No	Domain specific: Medical device software	True			Initial
MISRA C/C++	No	Domain specific: C coding standard		True		Initial
PCI PA-DSS v3	No	Domain specific: Payment Application Data Security Standard		True		Initial
NIST SP 800-203	Yes	N/A	True		True	Initial
NIST SP 800-214	Yes	N/A	True		True	Initial
NIST SP 800-220	Yes	N/A	True		True	Initial
NIST SP 800-211	Yes	N/A	True		True	Initial
NIST SP 800-206	Yes	N/A	True		True	Initial
NIST SP 800-195	Yes	N/A	True		True	Initial
NIST SP 800-182	Yes	N/A	True		True	Initial
NIST SP 800-176	Yes	N/A	True		True	Initial
NIST SP 800-170	Yes	N/A	True		True	Initial
NIST SP 800-165	Yes	N/A	True		True	Initial
NIST PRIVACY FRAMEWORK v1	Yes	N/A		True	True	Initial
BS 7799	No	Country specific: United Kingdom			True	Initial
Cloud Security Alliance	Yes	N/A			True	Initial
C2M2	Yes	N/A			True	Initial
IEEE 1402	No	Domain specific: Electric Power			True	Initial
IAEA	No	Domain specific: Nuclear Energy			True	Initial
Wiki 1	No	Company-specific			True	Initial
Wiki 2	No	Company-specific			True	Initial
Whitepaper: Vulnerability Management	No	Company-specific			True	Initial
ISO 30111:2019	Yes	N/A				Cross-Reference
ISO 29147:2018	Yes	N/A				Cross-Reference
Framework for Improving Critical Infrastructure Cybersecurity v1.1	Yes	N/A				Cross-Reference
ISO 31000:2018	Yes	N/A				Cross-Reference

Table A.1: Collected Documents

Appendix B

List of DevOps Principles

Table B.1 presents the list of applicable DevOps principles derived from [94, 68, 80] in Section 2.3.1. It presents the source of the principle in the leftmost column and its description in the right one.

Source	**Principle**
Gene Kim, 1st way	Make work visible
Gene Kim, 1st way	Reduce size of work per deployment cycle
Gene Kim, 1st way ISO 32675, Left-shift	Prevent defects to reach later stages
Gene Kim, 1st way Humble, Automation ISO 32675, Left-shift	Reduce process work through automation
Gene Kim, 1st way Humble, Automation ISO 32675, Left-shift	Continuous build, integration, test, and deployment processes
Gene Kim, 1st way Humble, Automation	Create environments on demand
Gene Kim, 2nd way	Shorten and fasten Feedback Loops
Gene Kim, 2nd way Humble, Automation ISO 32675, Left-shift	Amplify feedback
Gene Kim, 2nd way	Provide feedback to where it is needed
Gene Kim, 3rd way	Support structured experimentation and risk-taking
Gene Kim, 3rd way ISO 32675, Customer Focus Humble, Culture	Work is done with the cumulative and collective experience of everyone in the organization
Humble, Measurement ISO 32675, Mission first	Measure the performance of the project against business metrics
Humble, Culture ISO 32675, Systems thinking	Include cross-domain teams into established meetings

Humble, Sharing ISO 32675, Left-shift	Share project aspects between teams (success, responsibility, tools, infrastructure)
ISO 32675, Systems thinking	Establish a comprehensive, end-to-end view of the system for all stakeholders
ISO 32675, Mission first ISO 32675, Customer Focus	Balance concerns of risk against value for the customer
ISO 32675, Customer Focus	Keep stakeholders informed about changes without overloading them

Table B.1: Aggregated DevOps Principles

Appendix C

Visual Communication Interface

The webinterface and its three central pages described in Section 6.4 are depicted in Figures C.1,C.2,C.3. Figure C.1 presents the overview page for team members like developers. Figure C.2 shows the findings list with sorting and filtering mechanisms. Each of the findings listed can also be accessed as depicted in Figure C.3.

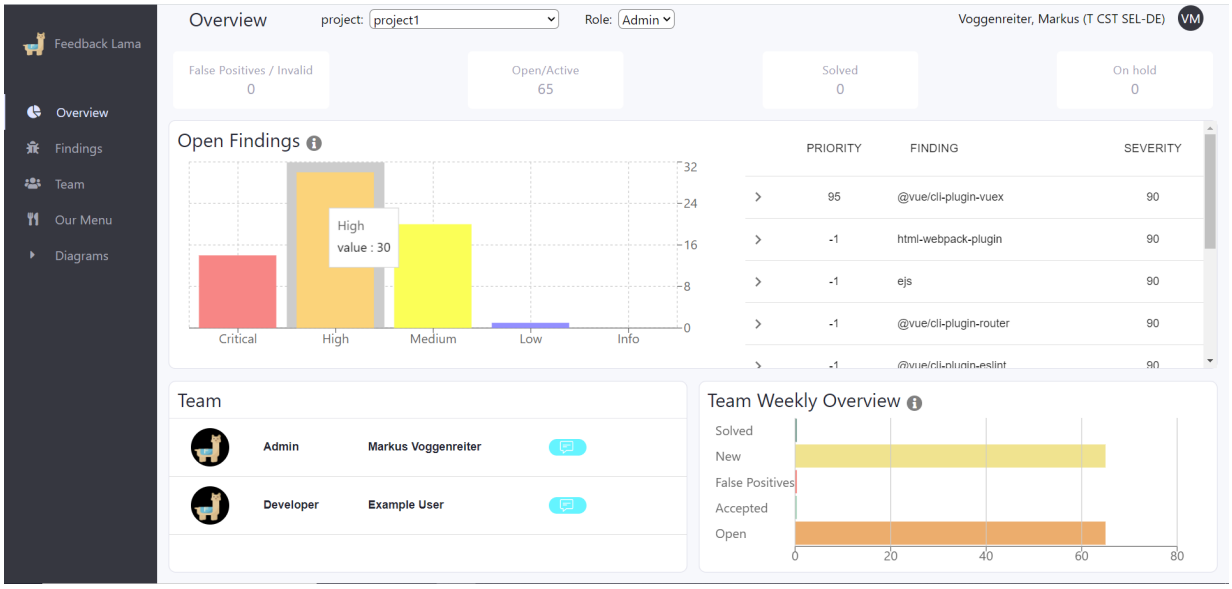

Figure C.1: Webinterface Overview Page

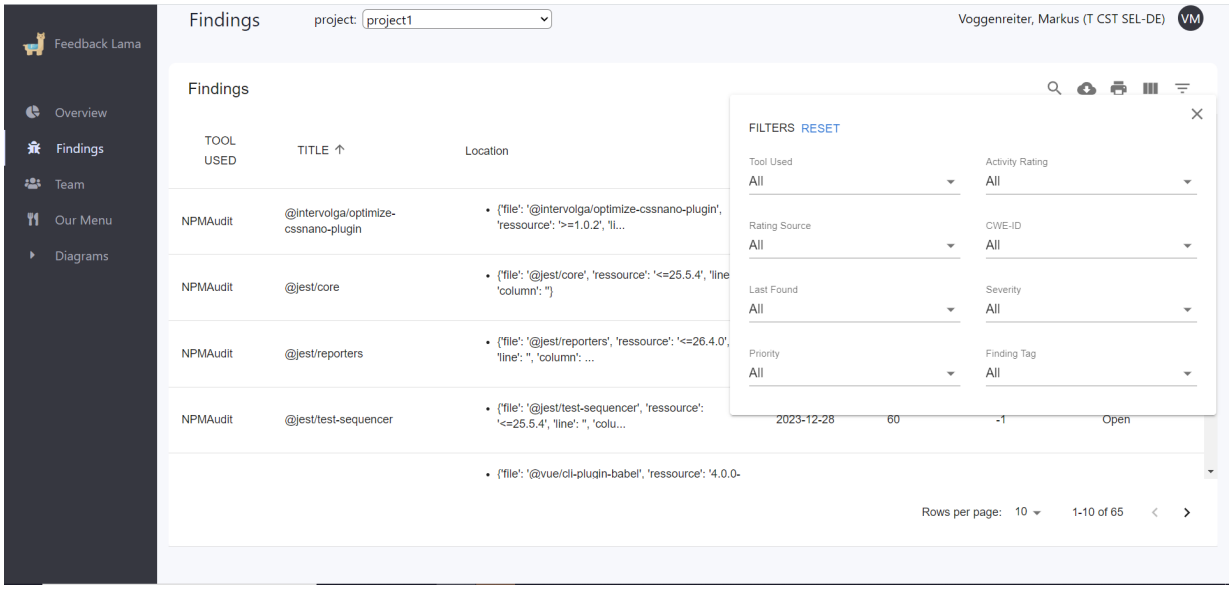

Figure C.2: Webinterface Findings List Page

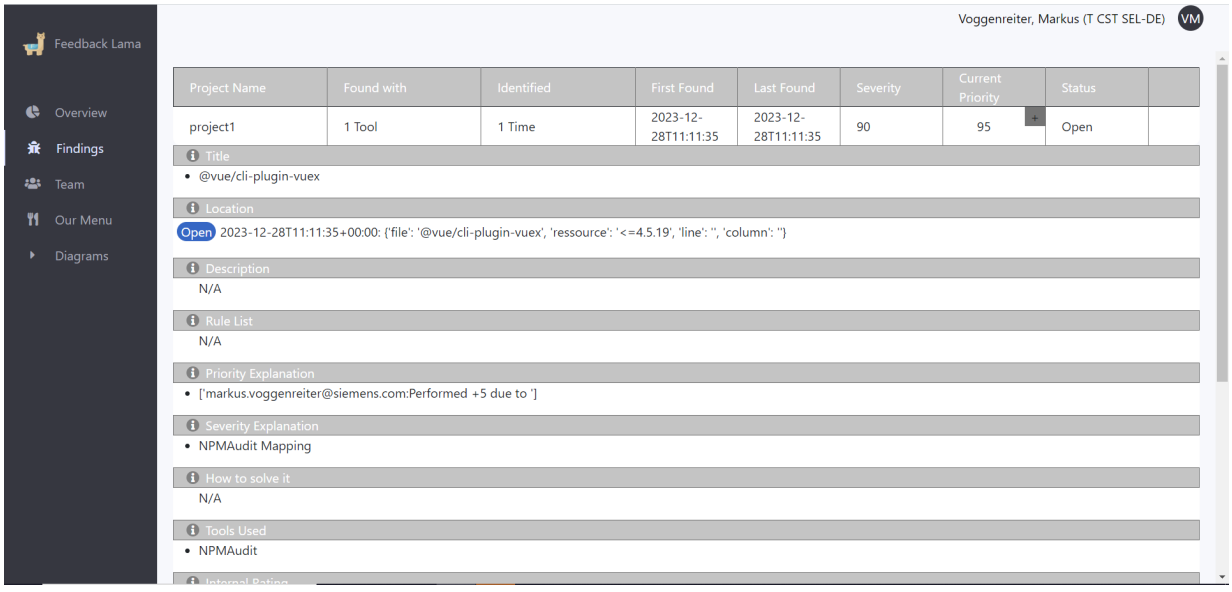

Figure C.3: Webinterface Finding Page

Appendix D

Changelog of the Security Flama during Evaluation

Table D.1 lists all changes in the implementation of the Security Flama that have been performed during the evaluation period. Each change is listed with its date, its type (Bug, Feature), and a description of the change. This data is reported as part of the evaluation results in Section 7.4.

Date	**Change Type**	**Change Description**
09.02.2023	Bug	Fixed bug, preventing users from listing more than ten findings on one page
15.02.2023	Feature	Added parser for new tool
15.02.2023	Feature	Added severity model for new tool
28.02.2023	Feature	Changed color scheme in bar diagram, due to visibility issues on projector
28.02.2023	Bug	Fixed bug, when sorting findings with special characters
10.03.2023	Bug	Fixed bug in network configuration for access from special locations
13.03.2023	Bug	Fixed bug, so that top priority diagram shows only open findings
13.03.2023	Bug	Fixed bug, so that top severity diagram shows only open findings
15.03.2023	Feature	Added HTML information tags to all diagrams and fields, explaining the content
15.03.2023	Feature	Added option to list more than 100 findings
04.04.2023	Feature	Added finding status to the default information presented, when listing findings
06.04.2023	Feature	Added project name to the page, where a specific finding is shown

Table D.1: List of Changes to the Security Flama during the Evaluation Period

Appendix E

Quantitative Evaluation Results

The quantitative data collected during the evaluation in Section 7.4 are reported in Table E.1 and Table E.2. Each row represents one data point collected weekly between January 9th and April 24th. Each column title describes the collected value.

Date	Reports	Findings	Aggr. Findings	New Findings	Open Findings	Input Prio	Sev. Crit.	Sev. High	Sev. Med.	Sev. Low	Sev. Info	Status Open	Status Dis.	Status FP	Status Acc.
01-09	214	1087	354	354	354	0	8	62	101	181	2	1087	0	0	0
01-16	672	1126	381	27	381	0	9	66	112	191	3	1126	0	0	0
01-23	790	1142	393	12	393	0	9	69	117	195	3	1142	0	0	0
01-30	1192	1193	421	28	400	0	9	73	121	194	3	1153	40	0	0
02-06	1595	1226	436	15	403	0	10	75	125	190	3	1151	75	0	0
02-13	1985	1480	472	36	386	0	9	66	110	194	7	1124	356	0	0
02-20	3874	1596	508	36	399	0	9	65	124	199	2	1179	417	0	0
02-27	4360	1612	514	6	391	0	9	65	121	195	1	1145	467	0	0
03-06	4406	1620	514	0	391	0	9	65	121	195	1	1146	474	0	0
03-13	4582	1638	523	9	85	0	2	15	10	58	0	198	1440	0	0
03-20	4582	1638	523	0	85	0	2	15	10	58	0	198	1440	0	0
03-27	4825	1652	534	11	96	0	2	15	11	68	0	210	1440	1	1
04-03	5409	1658	537	3	98	0	2	15	11	68	2	214	1442	1	1
04-10	5508	1670	546	9	98	0	2	16	12	68	0	214	1454	1	1
04-17	5508	1670	546	0	98	0	2	16	12	68	0	214	1454	1	1
04-24	5508	1670	546	0	98	0	2	16	12	68	0	214	1454	1	1

Table E.1: Project A - Quantitative Data

Date	Reports	Findings	Aggr. Findings	New Findings	Open Findings	Input Prio	Sev. Crit.	Sev. High	Sev. Med.	Sev. Low	Sev. Info	Status Open	Status Dis.	Status FP	Status Acc.
01-09	95	1112	408	408	408	0	10	80	121	190	7	1112	0	0	0
01-16	307	1143	409	1	409	0	10	80	121	191	7	1143	0	0	0
01-23	425	1156	412	3	412	0	10	80	124	191	7	1156	0	0	0
01-30	1266	1184	427	15	410	0	11	81	124	191	3	1155	29	0	0
02-06	1692	1217	442	15	413	0	12	83	128	187	3	1153	64	0	0
02-13	1692	1217	442	0	413	0	12	83	128	187	3	1153	64	0	0
02-20	2496	1574	502	60	424	0	12	78	129	203	2	1236	338	0	0
02-27	3316	1614	515	13	397	1	11	73	118	193	2	1142	472	0	0
03-06	3712	1643	518	3	393	0	11	73	114	193	2	1128	515	0	0
03-13	3856	1767	527	9	379	0	10	73	83	211	2	1085	682	0	0
03-20	4882	2034	588	61	371	0	10	72	76	211	2	1068	966	0	0
03-27	5150	2210	615	27	363	1	10	70	73	208	2	1075	1135	0	0
04-03	5309	2210	615	0	363	1	10	70	73	208	2	1075	1135	0	0
04-10	6082	2290	651	36	376	1	9	77	82	208	0	1100	1190	0	0
04-17	6082	2290	651	0	376	1	9	77	82	208	0	1100	1190	0	0
04-24	6082	2290	651	0	376	1	9	77	82	208	0	1100	1190	0	0

Table E.2: Project B - Quantitative Data

Bibliography

[1] Aggarwal, K., Timbers, F., Rutgers, T., Hindle, A., Stroulia, E., and Greiner, R. Detecting duplicate bug reports with software engineering domain knowledge. *Journal of Software: Evolution and Process 29*, 3 (2017).

[2] Ahmed, H. A., Bawany, N. Z., and Shamsi, J. A. Capbug-a framework for automatic bug categorization and prioritization using nlp and machine learning algorithms. *IEEE Access 9* (2021), 50496–50512.

[3] Alahmadi, B. A., Axon, L., and Martinovic, I. 99% false positives: A qualitative study of SOC analysts' perspectives on security alarms. In *31st USENIX Security Symposium (USENIX Security 22)* (Boston, MA, 2022), USENIX Association, pp. 2783–2800.

[4] Almeida, F., Simões, J., and Lopes, S. Exploring the benefits of combining devops and agile. *Future Internet 14*, 2 (2022), 10.3390/fi14020063.

[5] Alqahtani, S. S., Eghan, E. E., and Rilling, J. Sv-af — a security vulnerability analysis framework. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)* (2016), pp. 219–229.

[6] Alqahtani, S. S., Eghan, E. E., and Rilling, J. Tracing known security vulnerabilities in software repositories – a semantic web enabled modeling approach. *Science of Computer Programming 121* (2016), 153–175.

[7] Alshamrani, A., and Bahattab, A. A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model. *International Journal of Computer Science Issues (IJCSI) 12*, 1 (2015), 106–111.

[8] Althebeiti, H., and Mohaisen, D. Enriching vulnerability reports through automated and augmented description summarization. In *Information Security Applications - 24th International Conference, WISA 2023, Jeju Island, South Korea, August 23-25, 2023* (2023), H. Kim and J. M. Youn, Eds., vol. 14402 of *Lecture Notes in Computer Science*, Springer, pp. 213–227.

[9] Amankwah, R., Chen, J., Kudjo, P., Agyemang, B., and Amponsah, A. An automated framework for evaluating open-source web scanner vulnerability severity. *Service Oriented Computing and Applications 14* (2020).

[10] American National Standards Institute. List of Standards Developing Organizations. https://www.standardsportal.org/usa_en/resources/sdo.aspx, Last Accessed: 28.11.2022.

[11] Angermeir, F., Voggenreiter, M., Moyón, F., and Mendez, D. Enterprise-driven open source software: A case study on security automation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (2021), pp. 278–287.

[12] Ani, U. D., He, H., and Tiwari, A. Vulnerability-based impact criticality estimation for industrial control systems. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)* (2020), pp. 1–8.

[13] Anjum, M., Kapur, P., Agarwal, V., and Khatri, S. K. Assessment of software vulnerabilities using best-worst method and two-way analysis. *International Journal of Mathematical, Engineering and Management Sciences 5*, 2 (2020), 328–342.

[14] Aqua Security Software Ltd. Vulnerability database: Aqua security. https://avd.aquasec.com/, Last Accessed: 24.09.2023.

[15] Austin, A., and Williams, L. One technique is not enough: A comparison of vulnerability discovery techniques. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement* (USA, 2011), ESEM '11, IEEE Computer Society, p. 97–106.

[16] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., and et al. Principles behind the agile manifesto. https://agilemanifesto.org/principles.html, Last Accessed: 20.01.2024.

[17] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., and et al. Manifesto for agile software development, 2001. https://agilemanifesto.org/, Last Accessed: 20.01.2024.

[18] Begel, A., and Nagappan, N. Usage and perceptions of agile software development in an industrial context: An exploratory study. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)* (2007), pp. 255–264.

[19] Bellomarini, L., Sallinger, E., and Gottlob, G. The vadalog system: Datalog-based reasoning for knowledge graphs. *Proceedings of the VLDB Endowment 11*, 9 (2018), 975–987.

[20] Bhattacharjee, K., Islam, A., Vaidya, J., and Dasgupta, A. Privee: A visual analytic workflow for proactive privacy risk inspection of open data. In *2022 IEEE Symposium on Visualization for Cyber Security (VizSec)* (2022), pp. 1–11.

[21] Bonifati, A., Dumbrava, S., and Arias, E. J. G. Certified graph view maintenance with regular datalog. *Theory and Practice of Logic Programming 18* (2018), 372 – 389.

[22] Booch, G. The history of software engineering. *IEEE Software 35*, 5 (2018), 108–114.

[23] Brath, R., Peters, M., and Senior, R. Visualization for communication: the importance of aesthetic sizzle. In *Ninth International Conference on Information Visualisation (IV'05)* (2005), pp. 724–729.

[24] BSI. IT-Grundschutz Compendium 2022. `https://www.bsi.bund.de/dok/it-grundschutz-en`, Last Accessed: 26.02.2023.

[25] BSI. BSI-Standard 200-3: Risk Analysis based on IT-Grundschutz, 2017. `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Grundschutz/International/bsi-standard-2003_en_pdf.html`, Last Accessed: 16.07.2023.

[26] BSIMM. Building Security in Maturity Model. `https://www.bsimm.com/`, Last Accessed: 19.05.2023.

[27] Bukhsh, F. A., Bukhsh, Z. A., and Daneva, M. A systematic literature review on requirement prioritization techniques and their empirical evaluation. *Computer Standards & Interfaces 69* (2020), 103389.

[28] Bygstad, B., Ghinea, G., and Brevik, E. Software development methods and usability: Perspectives from a survey in the software industry in Norway. *Interacting with Computers 20*, 3 (2008), 375–385.

[29] Cadariu, M., Bouwers, E., Visser, J., and van Deursen, A. Tracking known security vulnerabilities in proprietary software systems. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (2015), pp. 516–519.

[30] Calì, A., Gottlob, G., and Lukasiewicz, T. Datalog±: A unified approach to ontologies and integrity constraints. In *Proceedings of the 12th International Conference on Database Theory* (New York, NY, USA, 2009), ICDT '09, Association for Computing Machinery, p. 14–30.

[31] Center for Internet Security. The solarwinds cyber-attack: What you need to know, 2021. `https://www.cisecurity.org/solarwinds`, Last Accessed: 03.01.2024.

[32] Chaffey, E. J., and Sgandurra, D. Malware vs anti-malware battle - gotta evade 'em all! In *2020 IEEE Symposium on Visualization for Cyber Security (VizSec)* (2020), pp. 40–44.

[33] Choi, H. J., Lee, H., and Choi, J.-Y. Is a false positive really false positive? In *2022 24th International Conference on Advanced Communication Technology (ICACT)* (2022), pp. 145–149.

[34] Christensen, H. B. Teaching devops and cloud computing using a cognitive apprenticeship and story-telling approach. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2016), ITiCSE '16, Association for Computing Machinery, p. 174–179. https://doi.org/10.1145/2899415.2899426.

[35] Conboy, K., and Carroll, N. Implementing large-scale agile frameworks: Challenges and recommendations. *IEEE Software 36*, 2 (2019), 44–50.

[36] de França, B. B. N., and Ali, N. B. The Role of Simulation-Based Studies in Software Engineering Research. In *Contemporary Empirical Methods in Software Engineering*, M. Felderer and G. H. Travassos, Eds. Springer International Publishing, Cham, 2020, pp. 263–287.

[37] DefectDojo, Inc. Defectdojo/Django-Defectdojo Repository. `https://github.com/DefectDojo/django-DefectDojo`, Last Accessed: 06.09.2023.

[38] Demner-Fushman, D., and Lin, J. Answer extraction, semantic clustering, and extractive summarization for clinical question answering. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics* (Sydney, Australia, July 2006), Association for Computational Linguistics, pp. 841–848.

[39] Dennig, F. L., Cakmak, E., Plate, H., and Keim, D. A. Vulnex: Exploring open-source software vulnerabilities in large development organizations to understand risk exposure. In *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)* (2021), pp. 79–83.

[40] Dimastrogiovanni, C., and Laranjeiro, N. Towards understanding the value of false positives in static code analysis. In *2016 Seventh Latin-American Symposium on Dependable Computing (LADC)* (2016), pp. 119–122.

[41] Diouf, R., Sarr, E. N., Sall, O., Birregah, B., Bousso, M., and Mbaye, S. N. Web scraping: State-of-the-art and areas of application. In *2019 IEEE International Conference on Big Data (Big Data)* (2019), pp. 6040–6042.

[42] Durão, F. A., Vanderlei, T. A., Almeida, E. S., and de L. Meira, S. R. Applying a semantic layer in a source code search tool. In *Proceedings of the 2008*

ACM Symposium on Applied Computing (New York, NY, USA, 2008), SAC '08, Association for Computing Machinery, p. 1151–1157.

[43] Ebert, D. S. Extending visualization to perceptualization: The importance of perception in effective communication of information. In *The Visualization Handbook* (2005), C. D. Hansen and C. R. J. 0001, Eds., Academic Press / Elsevier, pp. 771–780.

[44] Eiter, T., Gottlob, G., and Mannila, H. Disjunctive datalog. *ACM Transactions on Database Systems 22*, 3 (1997), 364–418.

[45] European Comission. Regulation of the european parliament and of the council on horizontal cybersecurity requirements for products with digital elements and amending regulation (EU) 2019/1020, 2022. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:52022PC0454`, Last Accessed: 14.01.2024.

[46] Eyal Salman, H., Hammad, M., Seriai, A.-D., and Al-Sbou, A. Semantic clustering of functional requirements using agglomerative hierarchical clustering. *Information 9*, 9 (2018).

[47] Faraday. Faraday security. `https://faradaysec.com/`, Last Accessed: 03.05.2023.

[48] Farris, K. A., Shah, A., Cybenko, G., Ganesan, R., and Jajodia, S. Vulcon: A system for vulnerability prioritization, mitigation, and management. *ACM Transactions on Privacy and Security 21*, 4 (2018).

[49] Felderer, M., and Travassos, G. H. The evolution of empirical methods in software engineering. In *Contemporary Empirical Methods in Software Engineering*, M. Felderer and G. H. Travassos, Eds. Springer International Publishing, Cham, 2020, pp. 1–24.

[50] Feldt, R., Zimmermann, T., Bergersen, G. R., Falessi, D., Jedlitschka, A., Juristo, N., Münch, J., Oivo, M., Runeson, P., Shepperd, M., Sjøberg, D. I. K., and Turhan, B. Four commentaries on the use of students and professionals in empirical software engineering experiments. *Empirical Software Engineering 23*, 6 (2018), 3801–3820.

[51] Fitzgerald, B., and Stol, K.-J. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software 123* (2017), 176–189.

[52] Frei, S., May, M., Fiedler, U., and Plattner, B. Large-scale vulnerability analysis. In *Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense* (New York, NY, USA, 2006), LSAD '06, Association for Computing Machinery, p. 131–138.

[53] Fruhwirth, C., and Mannisto, T. Improving cvss-based vulnerability prioritization and response with context information. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement* (2009), pp. 535–544.

[54] Public repositories matching the topic of security-tools. `https://github.com/topics/security-tools`, Last Accessed: 03.01.2024.

[55] Glanz, L., Schmidt, S., Wollny, S., and Hermann, B. A vulnerability's lifetime: enhancing version information in cve databases. In *Proceedings of the 15th International Conference on Knowledge Technologies and Data-Driven Business* (New York, NY, USA, 2015), i-KNOW '15, Association for Computing Machinery.

[56] Goodall, J. R. Introduction to visualization for computer security. In *VizSEC 2007: Proceedings of the Workshop on Visualization for Computer Security*, J. R. Goodall, G. Conti, and K.-L. Ma, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 1–17.

[57] Gove, R. Automatic narrative summarization for visualizing cyber security logs and incident reports. In *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)* (2021), pp. 1–9.

[58] Gowda, S., Prajapati, D. L., Singh, R., and Gadre, S. S. False positive analysis of software vulnerabilities using machine learning. *2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)* (2018), 3–6.

[59] Graziotin, D., Fagerholm, F., Wang, X., and Abrahamsson, P. On the unhappiness of software developers. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering* (New York, NY, USA, 2017), EASE'17, Association for Computing Machinery, p. 324–333.

[60] Greenbone AG. Greenbone OpenvAS. `https://www.openvas.org/`.

[61] Guéhéneuc, Y.-G., and Khomh, F. Empirical software engineering. In *Handbook of Software Engineering*, S. Cha, R. N. Taylor, and K. Kang, Eds. Springer International Publishing, Cham, 2019, pp. 285–320.

[62] Gupta, M. K., Govil, M. C., and Singh, G. An approach to minimize false positive in sqli vulnerabilities detection techniques through data mining. In *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)* (2014), pp. 407–410.

[63] Gökçeoğlu, M., and Sözer, H. Automated defect prioritization based on defects resolved at various project periods. *Journal of Systems and Software 179* (2021), 110993.

[64] Hackinger, J. Datagorri: A tool for automated data collection of tabular web content. *Netnomics 19*, 1–2 (2018), 31–41.

[65] Hasselbring, W., Henning, S., Latte, B., Möbius, A., Richter, T., Schalk, S., and Wojcieszak, M. Industrial devops. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)* (2019), pp. 123–126.

[66] Hernan, S., Lambert, S., Ostwald, T., and Shostack, A. Uncover Security Design Flaws Using the STRIDE Approach, 2006. https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach, Last Accessed: 12.02.2022.

[67] Huang, S., Tang, H., Zhang, M., and Tian, J. Text clustering on national vulnerability database. In *2010 Second International Conference on Computer Engineering and Applications* (2010), vol. 2, pp. 295–299.

[68] Humble, J., and Molesky, J. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal 24* (2011), 6–12.

[69] Ijaz, K. B., Inayat, I., and Allah Bukhsh, F. Non-functional requirements prioritization: A systematic literature review. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2019), pp. 379–386.

[70] International Electrotechnical Commission. *IEC 62443 series*. International Electrotechnical Commission, Geneva, Switzerland.

[71] International Organization for Standardization. *ISO/IEC 27000 family*. International Organization for Standardization, Vernier, Geneva, Switzerland. https://www.iso.org/search.html?q=27000.

[72] International Organization for Standardization. *Systems and software engineering - Systems and software assurance*, ISO/IEC/IEEE 15026 ed. International Organization for Standardization, Vernier, Geneva, Switzerland.

[73] International Organization for Standardization. *Information technology — Security techniques — Refining software vulnerability analysis under ISO/IEC 15408 and ISO/IEC 18045*, ISO/IEC TR 20004:2015 ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2015. https://www.iso.org/obp/ui#iso:std:iso-iec:tr:20004:ed-2:v1:en.

[74] International Organization for Standardization. *Information technology — Security techniques — Information security risk management*, ISO/IEC 27005:2018 ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2018. https://www.iso.org/standard/75281.html.

[75] International Organization for Standardization. *Information technology — Security techniques — Vulnerability disclosure*, ISO/IEC 29147:2018 ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2018. https://www.iso.org/standard/72311.html.

[76] International Organization for Standardization. *Risk management - Guidelines*, ISO/IEC 31000:2018 ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2018. https://www.iso.org/standard/65694.html.

[77] International Organization for Standardization. *Information technology — Security techniques — Vulnerability handling processes*, ISO/IEC 30111:2019 ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2019. https://www.iso.org/standard/69725.html.

[78] International Organization for Standardization. *Information security, cybersecurity and privacy protection — Evaluation criteria for IT security*, ISO/IEC 15408:2022 ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2022.

[79] International Organization for Standardization. *Information security, cybersecurity and privacy protection — Information security controls*, ISO/IEC 27002:2022 ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2022. https://www.iso.org/standard/75652.html.

[80] International Organization for Standardization. *Information technology — DevOps — Building reliable and secure systems including application build, package and deployment*, ISO/IEC/IEEE 32675:2022 ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2022. https://www.iso.org/standard/83670.html.

[81] Irene Christine, D., and Thinyane, M. Comparative analysis of cyber resilience strategy in asia-pacific countries. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)* (2020), pp. 71–78.

[82] ISACA. Cobit 2019: Control objectives for information technologies. https://www.isaca.org/resources/cobit, Last Accessed: 19.07.2023.

[83] ISACA. ISACA Glossary. https://www.isaca.org/-/media/files/isacadp/project/isaca/resources/glossary/glossary.pdf, Last Accessed: 03.07.2023.

[84] Isenberg, P., Elmqvist, N., Scholtz, J., Cernea, D., Ma, K.-L., and Hagen, H. Collaborative visualization: Definition, challenges, and research agenda. *Information Visualization 10*, 4 (2011), 310–326.

[85] Jaeger, D., Azodi, A., Cheng, F., and Meinel, C. Normalizing security events with a hierarchical knowledge base. In *Information Security Theory and Practice* (Cham, 2015), R. N. Akram and S. Jajodia, Eds., Springer International Publishing, pp. 237–248.

[86] Jaeger, D., Sapegin, A., Ussath, M., Cheng, F., and Meinel, C. Parallel and distributed normalization of security events for instant attack analysis. In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)* (2015), pp. 1–8.

[87] Jalali, S., and Wohlin, C. Global software engineering and agile practices: A systematic review. *Journal of Software: Evolution and Process 24* (2012).

[88] Jarzebowicz, A., and Weichbroth, P. A qualitative study on non-functional requirements in agile software development. *IEEE Access 9* (2021), 40458–40475.

[89] Jiang, L., and Eberlein, A. An analysis of the history of classical software development and agile development. In *2009 IEEE International Conference on Systems, Man and Cybernetics* (2009), pp. 3733–3738.

[90] Joh, H., and Malaiya, Y. K. Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics. In *The 2011 International Conference on Security and Management (SAM)* (2011), Citeseer, pp. 10–16.

[91] Kaiya, H., and Saeki, M. Using domain ontology as domain knowledge for requirements elicitation. In *14th IEEE International Requirements Engineering Conference (RE'06)* (2006), pp. 189–198.

[92] Kifer, M. Nonmonotonic reasoning in flora-2. In *Logic Programming and Nonmonotonic Reasoning* (Berlin, Heidelberg, 2005), C. Baral, G. Greco, N. Leone, and G. Terracina, Eds., Springer Berlin Heidelberg, pp. 1–12.

[93] Kim, G. *Phoenix project: A novel about it, devops, and helping your business win*, 5 ed. IT Revolution Press, Portland, OR, 2018.

[94] Kim, G., Debois, P., Willis, J., and Humble, J. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.* IT Revolution Press, 2016.

[95] Krótkiewicz, M., Wojtkiewicz, K., and Jodłowiec, M. Towards semantic knowledge base definition. In *Biomedical Engineering and Neuroscience* (Cham, 2018), W. P. Hunek and S. Paszkiel, Eds., Springer International Publishing, pp. 218–239.

[96] Krótkiewicz, M., Wojtkiewicz, K., Jodłowiec, M., and Pokuta, W. Semantic knowledge base: Quantifiers and multiplicity in extended semantic networks module. In *Knowledge Engineering and Semantic Web* (Cham, 2016), A.-C. Ngonga Ngomo and P. Křemen, Eds., Springer International Publishing, pp. 173–187.

[97] Kuhn, A., Ducasse, S., and Gîrba, T. Semantic clustering: Identifying topics in source code. *Information and Software Technology 49* (2007), 230–243.

[98] Kuhrmann, M., Méndez, D., and Daneva, M. On the pragmatic design of literature studies in software engineering: an experience-based guideline. *Empirical Software Engineering. 22*, 6 (2017), 2852–2891.

[99] Landauer, T. K., and Dumais, S. T. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review 104* (1997), 211–240.

[100] Le, T. H. M., Chen, H., and Babar, M. A. A survey on data-driven software vulnerability assessment and prioritization. *ACM Computing Surveys 55*, 5 (2022).

[101] Lee, D.-G., and Seo, Y.-S. Improving bug report triage performance using artificial intelligence based document generation model. *Human-centric Computing and Information Sciences 10* (2020).

[102] Leffingwell, D. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*, 1st ed. Addison-Wesley Professional, 2011.

[103] Leffingwell, D. Safe 5.0 framework, 2022. `https://www.scaledagileframework.com/`, Last Accessed: 14.05.2023.

[104] Leite, L., Rocha, C., Kon, F., Milojicic, D., and Meirelles, P. A survey of devops concepts and challenges. *ACM Computing Surveys 52*, 6 (2019).

[105] Lomborg, S., and Bechmann, A. Using apis for data collection on social media. *The Information Society 30*, 4 (2014), 256–265.

[106] Maidl, M., Kröselberg, D., Zhao, T., and Limmer, T. System-specific risk rating of software vulnerabilities in industrial automation amp; control systems. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (2021), pp. 327–332.

[107] Majewska, O., McCarthy, D., Vulić, I., and Korhonen, A. Acquiring verb classes through bottom-up semantic verb clustering. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)* (Miyazaki, Japan, May 2018), European Language Resources Association (ELRA).

[108] Marandi, M., Bertia, A., and Silas, S. Implementing and automating security scanning to a devsecops ci/cd pipeline. In *2023 World Conference on Communication and Computing (WCONF)* (2023), pp. 1–6.

[109] Marshall, D., and Viviano, A. Threat modeling for drivers - Windows drivers. `https://docs.microsoft.com/en-us/windows-hardware/drivers/driversecurity/threat-modeling-for-drivers`, Last Accessed: 12.02.2022.

[110] Matalonga, S., Nöel, R., Pedraza-Garcia, G., Astudillo, H., and Fernández, E. *Generating Software Security Knowledge Through Empirical Methods*. CRC Press, 11 2017, pp. 95–137.

[111] Matharu, G. S., Mishra, A., Singh, H., and Upadhyay, P. Empirical study of agile software development methodologies: A comparative analysis. *SIGSOFT Softw. Eng. Notes 40*, 1 (2015), 1–6.

[112] Matkovic, P., and Tumbas, P. A comparative overview of the evolution of software development models. *Journal of Industrial Engineering and Management 1* (2010), 163–172.

[113] Mayr, P., and Weller, K. Think before you collect: Setting up a data collection approach for social media studies. *The SAGE handbook of social media research methods* (2017), 679–712.

[114] Medeiros, I., Neves, N. F., and Correia, M. Automatic detection and correction of web application vulnerabilities using data mining to predict false positives. In *Proceedings of the 23rd International Conference on World Wide Web* (New York, NY, USA, 2014), WWW '14, Association for Computing Machinery, p. 63–74.

[115] Méndez, D., and Passoth, J.-H. Empirical software engineering: From discipline to interdiscipline. *Journal of Systems and Software 148* (2019), 170–179.

[116] Miller, G. A. Wordnet: A lexical database for english. *Communications of the ACM 38*, 11 (1995), 39–41.

[117] Mishra, A., and Otaiwi, Z. Devops and software quality: A systematic mapping. *Computer Science Review 38* (2020), 100308.

[118] Mitchell, S. M., and Seaman, C. B. A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement* (2009), pp. 511–515.

[119] Morali, A., and Wieringa, R. Risk-based confidentiality requirements specification for outsourced it systems. In *2010 18th IEEE International Requirements Engineering Conference* (2010), pp. 199–208.

[120] Motik, B., Nenov, Y., Piro, R., and Horrocks, I. Maintenance of datalog materialisations revisited. *Artificial Intelligence 269* (2019), 76–136.

[121] Moyón, F., Soares, R., Pinto-Albuquerque, M., Mendez, D., and Beckers, K. Integration of security standards in devops pipelines: An industry case study. In *Product-Focused Software Process Improvement* (Cham, 2020), M. Morisio, M. Torchiano, and A. Jedlitschka, Eds., Springer International Publishing, pp. 434–452.

[122] Nadeem, M., Williams, B. J., and Allen, E. B. High false positive detection of security vulnerabilities: a case study. In *Proceedings of the 50th Annual Southeast*

Regional Conference (New York, NY, USA, 2012), ACM-SE '12, Association for Computing Machinery, p. 359–360.

[123] Nambi, S. N. A. U., Sarkar, C., Prasad, R. V., and Rahim, A. A unified semantic knowledge base for iot. In *2014 IEEE World Forum on Internet of Things (WF-IoT)* (2014), pp. 575–580.

[124] National Institute of Standards and Technology. Common Platform Enumeration. `https://nvd.nist.gov/products/cpe`, Last Accessed: 07.09.2023.

[125] National Institute of Standards and Technology. National Vulnerability Database. `https://nvd.nist.gov/`, Last Accessed: 20.01.2024.

[126] National Institute of Standards and Technology. *NIST Special Publication (SP) 800 series*. National Institute of Standards and Technology, Gaithersburg, Maryland, United States. `https://csrc.nist.gov/publications/sp800`.

[127] National Institute of Standards and Technology. Security Requirement - Glossary. `https://csrc.nist.gov/glossary/term/security_requirement`, Last Accessed: 12.12.2023.

[128] National Institute of Standards and Technology. *Cybersecurity Framework Version 1.1*. National Institute of Standards and Technology, Gaithersburg, Maryland, United States, 2018. `https://www.nist.gov/cyberframework/framework`.

[129] National Institute of Standards and Technology. *Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy*. National Institute of Standards and Technology, Gaithersburg, Maryland, United States, 2018. `https://csrc.nist.gov/publications/detail/sp/800-37/rev-2/final`.

[130] National Institute of Standards and Technology. *Integrating Cybersecurity and Enterprise Risk Management (ERM)*. National Institute of Standards and Technology, Gaithersburg, Maryland, United States, 2020. `https://csrc.nist.gov/pubs/ir/8286/final`.

[131] National Institute of Standards and Technology. *Security and Privacy Controls for Information Systems and Organizations*. National Institute of Standards and Technology, Gaithersburg, Maryland, United States, 2020. `https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final`.

[132] National Institute of Standards and Technology. *Workforce Framework for Cybersecurity (NICE Framework)*. National Institute of Standards and Technology, Gaithersburg, Maryland, United States, 2020. `https://csrc.nist.gov/publications/detail/sp/800-181/rev-1/final`.

[133] National Institute of Standards and Technology. *Engineering Trustworthy Secure Systems*. National Institute of Standards and Technology, Gaithersburg, Maryland, United States, 2022. https://csrc.nist.gov/publications/detail/sp/800-160/vol-1-rev-1/final.

[134] National Institute of Standards and Technology. *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. National Institute of Standards and Technology, Gaithersburg, Maryland, United States, 2022. https://csrc.nist.gov/publications/detail/sp/800-218/final.

[135] NVD. NVD Vulnerability Severity Ratings. https://nvd.nist.gov/vuln-metrics/cvss, Last Accessed: 20.01.2024.

[136] Nyanchama, M. Enterprise vulnerability management and its role in information security management. *Information Systems Security 14*, 3 (2005), 29–56.

[137] OpenSSL Foundation, Inc. Openssl security policy. https://www.openssl.org/policies/general/security-policy.html, Last Accessed: 20.01.2024.

[138] Othmane, L. b., Jaatun, M. G., and Weippl, E. *Empirical research for software security*. Taylor & Francis Inc, 2017.

[139] OWASP. OWASP Defectdojo. https://owasp.org/www-project-defectdojo/, Last Accessed: 03.06.2022.

[140] OWASP. OWASP SAMM. https://owaspsamm.org/, Last Accessed: 19.12.2023.

[141] OWASP. OWASP Top Ten Proactive Controls 2018 | C1: Define Security Requirements. https://owasp.org/www-project-proactive-controls/v3/en/c1-security-requirements, Last Accessed: 23.09.2021.

[142] Pandey, N., Hudait, A., Sanyal, D. K., and Sen, A. Automated classification of issue reports from a software issue tracker. In *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications* (Singapore, 2018), P. K. Sa, M. N. Sahoo, M. Murugappan, Y. Wu, and B. Majhi, Eds., Springer Singapore, pp. 423–430.

[143] Peng, J., Zhang, M., and Wang, Q. Deduplication and exploitability determination of uaf vulnerability samples by fast clustering. *KSII Transactions on Internet and Information Systems 10* (2016), 4933–4956.

[144] Petersen, K., and Wohlin, C. The effect of moving from a plan-driven to an incremental software development approach with agile practices. *Empirical Software Engineering 15*, 6 (2010), 654–693.

[145] Plant, O. H., van Hillegersberg, J., and Aldea, A. How devops capabilities leverage firm competitive advantage: A systematic review of empirical evidence. In *2021 IEEE 23rd Conference on Business Informatics (CBI)* (2021), vol. 01, pp. 141–150.

[146] Rajapakse, R. N., Zahedi, M., and Babar, M. A. An empirical analysis of practitioners' perspectives on security tool integration into devops. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (New York, NY, USA, 2021), ESEM '21, Association for Computing Machinery.

[147] Ramirez, A., Aiello, A., and Lincke, S. J. A survey and comparison of secure software development standards. In *2020 13th CMI Conference on Cybersecurity and Privacy (CMI) - Digital Transformation - Potentials and Challenges(51275)* (2020), pp. 1–6.

[148] Rangnau, T., Buijtenen, R. v., Fransen, F., and Turkmen, F. Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)* (2020), pp. 145–154.

[149] Red Hat, Inc. Red hat cve database. `https://access.redhat.com/security/security-updates/#/cve`, Last Accessed: 14.12.2023.

[150] Reimers, N., and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing* (2019), Association for Computational Linguistics.

[151] Rindell, K., Bernsmed, K., and Jaatun, M. G. Managing security in software: Or: How i learned to stop worrying and manage the security technical debt. In *Proceedings of the 14th International Conference on Availability, Reliability and Security* (New York, NY, USA, 2019), ARES '19, Association for Computing Machinery.

[152] Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., and Männistö, T. Devops adoption benefits and challenges in practice: A case study. In *Product-Focused Software Process Improvement* (Cham, 2016), P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, and T. Mikkonen, Eds., Springer International Publishing, pp. 590–597.

[153] Rodrigues, I. M., Aloise, D., Fernandes, E. R., and Dagenais, M. A soft alignment model for bug deduplication. In *Proceedings of the 17th International Conference on Mining Software Repositories*. Association for Computing Machinery, New York, NY, USA, 2020, p. 43–53.

[154] Runeson, P., Engström, E., and Storey, M.-A. The design science paradigm as a frame for empirical software engineering. In *Contemporary Empirical Methods in*

Software Engineering, M. Felderer and G. H. Travassos, Eds. Springer International Publishing, Cham, 2020, pp. 127–147.

[155] Runeson, P., and Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering 14*, 2 (2009), 131–164.

[156] Ryzhyk, L., and Budiu, M. Differential datalog. *Datalog 2* (2019), 4–5.

[157] Sacher, D. Fingerpointing false positives: How to better integrate continuous improvement into security monitoring. *Digital Threats 1*, 1 (2020).

[158] SAFECode. Fundamental practices for secure software development, 2018. `https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf`, Last Accessed: 04.02.2023.

[159] Samonas, S., and Coss, D. The cia strikes back: Redefining confidentiality, integrity and availability in security. *Journal of Information System Security 10*, 3 (2014).

[160] Schneider, P. App ecosystem out of balance: An empirical analysis of update interdependence between operating system and application software. Master's thesis, TUM, 2020.

[161] Schneider, P., Voggenreiter, M., Gulraiz, A., and Matthes, F. Semantic similarity-based clustering of findings from security testing tools. In *Proceedings of the 5th International Conference on Natural Language and Speech Processing (ICNLSP 2022)* (Trento, Italy, Dec. 2022), Association for Computational Linguistics, pp. 134–143.

[162] Schreiber, A., Sonnekalb, T., and Kurnatowski, L. v. Towards visual analytics dashboards for provenance-driven static application security testing. In *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)* (2021), pp. 42–46.

[163] Schäfer, M., Fuchs, M., Strohmeier, M., Engel, M., Liechti, M., and Lenders, V. Blackwidow: Monitoring the dark web for cyber security information. In *2019 11th International Conference on Cyber Conflict (CyCon)* (2019), vol. 900, pp. 1–21.

[164] Seaman, C. B. Qualitative methods. In *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer London, London, 2008, pp. 35–62.

[165] Sharma, R., Sibal, R., and Sabharwal, S. Software vulnerability prioritization using vulnerability description. *International Journal of System Assurance Engineering and Management 12* (2020).

[166] Shu, K., Mahudeswaran, D., and Liu, H. Fakenewstracker: a tool for fake news collection, detection, and visualization. *Computational and Mathematical Organization Theory 25* (2019).

[167] Simpson, S. SAFECode whitepaper: Fundamental practices for secure software development 2nd edition. In *ISSE 2014 Securing Electronic Business Processes* (2014), H. Reimer, N. Pohlmann, and W. Schneider, Eds., Springer Fachmedien Wiesbaden, pp. 1–32.

[168] Software in the Public Interest, Inc. Security bug tracker. `https://security-tracker.debian.org/tracker/`, Last Accessed: 06.12.2023.

[169] Sołtysik-Piorunkiewicz, A., and Krysiak, M. The cyber threats analysis for web applications security in industry 4.0. In *Towards Industry 4.0 — Current Challenges in Information Systems*, M. Hernes, A. Rot, and D. Jelonek, Eds. Springer International Publishing, Cham, 2020, pp. 127–141.

[170] SonarSource S.A. Sonarqube. `https://www.sonarsource.com/products/sonarqube/`.

[171] Stieglitz, S., Mirbabaie, M., Ross, B., and Neuberger, C. Social media analytics – challenges in topic discovery, data collection, and data preparation. *International Journal of Information Management 39* (2018), 156–168.

[172] Stol, K.-J., and Fitzgerald, B. Guidelines for Conducting Software Engineering Research. In *Contemporary Empirical Methods in Software Engineering*, M. Felderer and G. H. Travassos, Eds. Springer International Publishing, Cham, 2020, pp. 27–62.

[173] Svensson, R. B., Gorschek, T., Regnell, B., Torkar, R., Shahrokni, A., Feldt, R., and Aurum, A. Prioritization of quality requirements: State of practice in eleven companies. In *2011 IEEE 19th International Requirements Engineering Conference* (2011), pp. 69–78.

[174] Sweetman, R., and Conboy, K. Portfolios of agile projects: A complex adaptive systems' agent perspective. *Project Management Journal 49* (2018), 875697281880271.

[175] Taherdoost, H. Understanding cybersecurity frameworks and information security standards—a review and comprehensive overview. *Electronics 11* (2022), 2181.

[176] Tan, X., Peng, X., Pan, S., and Zhao, W. Assessing software quality by program clustering and defect prediction. In *2011 18th Working Conference on Reverse Engineering* (2011), pp. 244–248.

[177] Thakur, K., Qiu, M., Gai, K., and Ali, M. L. An investigation on cyber security threats and security models. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (2015), pp. 307–311.

[178] The MITRE Corporation. Common Vulnerabilities and Exposures. `https://www.cve.org/`, Last Accessed: 29.12.2023.

[179] The MITRE Corporation. Common weakness enumeration. `https://cwe.mitre.org/index.html`.

[180] Thomas, R. J., Gardiner, J., Chothia, T., Samanis, E., Perrett, J., and Rashid, A. Catch me if you can: An in-depth study of cve discovery time and inconsistencies for managing risks in critical infrastructures. In *Proceedings of the 2020 Joint Workshop on CPS & IoT Security and Privacy* (New York, NY, USA, 2020), CPSIOTSEC'20, Association for Computing Machinery, p. 49–60.

[181] Tian, Y., Lo, D., Xia, X., and Sun, C. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering 20* (2014).

[182] Tofan, D. Information security standards. *Journal of Mobile, Embedded and Distributed Systems 3* (2011).

[183] Torkura, K. A., Sukmana, M. I., and Meinel, C. Integrating continuous security assessments in microservices and cloud native applications. In *Proceedings of The10th International Conference on Utility and Cloud Computing* (New York, NY, USA, 2017), UCC '17, Association for Computing Machinery, p. 171–180.

[184] Travassos, G. H., and Felderer, M. *Contemporary empirical methods in software engineering.* Springer, 2020.

[185] Tvaronavičienė, M., Plėta, T., Casa, S. D., and Latvys, J. Cyber security management of critical energy infrastructure in national cybersecurity strategies: cases of USA, UK, France, Estonia and Lithuania. *Insights into Regional Development 2*, 4 (2020), 802 – 813.

[186] Ullah, K. W., Ahmed, A. S., and Ylitalo, J. Towards building an automated security compliance tool for the cloud. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (2013), pp. 1587–1593.

[187] Velásquez, J. D., and Palade, V. A knowledge base for the maintenance of knowledge extracted from web data. *Knowledge-Based Systems 20*, 3 (2007), 238–248.

[188] Viertel, F. P., Brunotte, W., Strüber, D., and Schneider, K. Detecting security vulnerabilities using clone detection and community knowledge. In *The 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, Hotel Tivoli, Lisbon, Portugal, July 10-12, 2019* (2019), A. Perkusich, Ed., KSI Research Inc. and Knowledge Systems Institute Graduate School, pp. 245–324.

[189] VizSec: Visualization for Cyber Security. `https://vizsec.org/`.

[190] Vmware. Vmware/Differential-Datalog: DDlog . `https://github.com/vmware/differential-datalog`, Last Accessed: 14.06.2023.

[191] Voggenreiter, M. Online Material of the Preliminary Evaluation of the Analysis and Tracking Approach, 2022. `https://doi.org/10.6084/m9.figshare.22005656.v1`.

[192] Voggenreiter, M. Supplementary Material for the Empirical Evaluation, 2023. `https://figshare.com/articles/figure/Supplementary_Material/24235147`.

[193] Voggenreiter, M., Angermeir, F., Moyón, F., Schöpp, U., and Bonvin, P. Automated security findings management: A case study in industrial devops. *2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (2024). to be published.

[194] Voggenreiter, M., Schneider, P., and Gulraiz, A. Aggregating industrial security findings with semantic similarity-based techniques. In *Practical Solutions for Diverse Real-World NLP Applications*, M. Abbas, Ed. Springer International Publishing, Cham, 2024, pp. 121–139.

[195] Voggenreiter, M., and Schöpp, U. Using a semantic knowledge base to improve the management of security reports in industrial devops projects. *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (2022), 309–310.

[196] Voggenreiter, M., and Schöpp, U. Prioritizing industrial security findings in agile software development projects. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)* (2023), pp. 375–379.

[197] Volz, R., Staab, S., and Motik, B. Incremental maintenance of materialized ontologies. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE* (Berlin, Heidelberg, 2003), R. Meersman, Z. Tari, and D. C. Schmidt, Eds., Springer Berlin Heidelberg, pp. 707–724.

[198] Wagner, S., Méndez, D., Felderer, M., Graziotin, D., and Kalinowski, M. Challenges in survey research. In *Contemporary Empirical Methods in Software Engineering* (2020).

[199] Wang, J. A., and Guo, M. Ovm: An ontology for vulnerability management. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies* (New York, NY, USA, 2009), CSIIRW '09, Association for Computing Machinery.

[200] Wang, J. A., and Guo, M. Security data mining in an ontology for vulnerability management. In *2009 International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing* (2009), pp. 597–603.

[201] Wang, S., Pan, J. Z., Zhao, Y., Li, W., Han, S., and Han, D. Belief base revision for datalog+/- ontologies. In *Semantic Technology* (Cham, 2014), W. Kim, Y. Ding, and H.-G. Kim, Eds., Springer International Publishing, pp. 175–186.

[202] Wiedemann, A., Wiesche, M., and Krcmar, H. Integrating development and operations in cross-functional teams - toward a devops competency model. In *Proceedings of the 2019 on Computers and People Research Conference* (New York, NY, USA, 2019), SIGMIS-CPR '19, Association for Computing Machinery, p. 14–19.

[203] Wirth, N. A brief history of software engineering. *IEEE Annals of the History of Computing 30*, 3 (2008), 32–39.

[204] Wohlin, C. Empirical software engineering research with industry: Top 10 challenges. In *2013 1st International Workshop on Conducting Empirical Studies in Industry (CESI)* (2013), pp. 43–46.

[205] Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. *Experimentation in Software Engineering*. Springer Publishing Company, 2012.

[206] Yazdani, A., and Thakur, M. S. OWASP DevSecOps Guideline, 2023. `https://github.com/OWASP/DevSecOpsGuideline`, Last Accessed: 29.06.2023.

[207] Yun, B., Croitoru, M., Vesic, S., and Bisquert, P. Naked: N-ary graphs from knowledge bases expressed in datalog±. In *AAMAS: Autonomous Agents and MultiAgent Systems* (2019), pp. 2390–2392.

[208] Zaman, S., Adams, B., and Hassan, A. E. Security versus performance bugs: A case study on firefox. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (New York, NY, USA, 2011), MSR '11, Association for Computing Machinery, p. 93–102.

[209] Zhou, Y., Tong, Y., Gu, R., and Gall, H. Combining text mining and data mining for bug report classification. In *2014 IEEE International Conference on Software Maintenance and Evolution* (2014), pp. 311–320.