Designing a Security-First Scrum Framework with AI Enhancements

University of Turku
Department of Computing
Master of Science (Tech) Thesis

June 2026
Arslaan Asad

UNIVERSITY OF TURKU
Department of Computing

Arslaan Asad: Designing a Security-First Scrum Framework with AI Enhancements

Master of Science (Tech) Thesis, 137 p., 6 app. p.

June 2026

This thesis addresses the challenge of integrating software security into Scrum-based Agile development, where security activities are often neglected or treated as separate from the development process. Despite extensive research on securing Scrum, many existing approaches remain impractical for real-world adoption due to their complexity or reliance on specialised expertise.

To address this gap, this study proposes an LLM-Supported Secure Scrum (LSS) framework that embeds security practices directly into Scrum artefacts, thereby ensuring continuous visibility and prioritisation of security early in the development lifecycle. The framework design is grounded in a structured literature review with thematic synthesis, which identifies security practices suitable for Agile environments. In addition, the framework incorporates Large Language Models (LLMs) as an enabling mechanism to operationalise and support these practices within development workflows.

Specifically, in this thesis, LLMs are used to assist key security activities, including risk identification, S-Tag generation, misuse and abuser stories, security acceptance criteria (SAC), and an adaptive security Definition of Done (DoD). A prototype tool is developed to demonstrate the feasibility of integrating these capabilities into a practical development setting.

The proposed framework is evaluated through a mixed-method approach combining a controlled demonstration with expert evaluation and a survey. The results suggest that the framework may improve security visibility, reduce the knowledge gap between developers and security practices, and support integration with existing Agile workflows without substantially disrupting perceived development velocity.

Overall, this research contributes an approach for achieving AI-assisted security integration in Scrum, helping to bridge the gap between Secure Scrum research and its application in real-world development environments.

Keywords: Security in Scrum, Secure Agile Development, Software Security, Large Language Models (LLMs), Security-First Framework, Agile Security Integration, Security User Stories, S-Tags, AI-Assisted Software Engineering

Contents

List of Figures

List of Tables

List of acronyms

AC Acceptance Criteria

AI Artificial Intelligence

ASVS Application Security Verification Standard

Auto-CoT Automatic CoT

CI/CD Continuous Integration and Continuous Deployment

CIA Confidentiality, Integrity, and Availability

CoT Chain-of-Thought

CWE Common Weakness Enumeration

DAST Dynamic Application Security Testing

DoD Definition of Done

LGPD Brazilian General Data Protection Law

LLM Large Language Model

LogiCoT Logical CoT

LSS LLM-Supported Secure Scrum

OWASP Open Worldwide Application Security Project

PBI Product Backlog Item

PO Product Owner

RAT Retrieval-Augmented Thoughts

S-SDLC Secure Software Development Life Cycle

SACS Smart Aging Care Systems

SAST Static Application Security Testing

SDLC Software Development Life Cycle

SLR Systematic Literature Review

SME Small and Medium-sized Enterprise

SR Security Risk

ST Security Task

ToT Tree-of-Thoughts

XSS Cross-Site Scripting

1 Introduction

In the past few decades, one of the most popular software development approaches has been Agile [1], [2]. The approach is based on the Agile principles which are highlighted in the Agile Manifesto [3]. These principles aim to support faster delivery of the project while also prioritising communication and flexibility during development. This was revolutionary compared to previous development methods such as Waterfall, which were rigid in process and often lacked flexibility during the development cycle [4].

Scrum is one of the most widely used frameworks within Agile methodologies. It is used across many development teams. According to the 17th Annual State of Agile Report, approximately 71% of organisations include Agile practices in their software development lifecycle, with Scrum used by over 60% of Agile teams [1]. Ensuring the security of software developed using Scrum has become a prominent concern due to its widespread adoption. In the past decade, much of the research has focused on making the Scrum development process more secure to combat the rising cyber threats [5]. Recent studies in Secure Scrum identify a range of practices and potential additions to the Agile framework that can overall make the entire development process more secure.

In industrial practice, a unified consensus among researchers and practitioners has not yet been reached, but it can be observed that these specific Secure Scrum practices have been shown to potentially improve security assurance when applied

correctly. However, many organisations still struggle to integrate security into their regular Scrum process because the current guidelines are mostly fragmented, inconsistent, or difficult to adopt in fast-paced environments [6], [7]. This creates a visible gap between what the research proposes and what development teams can realistically apply during sprints [8].

Due to these challenges, designing a structured and practical approach that can bring security into the early stages of Scrum, while also minimising negative impact on agility has become an important research topic. At the same time, with recent advancements in Artificial Intelligence (AI), specifically Large Language Models (LLMs), new opportunities have emerged for automating or assisting complex tasks inside the software development lifecycle (e.g. requirements engineering, testing, debugging, documentation) [9]. These technologies can support teams by generating security artefacts (e.g. misuse stories, security AC), identifying potential risks, or improving communication between developers and security experts.

Therefore, this thesis focuses on designing an LLM-Supported Secure Scrum (LSS) framework that can integrate security practices into Scrum events and artefacts, while simultaneously exploring the role of LLMs in enhancing these practices. The goal is to create a practical and adaptable framework that is grounded in established prior research and suitable for real-world Scrum teams.

1.1 Research Motivation

In recent years, cybercrime has increased significantly, hence making security an essential aspect of modern digital systems [10]. In many organisations, security is treated as a stand-alone activity rather than being integrated into the Agile development process. Additionally, the fast-paced nature of Agile development may lead to security being neglected or deprioritised compared to functional requirements [11].

A significant amount of research exists on secure Agile and Secure Scrum; how-

ever, many of the proposed solutions are either too theoretical or too difficult to adopt in real development environments. Furthermore, some solutions require specialised experts, for example security analysts or threat hunters, which many small and medium-sized software companies cannot afford. Development teams are also under constant pressure to deliver product features quickly. This often leads to security-related tasks being postponed or completely ignored because security work does not usually provide immediate visual or functional improvements to the customer [7], [12]. This situation has created a clear gap between academic research and the practical integration of security in the development process, leaving security tasks continuously untracked and undervalued. However, recent advancements in LLMs present new opportunities to bridge this gap by automating or simplifying security-related activities.

Consequently, there is a strong motivation to design a practical framework that selects suitable security practices and integrates them directly into the Scrum process, while also exploring how LLMs can be used to enhance this integration. A structured, secure approach may help development teams reduce security risks at earlier stages without significantly slowing down their development pace.

1.2 Research Goal

There are two main goals of this thesis. The first goal is to design an LSS framework that integrates essential security activities into each Scrum event and artefact. This is achieved by conducting a Literature Review to identify security practices that are flexible enough to keep pace with the fast development cycles of Scrum and can therefore be applied in practice. The objective of this thesis is to create an approach that builds upon existing Secure Scrum research by choosing the best practices.

The second goal of this thesis is to explore how LLMs can enhance and automate these selected security practices. This includes examining how LLMs can support

backlog refinement, risk identification, documentation, and the validation of security requirements. By combining the pre-selected Secure Scrum practices with LLM-based assistance, this thesis aims to propose a framework that is practical enough to keep up with the flexibility of Scrum. At the same time, the framework is designed to be effective in tracking security vulnerabilities within the project, preserving the agility in Scrum while increasing the visibility, consistency, and automation of security-related work.

To address these objectives, this thesis is guided by two research questions.

- RQ1: How can security practices be effectively integrated into Scrum workflows?
- RQ2: How can Large Language Models (LLMs) support and enhance the integration of security practices in Scrum-based Agile development?

The first research question focuses on the integration of security practices into Scrum and is addressed through a literature review. The second research question extends this by exploring how LLMs can support and enhance these practices through framework design, implementation, and evaluation.

1.3 Thesis Outline

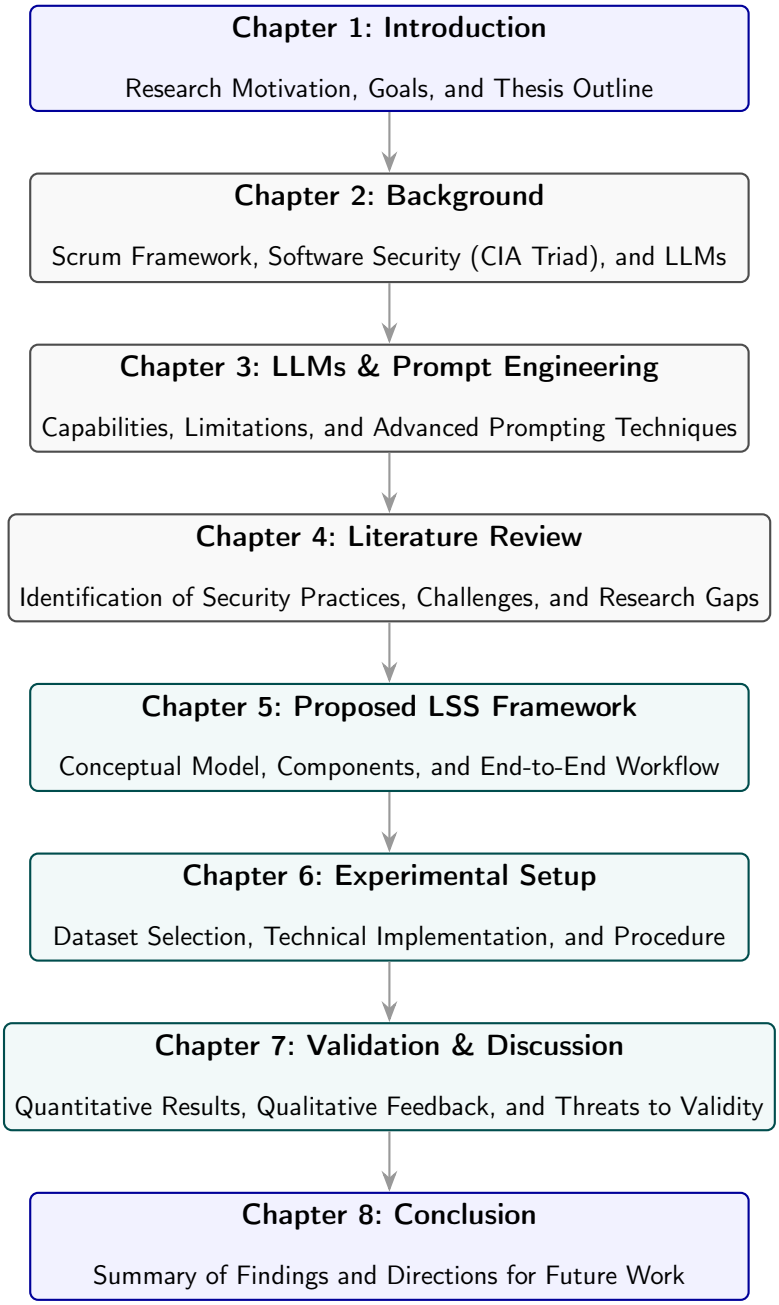

Figure 1.1: Methodological flow and structural organisation of the thesis

This thesis is structured as follows. Chapter 1 introduces the research, explains the motivation, and defines the main goals of the study. Chapter 2 provides the necessary background on software security, Scrum, and the various challenges of integrating

security into Agile environments, along with an overview of LLMs in this context. Chapter 3 presents the fundamentals of LLMs and prompt engineering, thereby explaining their capabilities, limitations, and how they can support software engineering activities. Chapter 4 presents the Literature Review that was performed to examine existing secure Scrum practices and identify the major challenges, mechanisms, and research gaps in integrating security into Agile environments. Chapter 5 introduces the proposed LSS framework, explaining the design of the framework, its core elements, and how it differs from existing secure Scrum models. Chapter 6 describes the experimental setup, including dataset selection, technical implementation, and evaluation design. Chapter 7 presents the validation of the proposed approach, including both quantitative results and qualitative feedback, followed by a discussion of the findings and threats to validity. Finally, Chapter 9 concludes the thesis with a summary of the findings and potential future research directions.

1.4 AI Declaration

In this thesis, Artificial Intelligence (AI) serves a dual purpose. Primarily, it is the core technology driving the proposed LSS framework, acting as the main subject of research and experimentation. Secondarily, AI was used in a limited capacity during the writing process to assist in improving the academic tone, sentence clarity, and grammar checking of the text. However, the core ideas, research design, framework development, implementation, and analysis presented in this thesis are the original work of the author.

2 Background

This chapter provides the foundational concepts necessary for understanding the proposed framework. It introduces the core principles and mechanics of Scrum, outlines the fundamental concepts of software security, and provides an overview of Large Language Models (LLMs) and their application in software engineering.

2.1 History and Overview of Scrum Framework

Scrum is a widely used Agile framework that supports iterative and incremental software development through a structured set of roles, events, and artefacts [1], [13]. It is based on empirical process control, which focuses on transparency, inspection, and adaptation as key principles for managing complex software projects [13]. Instead of relying on extensive upfront planning, Scrum allows teams to deliver work in small increments while continuously incorporating feedback throughout the development lifecycle.

Scrum organises development work into fixed-length iterations called sprints, which last one month or less [13]. Each sprint produces a usable Increment, enabling teams to respond to changing requirements and stakeholder needs [1], [3]. Work is managed through the Product Backlog, an ordered list of items needed to improve the product [13]. During sprint planning, a subset of items are selected for implementation, helping to ensure ongoing alignment with business value.

Scrum Roles: Scrum defines three primary roles: the Product Owner (PO), the Scrum Master, and the Developers [13]. The PO is responsible for managing and ordering the Product Backlog. The Scrum Master supports the team by facilitating Scrum practices and removing impediments. The Developers are responsible for delivering a usable Increment each sprint. These roles are intentionally minimal to support flexibility and accountability within the team.

Scrum Events: Scrum includes a set of time-boxed events that support inspection and adaptation [13]. Sprint Planning defines the work for the upcoming sprint. The Daily Scrum enables Developers to coordinate their activities and identify obstacles. At the end of the sprint, a Sprint Review is held to inspect the Increment and gather feedback. This is followed by a Sprint Retrospective, where the team reflects on its processes and identifies improvements.

Scrum Artefacts: Scrum defines three key artefacts: the Product Backlog, the Sprint Backlog, and the Increment [13]. The Product Backlog contains all the work items needed for the product. The Sprint Backlog represents the selected work for the current sprint, along with a plan for delivering it. The Increment is the sum of completed work and must meet the Definition of Done (DoD). These artefacts provide transparency and help track progress.

Strengths and Limitations of Scrum: Scrum has been widely reported to improve responsiveness to change and support the frequent delivery of working software [1]. While Scrum has been widely reported as effective for improving productivity and adaptiveness against changing functional requirements, Scrum does not explicitly prescribe security practices.

Security requirements are often described as non-functional and are typically implicit and difficult to specify, test, and implement in a short development cycle

[7], [12]. As a result, security considerations are frequently postponed or addressed reactively. Alternatively, they may be handled by separate teams or processes that are independent of the Scrum development cycle.

The framework does not explicitly define practices for addressing non-functional requirements such as security [14]. As discussed in prior research, this can create challenges in systematically integrating security activities into Scrum processes [7], [15]. This can create tension because Scrum focuses on speed and flexibility, while effective security engineering relies on a structured, systematic development process.

This limitation highlights a broader challenge of integrating security into Scrum-based development. As a result, additional approaches are required to incorporate security considerations into Scrum-based development. This challenge is further examined in the literature review (Chapter 4).

2.2 Software Security

Software security is the discipline of designing, building, and testing software to withstand malicious attacks by integrating security practices throughout the software development lifecycle [16]. The primary goal of software security is for the software to behave as intended even in unexpected conditions.

The Confidentiality, Integrity, and Availability (CIA) triad is one of the most widely used foundations for evaluating software security. Together, these three principles represent the core security objectives that software systems are expected to achieve [17]. So, before integrating robust security measures into flexible development processes like Scrum, it becomes necessary to understand these core principles.

Confidentiality: Confidentiality ensures that information is protected from unauthorised access and disclosure and is only available to authorised users and purposes

[17]. Its primary objective is to restrict access to information so that it is not exposed to entities that lack the adequate permission to use it.

Integrity: Integrity ensures that information remains accurate and protected from unauthorised modification or destruction [17]. Furthermore, data must not be altered improperly and must remain consistent throughout its lifecycle.

Availability: Availability ensures that information is accessible and usable in a timely and reliable manner when required [17]. The emphasis here is on maintaining system uptime and access so that authorised users can interact with the software whenever necessary.

Together, the CIA triad provides a fundamental basis for information security. Security practices and controls are traditionally designed to protect confidentiality, integrity, and availability [17]. If any weakness in confidentiality, integrity, or availability is observed, it can lead to system compromise and eventually loss of trust in the software.

2.2.1 Security in Traditional vs Agile Development

Traditional software development methods addressed security through early security analysis, architectural design, and explicit security requirements before implementation begins [14], [18]. These approaches rely on more structured planning and documentation that support security concerns during the early stages of development. However, we identified that these practices do not directly translate to Agile development methods. Scrum by design reduces upfront planning and documentation overhead in favour of incremental software delivery [14], [19]. This may reduce opportunities for systematic threat analysis, security requirement elicitation, and architectural risk assessment during the early stages of development. As a result,

security concerns may be addressed later in the development process or less completely, which increases the chances of vulnerabilities and rework costs [14], [18].

2.2.2 Security in Scrum

Many software vulnerabilities can originate from missing security requirements, insufficient early security analysis, or weaknesses in design decisions rather than purely implementation errors [7], [15].

This can be particularly problematic in Scrum environments, where Scrum highlights working features and sprint progress. In such settings, security requirements and other non-functional requirements may receive less attention, especially during early requirement elicitation and specification [19], [20].

To address these limitations, different extensions to Scrum such as Secure Scrum [14], ScrumS [21], and Security-Backlog-Based approaches [22], [23] have been proposed. These approaches introduce concepts such as security backlogs, misuse or abuse stories, security roles, and additional security activities within the development process [23], [24].

While these approaches show that security and agility are not fundamentally incompatible, embedding security practices into Scrum remains challenging in practice. This is particularly due to limited security expertise and the difficulty of incorporating security activities within sprints [7], [8].

2.3 Evolution of Large Language Models

Given these ongoing challenges in integrating security without compromising agility, there is growing interest in exploring new forms of automated support mechanisms. At the same time, recent advances in AI, particularly LLMs, have created new opportunities for supporting software engineering tasks through automated analysis,

generation, and recommendation. These LLMs are deep learning algorithms that are trained on massive datasets to understand and generate human language [25]. They introduce novel opportunities to support security integration within Agile processes. LLMs have been increasingly applied across a range of software engineering activities, including requirements engineering, documentation generation, testing, and code-related tasks [9]. More about LLMs is explained in Section 3.1.

Existing research further indicates that LLMs have shown potential for language-intensive activities such as requirements elicitation and analysis, where generative approaches may support early development phases [26].

When an LLM is applied with appropriate safeguards, it may assist development teams in identifying potential risks and generating security-related artefacts, while also improving traceability across development activities. However, their outputs still require careful validation, as current research highlights limitations in both reliability and correctness. In particular, LLMs may generate outputs that seem plausible but are factually incorrect or not faithful to the input, a phenomenon commonly referred to as hallucination [9], [27].

Overall, these characteristics suggest that LLMs can support security-oriented practices within Scrum, for example by helping generate security artefacts. However, their outputs still require systematic validation to ensure trustworthiness in practice.

LLM-Supported Secure Scrum: Recent developments and prior research on security in agile software development indicate a need for more systematic approaches to integrating security into Scrum processes. Prior studies highlight ongoing challenges in aligning security practices with iterative development, especially due to mismatches between traditional security activities and agile workflows [7], [8], [28].

These findings suggest that security integration should be more visible, traceable, and actionable within sprints, while also remaining practical for teams operating in real-world development environments. Rather than introducing heavyweight pro-

cesses, existing literature emphasises the importance of approaches that preserve agility while improving the consistency and effectiveness of security practices [8], [18].

In this context, LLMs can be used as supportive tools with appropriate checks and balances, rather than as replacements for human judgement [9], [26]. This suggests their potential to help bridge the gap between security requirements and development practices.

Building on the identified challenges of security integration in Scrum and the potential of LLMs, this thesis proposes a structured and adaptable approach for integrating security considerations into Scrum, with particular attention to confidentiality, integrity, and availability as core security concerns [17].

3 Large Language Models (LLMs) and Prompt Engineering

This chapter discusses the fundamentals of LLMs and prompt engineering, which serve as the main driving forces behind the framework proposed in this thesis.

3.1 Large Language Models

In recent years, LLMs have demonstrated capabilities in supporting various reasoning-related tasks and a wide range of software development activities. Their underlying architecture is based on the transformer model, which processes input sequences using attention mechanisms to capture contextual relationships between tokens. Text is first decomposed into smaller units, which are commonly referred to as tokens that may represent words, subwords, or symbols. These are then mapped into numerical representations for model processing [25], [29]. Through self-attention, the model assigns varying levels of importance to tokens in a sequence, allowing it to capture dependencies across both local and long-range contexts.

However, the detailed architectural complexity and internal mechanisms of LLMs are not within the scope of this study. Instead, the focus is on their practical capabilities, particularly their ability to process contextual information, reason over diverse software artefacts (e.g. requirements, user stories, code snippets), and generate structured outputs that align with software development practices. Furthermore,

their ability to interpret instructions expressed in natural language makes them suitable for integration as supportive tools within Scrum teams [29].

Modern LLMs are instruction-tuned, meaning that they undergo supervised fine-tuning on instruction and response datasets. This process trains them to interpret and follow instructions in natural language rather than modelling text in a purely unsupervised manner [25]. As a result, instruction-tuned LLMs can be guided by concise prompts to produce responses with a predictable structure and format. This characteristic is well suited to agile development, where requirements, constraints, and priorities change frequently. Furthermore, as discussed in existing surveys, this shift from pure language modelling to structured instruction-following has been reported to improve the reliability of LLMs in certain operational settings [25].

The contextual ability of the LLM is further extended through integration with external retrieval systems (e.g. search APIs). When integrated with such tools, LLMs can be guided to incorporate the latest information into their outputs [25]. In our case, this will be beneficial, as we can obtain the latest security-related concerns relevant to our project domain and ask the LLM to generate output to better protect the project from these concerns.

3.1.1 Limitations of LLMs

It is important to note that while exhibiting strong performance in reasoning tasks, LLMs remain probabilistic systems with known limitations. Even though their operational capabilities are enormous, they can still produce information that looks correct but is actually false, a phenomenon known as hallucination [27], [30]. Furthermore, they can produce inconsistent outputs, or become sensitive to small changes in instruction phrasing. This is concerning when we consider the security-sensitive development environment we are trying to build with our framework. Evaluations across different benchmarks highlight that LLMs face challenges related to factual

reliability, robustness against adversarial prompts, limited creativity, bias amplification, and evaluation inconsistencies [31], [32], [33]. For our framework, this means that outputs generated will have to be verified. This is especially true when it concerns security requirements, risk identification, or threat modelling.

3.1.2 Integrating the Framework with Different LLMs

The integration of LLMs in the framework should remain flexible. This is because every development team may require the model to perform differently according to their needs. For example, if an LLM is overly prioritising a certain matter, then this behaviour can be tuned by giving and refining the high-level instructions at the end of the sprint (sprint retrospective). As noted in the literature, model behaviour can often be effectively adjusted through prompt-based instruction refinements, although such changes may remain sensitive to wording variations [29]. Furthermore, models are constantly upgraded to improve their capabilities, and older models become obsolete. This is why our new proposed framework must be loosely coupled with any LLM, so that any LLM model can easily be plugged in without changing the core working mechanics of the framework.

3.2 Prompt Engineering

Given these capabilities and limitations, effective interaction with LLMs becomes critical. Prompt engineering provides a straightforward and structured process for guiding model reasoning to ensure outputs remain consistent with the task context. When incorporating LLMs into development workflows, effective prompts elevate from a usability feature to a necessity. This aligns with recent research describing prompt engineering as a key mechanism for enabling pre-trained LLMs to adapt to

downstream tasks without modifying model parameters, effectively bridging general capabilities and application-specific requirements [34].

3.2.1 Prompt Design and Instruction Clarity

Academic literature describes prompt engineering as the use of task-specific prompts that guide pre-trained LLMs toward downstream tasks without retraining or modifying the core model parameters [34]. Consequently, a well-crafted prompt enables the same model to perform various tasks, such as reasoning or generation, simply by altering its structure and content. This approach provides notable accuracy improvements across different domains [34].

A core element of prompt engineering is instruction clarity. Vague prompts produce overly general outputs, whereas prompts with precise structure and content, including specific phrasing and level of detail, produce responses that better align with the intended task [35]. This is critical in security-related environments, such as our proposed framework. If a prompt does not clearly specify the domain, threat environment, or underlying assumptions, the model may rely on general-world knowledge rather than domain-specific security requirements. This behaviour can undermine the reliability of the secure framework and the overall development process.

3.2.2 Advanced Prompting Techniques

This subsection explains some of the prompting techniques currently used to effectively leverage the capabilities of LLMs.

Chain-of-Thought (CoT): Beyond basic prompting, several advanced techniques have emerged to enhance the contextual and operational capabilities of LLMs. Chain-of-Thought (CoT) prompting is one such technique, where the model is guided to produce intermediate reasoning steps that connect the input to the final output.

This allows complex problems to be broken down into sequential steps, hence improving performance on arithmetic, common-sense, and symbolic reasoning tasks [36].

Few-shot and Zero-shot: CoT can be applied in both few-shot and zero-shot settings. In few-shot prompting, a small number of input and output examples with intermediate reasoning steps are provided to guide the model. On the other hand, zero-shot prompting relies only on task instructions without examples. Notably, simple instructions such as "Let's think step by step" have been shown to elicit reasoning behaviour in zero-shot settings, enabling models to generate intermediate reasoning chains without manually provided examples [36], [37].

Automatic Chain-of-Thought (Auto-CoT) and Logical Chain-of-Thought (LogiCoT): To minimise the reliance on manually crafted demonstrations, Automatic Chain-of-Thought (Auto-CoT) generates diverse reasoning chains automatically, thereby improving the robustness and scalability of CoT prompting [37]. Furthermore, Logical Chain-of-Thought (LogiCoT) introduces verification mechanisms that validate and revise intermediate reasoning steps, thereby reducing logical errors and inconsistencies through a think–verify–revise process [34].

Retrieval-Augmented Thoughts (RAT) and Tree-of-Thoughts (ToT): More recently, Retrieval-Augmented Thoughts (RAT) extends CoT by incorporating external knowledge retrieval into the reasoning process. This enables iterative refinement of intermediate reasoning steps and improves factual grounding in complex tasks [38]. In contrast, structured frameworks such as Tree-of-Thoughts (ToT) allow for the exploration of multiple reasoning paths through search-based strategies. This enables models to evaluate, backtrack, and select more promising solutions [39].

Self-Consistency Prompting: Another technique is self-consistency prompting. This technique is based on the idea that complex problems can have multiple valid reasoning paths. Instead of relying on a single reasoning path, the model generates multiple reasoning paths and combines them to produce a final answer based on consistency across outputs. Prior work has shown that this approach improves performance in arithmetic, common-sense, and symbolic reasoning tasks [35]. In the context of security-related software engineering tasks, this consistency-oriented prompting approach may help reduce contradictory or unsupported outputs. This is particularly true in activities such as threat modelling or risk analysis, where factual consistency is critical [27].

3.2.3 Prompt Sensitivity and Optimisation

The field of prompt engineering also recognises the issue of prompt sensitivity. It is the phenomenon where small changes in a prompt can lead to drastically different results. Ye et al. therefore emphasise that iterative work is needed to refine the prompt to achieve accurate results [40]. Their work introduces PE2, which is essentially a prompting framework where an LLM is instructed to evaluate the inadequacies of a given prompt and then propose targeted improvements. The authors also managed to demonstrate that this approach outperforms existing prompt optimisation baselines, and is reported to improve prompt stability relative to evaluated baselines [40]. This suggests that prompts can be automated to some extent, which may reduce the reliance on manual trial and error and support more adaptive refinement of prompts during and after sprints based on developer and Scrum Master reviews. However, automated prompt improvement is considered future work and is not included in this thesis or the proposed framework.

3.2.4 Contextualisation in Secure Software Development

Current prompt engineering research highlights the importance of including contextual information in prompts to guide LLM behaviour and improve the relevance of outputs. Both Sahoo et al. [34] and Chen et al. [35] emphasise that prompt structure, specificity, and contextual detail significantly influence model performance and response quality. This is relevant for our LSS framework. In our case, it is vital to specifically incorporate the context of the problem, namely, the project domain and its security concerns, so that the LLM becomes more capable of generating or incorporating threat-related information (potentially via retrieval augmentation), identifying risks, or generating domain-appropriate mitigation strategies, while also reducing the likelihood of hallucination or irrelevant suggestions.

3.2.5 Security Risks of Prompt Engineering

Finally, recent studies also highlight security risks associated with prompt engineering. For example, Chen et al. [35] discuss adversarial attacks that exploit vulnerabilities in prompt design. Their findings suggest that prompt engineering introduces several inherent weaknesses, including vulnerability to adversarial manipulation, strong sensitivity to prompt structure, and limited control over model outputs. Since LLM behaviour is strongly influenced by the content and formulation of prompts, poorly designed or malicious inputs can often lead to unreliable responses. These characteristics indicate that, while prompt engineering is powerful, it can introduce an additional attack surface if not carefully managed. In this case, it introduces two governance requirements within the framework. First, guardrails (e.g. output validation rules, prompt constraints, or human-in-the-loop review mechanisms) must be put in place to ensure that model outputs remain within safe and intended boundaries. Second, we must have checks and balances (e.g. via the Scrum Master, a security reviewer, or an automated validation pipeline) when we incorporate feedback at the

end of the sprint to slightly modify the behaviour of the model. If the intent is malicious, then instead of providing benefits, the entire framework can become a liability.

4 Literature Review

This chapter presents the Literature Review conducted to address the research objectives of this thesis. The Literature Review focuses on the integration of security practices into Scrum (RQ1) and is guided by the following sub-questions:

- *RQ1: How can cybersecurity best practices be systematically integrated into Scrum events and artefacts while preserving agility?*
- *RQ2: What additional roles, checkpoints, or process adaptations are necessary to achieve continuous security assurance in Scrum-based development?*

The method used in this section is inspired by the guidelines for Systematic Literature Reviews (SLRs) proposed by Kitchenham and Charters [41], but it does not fully follow their prescribed protocol. Instead, this study adopts a structured literature review approach with thematic synthesis.

While the main phases of an SLR, which include planning, conducting, and reporting, are followed conceptually, several methodological adaptations were made to suit the scope and constraints of this Master's thesis [41]. This approach is consistent with prior software engineering studies that adapt systematic review guidelines to support exploratory and design-oriented research.

First, the selection and data extraction processes were carried out by a single researcher, whereas Kitchenham and Charters recommend multiple independent reviewers to minimise bias and enable inter-rater agreement [41]. Second, although

explicit inclusion and exclusion criteria were applied, there was no formal quantitative quality assessment scoring that was used to weight the included studies. As a result, the findings should be interpreted as indicative patterns rather than statistically weighted evidence. Third, the search was limited to two major databases (Scopus and IEEE Xplore), rather than a broader multi-source search strategy.

Finally, due to the relatively large number of included studies (n = 42) and the broad scope of the research questions, data extraction focused on identifying clearly defined and explicitly described security practices that met the inclusion criteria, rather than exhaustively cataloguing all possible practices within each study. This approach aligns with thematic synthesis methods commonly used in qualitative literature reviews, where the objective is to identify recurring patterns and develop descriptive and analytical themes across studies [42], [43]. To mitigate potential selection bias, practices were included only when supported by clear textual evidence.

Therefore, this work is positioned as a structured literature review with thematic synthesis, informed by systematic review principles rather than a fully protocol-driven SLR. The structure of this chapter, however, broadly follows the three main phases of an SLR as summarised by Kitchenham and Charters: planning, conducting, and reporting the review [41].

4.1 Methodological Setup for the Literature Review

In this subsection, we will describe the planning phase of the literature review. This includes the description of the preliminary work done before conducting the review. This section also explains the search query that was selected for each research question, as well as the justification for using specific search terms.

4.1.1 Query Selection

The search terms were selected to directly support the research questions. Since the objectives of RQ1 (embedding cybersecurity practices into Scrum events and artefacts) and RQ2 (roles, checkpoints, and process adaptations for continuous assurance) are closely related, they have been combined into a single research query from which articles and journals will be searched in the databases. The rationale is that literature addressing embedding practices into events/artefacts will often simultaneously discuss new roles or checkpoints. Therefore, papers grouped under the categories *Agile Security Integration*, *Secure Scrum*, and *additional roles and checkpoints in Scrum* will be analysed together to capture this overlap and provide a more holistic understanding of how cybersecurity can be systematically embedded into Scrum without compromising agility. The search query prioritised precision, which may have excluded some relevant studies that use alternative terms such as "secure SDLC", but overall it aims to produce a more focused set of results. Table 4.1 shows the four categories of search terms used to target and collect papers.

Query Label	`Search String`
RQ1 and RQ2 - Scrum Security Integration and Modification	`("Agile" OR "Scrum") AND ("cybersecurity" OR "software security" OR "secure software development" OR "DevSecOps") AND ("Secure Scrum" OR "Agile security" OR "security backlog" OR "S-Tag" OR "security user stories")`

Table 4.1: Search query for research questions 1 and 2

4.1.2 Databases Used

In this thesis, two primary databases were used for the collection of relevant papers. They were IEEE[1] and Scopus[2].

[1] `https://ieeexplore.ieee.org/Xplore/home.jsp`

[2] `https://www.scopus.com/pages/home`

4.1.3 Additional Boundaries and Filters

To keep the results relevant, further boundaries and filters are also proposed, which include the following:

1. Time period: papers from after the year 2000 are selected
2. Language: English
3. Document types: peer-reviewed journals and conference papers
4. Proceedings: no preprints, only peer-reviewed
5. Domains: software engineering, computer science, cybersecurity, Agile, Scrum (further expanded in Table 4.2)

Building upon these initial boundaries, further inclusion and exclusion criteria were established to systematically evaluate the literature within the targeted domains.

Inclusion Criteria	**Exclusion Criteria**
• Preprints: not included in this literature review • Papers dealing with Agile and Scrum • Papers dealing with case studies related to modified Agile or Scrum to enhance security • Journals and Conferences are included • Any paper offering practical problems or solutions for integrating Security and Agile methodologies	• Papers unrelated to security or software development • Duplicates or incomplete records • Papers focusing solely on other Agile methodologies (e.g. Kanban, XP) • Papers focusing on other software development models (e.g. Waterfall) • Papers focusing only on testing or validation of Agile security

Table 4.2: Further inclusion and exclusion criteria based on domains

4.1.4 Search Results

The search query was used in both the IEEE and Scopus databases. The total number of articles that were fetched from IEEE was 13, excluding a reference to a book. On the other hand, Scopus database on the same query showed 196 results. This was further refined to 90 results by searching Scrum within the fetched 196 results.

4.1.5 Screening the Papers

After getting the results from the search query, the screening process of the literature review started.

According to Kitchenham and Charters [41], at the beginning of the selection process, the criteria should be applied broadly. This means that unless a study can be clearly excluded based on its title or abstract, the full text should be retrieved for review. Accordingly, titles, abstracts, and keywords were initially screened, and in cases that were unclear, the full article was examined to determine whether it should be included in the review.

From the total of 13 papers/journals/conferences in IEEE, 6 were chosen based on the selection criteria and closeness to the domain of the research question. On the other hand, out of 90 results in Scopus, a total of 59 papers were selected.

These selected papers from both databases (65 in total) were then combined and further analysed based on the extensive inclusion and exclusion criteria provided in 4.2, and as a result, a total of 42 primary papers were selected for this literature review. It is also to be noted that 7 secondary studies were also identified among the original papers. However, they were not thematically analysed and are listed independently to further validate the findings of this literature review. The entire screening process is also visualised in Figure 4.1.

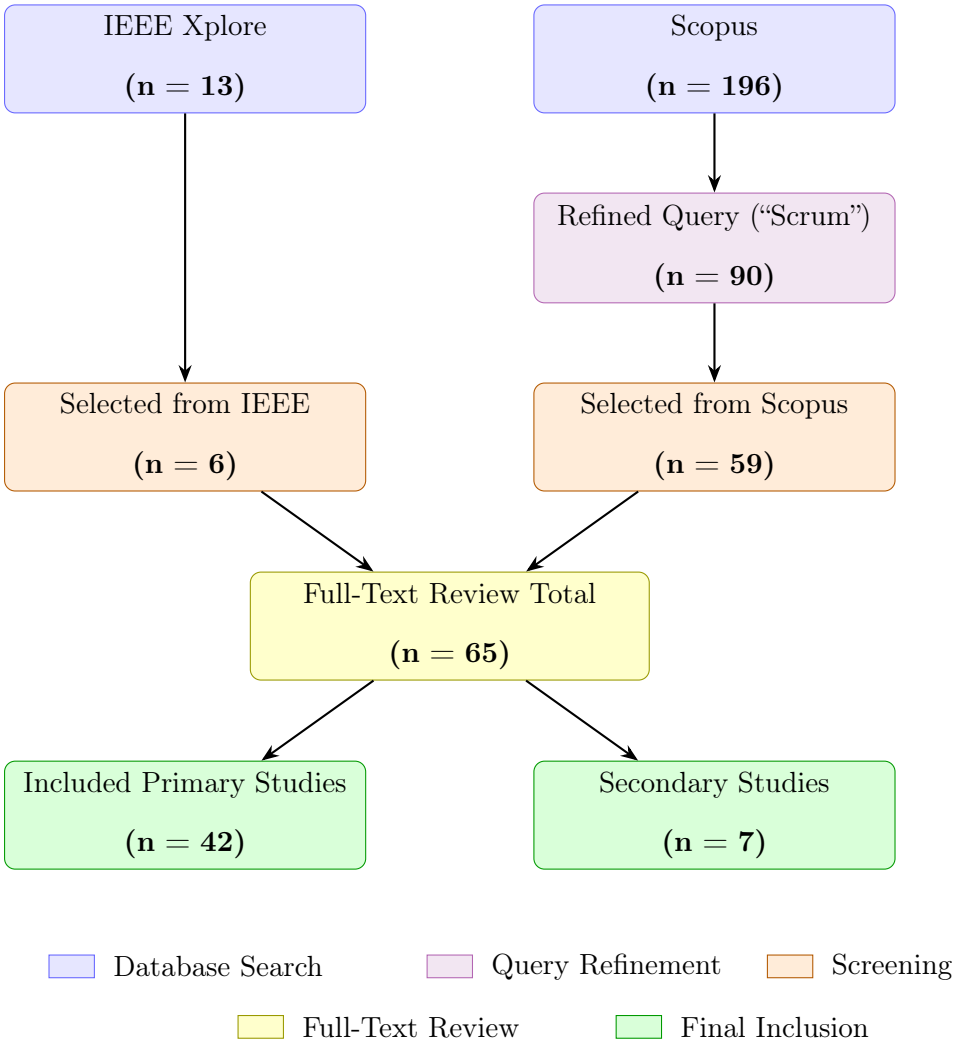

Figure 4.1: Study selection process

4.2 Findings of the Literature Review

This section synthesises the findings of the reviewed studies to address RQ1 and RQ2. While the literature often discusses security challenges in Agile environments more broadly, this thesis focuses specifically on Scrum as the most widely adopted Agile framework. Accordingly, findings derived from general Agile contexts are interpreted in terms of their relevance and applicability to Scrum-based development.

4.2.1 Scrum: Overview & Conceptual Models

The Agile framework, and in particular Scrum, are widely used approaches in software development. However, despite their strengths in flexibility and rapid delivery, early research has pointed out their limitations in addressing modern security concerns. This is because security concerns are often non-functional requirements that are neither visible nor easily demonstrable to stakeholders. One of the main principles of Agile is to deliver tangible software to the client as soon as possible [3]; however, security does not manifest as tangibly as functional requirements do. Furthermore, defining Scrum's agility is not feasible, as each organisation defines it differently. This raises concerns about the extent to which Scrum aligns with the detailed requirements of secure software engineering.

As a result, a growing number of researchers worked to overcome this incompatibility by extending or modifying Scrum with security-focused practices. Esteves et al. introduced ScrumS, which is a conceptual model that modifies the Scrum lifecycle with established security techniques with the introduction of "Sprint Zero," where the project tools, settings, and technologies are prepared before regular development sprints start [21]. This model also introduces explicit Security User Stories and vulnerability analysis through attack trees to ensure that security is incorporated in early development [21].

In contrast, Mohino et al. propose a new Secure Software Development Life Cycle (S-SDLC) which is tailored to Agile settings [44]. Within this model, security requirements, risk analysis, and verification activities such as code analysis and penetration testing are integrated across the requirements, design, implementation, testing, deployment, and maintenance phases. It also reuses Agile concepts like sprints, roles, and prioritised backlogs so that security requirements engineering and security testing are treated as regular parts of development rather than add-ons at the end [44]. S-SDLC provides a structured approach to aligning Agile development with security

engineering principles. However, its evaluation is mainly methodological and based on specific scenarios, which means that broader validation in real industrial settings is still required [44].

Another prominent model is Secure Scrum, which is proposed by Pohl and Hof, and it focuses on identifying security needs without altering standard Scrum roles [14]. A key contribution of this model is the use of "S-tags" (Security Tags) to mark user stories that have security implications, ensuring they receive appropriate attention during planning and implementation [14]. This notation system enables the definition of security properties directly within the Product Backlog, making security visible to both developers and stakeholders.

The reviewed conceptual models mainly focus on integrating security practices into Scrum through modifications to artefacts and processes.

4.2.2 Challenges in Secure Agile Adoption

Despite these conceptual advancements, empirical studies consistently report ongoing challenges in their practical adoption. Across the literature, a consistent theme is that security work is often invisible, undervalued, and deprioritised compared to functional requirements. This section summarises the barriers and tensions faced by Agile and Scrum teams when implementing secure practices.

Structural Issues: One of the central challenges in integrating security into Scrum is the visibility of security work within Scrum artefacts and processes. Türpe and Poller explain that Scrum is optimised to fulfil explicit functional requirements and maximise business value, while security concerns are often implicit and treated as quality or non-functional requirements, which contributes to their low visibility [12]. Their analysis identifies three modes of managing security work in Scrum: (1) security as on-demand bug fixing, (2) as a quality requirement embedded in the DoD,

and (3) as a set of security-related features in the product backlog [12]. They argue that each of these approaches is inadequate on its own for achieving substantial and systematic security improvement [12]. When security is handled as on-demand fixing, it tends to remain reactive and unsystematic [12]. When addressed through backlog items, security features must compete with other requirements for limited attention and resources, and their implementation depends on backlog ordering and AC set by the PO [12]. If security concerns are not made visible and tangible in core Scrum artefacts, such as the product backlog or the DoD, development teams are unlikely to address them systematically, and security properties may be difficult to verify during sprint reviews [12]. This suggests a structural mismatch in how Scrum handles security as a non-functional requirement. At the same time, empirical studies also indicate that the definition, management, and tracing of security-relevant artefacts remain highly context-dependent and challenging in practice, as they are influenced by organisational processes, standards, and stakeholder interpretation. [45].

Decision-Making and Prioritisation: Beyond visibility, decision-making and prioritisation represent further challenges, which have been reported in broader Agile development contexts. Mohino et al. describe environments in which delivering functional products in the shortest possible time and with minimal resource consumption is prioritised, and they note that in such contexts, quality attributes, including security, may not be adequately considered [44]. They further state that in traditional methodologies, security is often introduced in later phases of development [44]. Across multiple studies, prioritisation mechanisms driven by business value often marginalise security requirements, especially when time, budget, and security expertise are constrained [12], [46]. At the level of prioritisation practices specifically, studies show that POs and other decision makers also factor in inputs such as implementation effort, cost, and constraints including release dates and

available resources [47]. Tøndel et al. further note the absence of clearly defined key performance indicators on security in PO procedures, which reduces the formal emphasis placed on security during prioritisation [47].

Process Tensions: These prioritisation dynamics directly contribute to observable tensions at the process level. The trade-off between agility and security has been empirically validated in several studies. For instance, researchers examine how the incorporation of security activities into Agile development affects practitioners' perceptions of both security and process outcomes [28]. In their survey of 34 practitioners, 97% reported using Agile methods, and 72% indicated that their teams include security-related activities in their Agile processes [28]. While practitioners generally consider security activities important and view them positively, many respondents report limited confidence in the overall security of the resulting software and remain uncertain about whether these activities meaningfully improve security outcomes [28]. Participants noted that security work introduces additional tasks, which are often perceived as increasing effort and potentially reducing delivery speed [28]. Participants reported a preference for integrating security activities into existing workflows. This, however, also highlights a constraint, as activities that are perceived as disruptive are less likely to be adopted, hence reinforcing the challenge of aligning security practices with Agile processes [28]. Although the authors do not conclude that security activities inherently undermine agility, their findings indicate a perceived tension in which teams attempt to strengthen security without compromising productivity or the light-weight character of Agile development [28].

Survey-based investigations provide further evidence of these process-level patterns. For example, Rindell et al. surveyed Finnish software professionals about their use of security engineering activities, many of which were drawn from BSIMM [3] and

[3]BSIMM stands for *Building Security In Maturity Model*. It is an observational framework that catalogues software security activities practised in real organisations to help assess and improve software security programmes; see `https://www.bsimm.com`.

Microsoft SDL [4], and compared the reported usage and perceived impact of these activities to BSIMM data [48]. The results indicate that some activities, such as penetration testing, were reported as having high perceived impact on security but were only seldom used systematically, and that, overall, perceived impact tended to be higher than reported use [48]. Difficulties in aligning security work with Agile settings are also discussed in empirical interview work by Bartsch, which highlights practitioner-reported challenges around requirements, awareness, and process adaptation [49].

Human Expertise and Relational Factors: Alongside process tensions, human and relational factors also pose significant barriers. Empirical evidence indicates a lack of sufficient security expertise among practitioners, which affects both requirement specification and implementation. In a grounded theory study, Terpstra et al. observed that security requirements are frequently poorly defined and owned, and teams sometimes respond by involving security experts as one of several coping strategies, thus raising questions about who truly owns security requirements in Agile projects [20]. Beyond a lack of technical expertise, cultural and relational tensions between developers and security experts represent a significant barrier. Empirical findings indicate that Scrum teams face conflicting priorities between business goals and security needs. This contributes to interaction difficulties, limited communication, and resistance between developers and security experts [46]. The specific impacts of these relational dynamics on team roles and governance are explored further under Governance and Organisational Adaptations (Section 4.2.4).

Contextual and Environmental Factors: Finally, the manifestation of these challenges varies across different environments. Following the introduction of the

[4]SDL stands for *Security Development Lifecycle*. The Microsoft SDL is a software security assurance process that defines security practices to be integrated throughout the software development lifecycle; see `https://www.microsoft.com/securityengineering/sdl`.

Brazilian General Data Protection Law (LGPD), empirical research combining an SLR, a survey, and semi-structured interviews reports that Agile teams recognise privacy as a relevant non-functional requirement and demonstrate awareness of LGPD principles [50]. However, the study identifies challenges in integrating privacy-related requirements into Agile practices. Specialised techniques and tools proposed in the literature are rarely used, and many teams rely on informal or ad hoc approaches. Practitioners also report difficulties in correctly specifying privacy requirements. These findings indicate that the systematic incorporation of privacy requirements into Agile development remains challenging in practice.

Large-scale Agile environments present particular difficulties in integrating security practices [51]. Van der Heijden et al. identify challenges that appear to be specific to large-scale settings, including the alignment of security objectives in distributed contexts and the development of a shared understanding of roles and responsibilities in security activities [51]. The study further reports unclear accountability for security actions, as well as issues in communication and documentation between security officers and Agile teams [51]. Fixed time and budget constraints were also described as limiting the attention given to security considerations [51]. Complementing these findings, Ascensão et al. show that although organisations adopt various security practices in large-scale Agile development, their implementation varies across cases [52]. Together, these results indicate that coordination, role clarity, and consistent implementation remain key challenges when integrating security into large-scale Agile environments.

Empirical studies in small and medium-sized enterprises (SMEs) indicate similar but context-specific challenges in adopting secure Agile practices. In their investigation of Agile security adoption in SMEs, Mihelic et al. [53] found that the absence of specialised roles and weak managerial commitment hinders the consistent implementation of secure development practices. Security-related roles and activities

aimed at strengthening security, such as specialised security teams or extensive risk management efforts, may increase costs and reduce agility, making them difficult for SMEs to adopt comprehensively. These findings also indicate that resource constraints limit the feasibility of adopting more resource-intensive security practices [53].

Across the reviewed studies, difficulties in achieving secure Agile development arise from interrelated structural, organisational, and cultural factors. These findings show how systemic challenges manifest in prioritisation practices, organisational structures, and resource constraints. These challenges are further amplified in large-scale environments due to coordination and accountability issues, and in SMEs due to resource constraints. Collectively, the evidence suggests that challenges in secure Agile adoption are not solely technical but are deeply rooted in visibility, prioritisation, and organisational alignment issues.

4.2.3 Mechanisms for Integrating Security into Scrum (RQ1)

In response to the previously identified structural, organisational, and process-related challenges, various mechanisms have been proposed in the literature to integrate security into Scrum while preserving agility. This subsection focuses specifically on artefact-level mechanisms that improve the visibility and management of security throughout the development lifecycle.

4.2.3.1 Artefact-Level Mechanisms

Artefact-level mechanisms extend Scrum artefacts to explicitly incorporate security, improving visibility and traceability of security requirements. They enable systematic management of security concerns alongside functional development throughout the lifecycle.

Security Backlog: Azham et al. propose a Security Backlog as an additional backlog element in Scrum, with the aim of integrating security principles into Scrum development phases and supporting security feature analysis and implementation explicitly [23]. In this approach, the Security Backlog is used to manage security risks and is guided by established software security principles, such as least privilege, defence in depth, and complete mediation, among others [23]. The Security Backlog is added alongside existing Scrum backlogs so that features in the Product Backlog pass through a security checkpoint, are "security-pruned," and have their security concerns carried into the Sprint Backlog for developers' attention, while the authors note that this extra backlog is intended to manage security risks without affecting the agility of the Scrum method [23].

The later work of Ghani et al. assesses the previously proposed Security Backlog concept in an industry-based Scrum case study, extending their earlier research on the topic [22]. Their study reported that integrating the Security Backlog into Scrum was associated with improved agility in the development process [22]. Using the 4-DAT agility framework [5], they measured an increase in the overall degree of agility for Scrum practices from 0.80 before integration to 0.83 after integration, and concluded that adding a Security Backlog did not negatively affect flexibility, speed, leanness, learning, or responsiveness [22]. The authors also observed that secure software could still be developed quickly, even when requirements changed, and that incorporating the Security Backlog into Scrum was feasible for Agile teams in their case study [22].

S-Tags and Security Labels: Pohl and Hof introduce the concept of "Secure Scrum," which uses a marking mechanism known as S-Tags to ensure traceability and explicit representation of security concerns throughout the development process

[5] Qumer and Henderson-Sellers' 4-DAT framework for evaluating the degree of agility in software development methods; see, for example, `https://doi.org/10.1016/j.infsof.2007.02.002`.

[14]. In their model, an S-Tag links a specific security concern to one or more Product Backlog Items (PBIs). To ensure clear representation, they use a visual marker called an S-Mark on relevant backlog items, making security-sensitive stories easy to distinguish from functional ones [14]. The S-Tag itself provides a detailed description of the security issue, which can take the form of user stories, misuse stories, or abuse stories (these concepts are explored further in the following paragraph), and may also link to an external knowledge base for mitigation guidance [14]. By using this tagging system, the authors aim to maintain a clear link between security risks and development tasks without altering the fundamental dynamics of the Scrum process [14].

Kagombe et al. describe S-tags as a concept that links security requirements to PBIs, thereby addressing the issue of managing security and functional requirements in separate requirement pools [54]. Their approach integrates security requirements directly into the Product Backlog instead of introducing a separate security backlog, which they argue would create multiple sources of requirements [54].

Subsequent work also identifies tagging-based mechanisms, such as S-Tags, S-Marks, and security-related keywords, as approaches that are used in Agile development to link security concerns with development artefacts and support their management within the process [55]. Mihelič et al. describe these tags as annotations on backlog items that highlight security relevance [55].

Security User Stories, Misuse Stories, Abuse Stories: Beyond tagging mechanisms that annotate existing backlog items, other approaches extend the structure of requirements themselves to explicitly represent security concerns. Ben Othmane et al. extend the Agile development process by integrating security engineering activities, including threat modelling, risk estimation, elicitation of security goals as security user stories, and security assurance cases, into the inception, construction, and transition phases so that functional and security user stories are selected to-

gether based on both risk assessment results and customer needs [56]. In their process, security goals are added as security user stories to the project backlog, security requirements are elicited using threat-based techniques, and evidence from security tests and reviews is linked to security claims within assurance cases to support traceability of security decisions across iterations [56].

Another complementary line of research in secure Agile development focuses on operationalising negative scenarios alongside functional requirements. Peeters introduced abuser stories as an informal extension of user stories to describe how attackers may abuse system functionality and to support the traceability of security requirements within Agile requirements engineering [57]. This work builds on earlier research on misuse and abuse cases, which emphasises anticipating abnormal or malicious behaviour and analysing how systems should respond to illegitimate use [58].

Subsequent research has explored a range of related artefacts and practices rather than focusing on a single unified approach. For example, Ardo identifies "defining evil user stories" as one of several security practices within a broader practice-based secure Agile process model, alongside other roles, ceremonies, and artefacts [59].

These approaches represent variations in how security requirements are expressed and managed within Agile artefacts.

DoD Extensions: Maier et al. propose a Secure Scrum process in which security is integrated through security-related user stories and abuse cases, derived from sources such as the OWASP Top 10[6], to systematically identify threats [60]. Within this framework, the DoD serves as a formal completion criterion, requiring that security concerns that are linked to user stories are verified before an increment is accepted [60]. This ensures that security verification is embedded directly into

[6]OWASP stands for *Open Worldwide Application Security Project*. The OWASP Top 10 is a standard awareness document representing a broad consensus about the most critical security risks to web applications; see `https://owasp.org/www-project-top-ten/`.

the development lifecycle rather than being treated as a post-development activity [60]. Empirical evaluations indicate that such approaches can be viable; specifically, Maier et al.'s process was rated as "medium" in agility and cost-effectiveness by practitioners, suggesting a manageable trade-off between security rigour and Agile flexibility [60].

Security AC: From a broader perspective, AC provide a way to define how system acceptability is evaluated. Neugent established the need for measurable AC to assess computer security quality, defining them as criteria that specify both the required level of quality and how that quality is evaluated [61]. In this sense, AC function as decision mechanisms for determining whether security requirements have been satisfactorily fulfilled, while also guiding developers in implementing appropriate levels of control quality.

In Agile software development, requirements are commonly expressed using lightweight artefacts such as user stories. In this context, Villamizar et al. report that Agile requirements engineering relies on minimal documentation and that requirements can be under-specified or too abstract, which may lead to issues in quality and correctness [62]. They further note that security requirements are often misunderstood or incorrectly specified due to a lack of expertise and insufficient emphasis in the early stages of development [62]. To address these issues, Villamizar et al. propose an approach for reviewing security-related aspects in Agile requirements specifications. Their approach considers user stories together with security specifications as input, relates them to security properties, and identifies high-level security requirements from OWASP to be verified [62]. This enables the detection of defects such as omissions, ambiguities, inconsistencies, and incorrect facts in security-related requirements [62].

Building on this perspective, Dalpra et al. operationalise the role of AC within Agile development through the Agile Security Framework (ASF), where security

requirements are systematically incorporated into user stories as feature-specific AC [63]. At the same time, the framework distinguishes these from general security requirements, which are enforced at the increment level through the DoD. This separation ensures that security validation occurs both at the story level, through explicit and testable conditions, and at the system level, through consistent quality checks applied to the increment.

4.2.3.2 Event-Level Mechanisms

This subsection examines mechanisms that embed security activities within Scrum events and iterative workflows. At a broader process level, some approaches integrate security into the overall iterative structure rather than into specific events. For example, Carlsson et al. present a security-enhanced Agile process in which activities such as security requirements engineering, risk analysis, and static code analysis are integrated into an incremental development model resembling Scrum iterations [24]. Rather than introducing new artefacts or event-specific practices, their approach embeds security activities within iterative planning and execution cycles, enabling security concerns to be addressed alongside functional requirements.

Threat Modelling Practices (e.g. Threat Poker): Integrating threat Modelling into Agile workflows requires light-weight, iterative methods that can be used naturally within Scrum's fast-paced environment. Traditional Threat Modelling frameworks, such as STRIDE [7] and PASTA [8], are too resource-intensive for short sprint cycles. This prompted the development of new frameworks that can be used with Scrum. One example is Threat Poker, which Rygge and Jøsang present

[7]STRIDE is a mnemonic for a threat classification model developed by Microsoft, standing for **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privilege. See: `https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats`

[8]PASTA is a risk-centric threat modelling methodology developed by VerSprite. See: `https://versprite.com/blog/what-is-pasta-threat-modeling/`

as a card-based game to be played in Scrum or other Agile team meetings to support discussion of security and privacy threats for new features and user stories [64]. Threat Poker is used by team members to estimate both security risk and privacy risk for a given threat scenario, as well as the effort required to remove vulnerabilities related to that scenario, by playing cards in two rounds [64]. In the first, risk round, cards are used to represent the assessed risk levels, and in the second, solution round, cards are used to represent the estimated implementation effort for possible solutions, with repeated rounds supporting convergence towards a shared assessment [64]. The procedure is inspired by Planning Poker, in that participants first select cards individually and then discuss any differing estimates until an approximate consensus is reached [64]. The authors also relate this method to GDPR's requirement to consider both security risks and privacy risks during software development and describe Threat Poker as a team-based method for secure systems development that addresses these two dimensions together in Agile projects [64].

Sprint Zero for Security Preparation: Kagombe et al. [54] introduce a security-focused Sprint Zero within Scrum that serves as an initial mechanism for incorporating security activities before iterative development begins. This Sprint Zero centres on a semi-structured risk analysis process, where stakeholders collaboratively identify threats, vulnerabilities, and potential impacts based on early system understanding and PBIs [54]. The outcome of this activity is the derivation of security requirements and mitigation strategies, which are then incorporated into the Product Backlog and linked to user stories for further development. Importantly, the authors emphasise that this process is not intended to produce exhaustive or static specifications, but rather to establish an initial security baseline that evolves across subsequent sprints [54]. In this way, Sprint Zero functions as a preparation phase that promotes early risk awareness, supports backlog seeding with security-related

items, and aligns stakeholders on security concerns before implementation begins, while allowing for continuous refinement during the Agile process.

4.2.3.3 Automation and Tool-Supported Integration

This subsection analyses automation mechanisms supporting security integration in development environments.

DevSecOps Practices within Sprints: Over the years, Agile and Scrum methods have evolved to support both flexibility and continuous delivery. Both researchers and practitioners have increasingly emphasised the automation of security activities within iterative workflows of Scrum. DevSecOps is proposed as a complementary approach to introduce automated assurance mechanisms directly into sprint cycles, supporting closer collaboration between development and security teams and reducing the incompatibility between rapid delivery speed and security verification [46]. Sharma and Bawa propose a framework for secure Agile development that is implemented in Java and automates the integration of selected security activities from established Security Engineering processes, including CLASP, Common Criteria, Cigital Touchpoints, and Microsoft's SDL [15]. The framework uses results from a SLR and a survey study, together with non-parametric statistical tests, to identify and select security activities that are both beneficial and suitable in terms of cost for different development phases [15]. These selected activities are then integrated with Agile activities using a light-weight dynamic integration algorithm, which is described as operating without compromising the agility of the process [15]. The Java implementation is reported to automate this integration process and to provide maximum benefit at a low integration cost, covering the project life cycle while treating security as an integral part of the development approach [15].

Static and Dynamic Analysis Tools: Ali and Ben Othmane describe a prototype under development that uses security assurance cases to represent the security of Zen Cart and to trace how code changes affect these cases [65]. As part of this work, they implement a small tool and a Penetration Testing Engine that automate a limited number of penetration tests for SQL injection and cross-site scripting (XSS) using Selenium and JSoup for Zen Cart version 1.5.4 [65]. The prototype also includes an Eclipse-based Security Assurance Case Plug-in for designing structured assurance cases, as well as a User Story Security Mapping Plug-in that links user stories, associated security requirements, and specified security tests, which the Penetration Testing Engine can execute based on XML-defined test cases [65].

Automated Prioritisation and Scoring Mechanisms: Voggenreiter and Schöpp propose a partially automated process for prioritising security findings in industrial Agile software development projects [66]. Their approach applies a two-stage scoring system: an initial Base Score is derived from security tools and other sources, which is then converted into a Refined Score using project-specific factors such as stakeholder influence and threat modelling results. By providing explanations of how scores are computed, the process supports transparent prioritisation. Evaluations through structured interviews and integration into industrial projects indicate that practitioners viewed the methodology as promising for reducing effort and improving transparency, with project leads willing to continue its use [66].

Ionita et al. propose a risk-driven framework that integrates security requirements into Agile development by combining a stakeholder-oriented risk assessment with a threat–feature mapping mechanism [67]. In this approach, risks are quantified based on likelihood and impact, mapped to predefined threats, and processed to generate a prioritised list of operationalised security requirements that can be imported directly into Agile tracking tools like JIRA [67]. While evaluations across five domains at Centric B.V. showed that the framework improved risk awareness

among non-experts, comparisons revealed that its automated prioritisation often differed from expert judgment and did not perform significantly better than random overall, although it remains useful for providing an initial indication of priorities [67].

4.2.4 Governance and Organisational Adaptations (RQ2)

While RQ1 examined how security can be embedded into Scrum through artefacts, events, and technical practices, addressing RQ2 requires shifting the focus from integration mechanisms to the organisational and governance conditions that support their sustained use. The literature consistently shows that the successful implementation of security practices in Scrum is not only a matter of introducing new artefacts or activities, but also depends on how responsibilities are assigned, how decisions are coordinated, and how security expertise is integrated within Agile teams.

In this context, RQ2 examines the additional roles, checkpoints, and process adaptations that support continuous security assurance in Scrum-based development. These adaptations operate at the governance level and shape how security is prioritised, communicated, and enforced across iterations. Rather than directly modifying Scrum artefacts, they influence the organisational structures and interaction patterns that determine whether security practices are applied consistently in practice.

The reviewed studies suggest that achieving continuous security assurance requires three interrelated forms of adaptation: (1) role-based mechanisms that embed or distribute security expertise within teams, (2) coordination and checkpoint mechanisms that ensure the visibility and prioritisation of security work, and (3) broader organisational adjustments that address factors such as scalability, resource constraints, and team capabilities.

The following subsections synthesise the literature along these dimensions, begin-

ning with role-based mechanisms and their impact on security integration in Scrum environments.

4.2.4.1 Role-Based Mechanisms

This subsection reviews security-related role adaptations in Scrum.

Security Champion/Advocate: At the governance and coordination level, Tøndel et al. analysed how security prioritisation is shaped within Agile contexts [47]. Through a longitudinal case study in an SME, they developed a model of influences on security prioritisation. The study describes how a Security Officer, together with an established Security Champion role, introduced a Security Requirements Initiative to support the elicitation, documentation, prioritisation, and follow-up of security requirements, and how security experts sought to influence prioritisation through interaction and support rather than through prescribed authority. Their analysis further shows that POs may become hindrances when security is treated as an add-on and is not embedded in procedures or key performance indicators. While formal security roles are introduced to improve coordination, studies also highlight limitations in their scalability and integration within Agile teams. Extending this governance perspective to smaller enterprises, Mihelič et al. developed and evaluated the ATTRACT approach as a light-weight, delegation-based governance add-on in an SME context, in which security responsibilities are distributed among existing team members and security-related decisions are guided by evaluations of their impact on security, cost, and agility rather than by formalised authority structures [55].

In contrast to SME-oriented governance approaches that emphasise light-weight and distributed coordination, Nägele et al. report a six-month single-case study in one of the world's largest financial organisations to examine the adoption of Agile security approaches in a large-scale Agile development environment [68]. They iden-

tified 27 Agile security approaches from academic literature, found 11 of these in use at the company, and evaluated 14 selected approaches through expert interviews and a survey [68]. The study shows that activities that are easy to integrate, such as a security tagging system, peer security code reviews, security stories, and threat poker were considered particularly beneficial in this context [68]. It also shows that role and knowledge based approaches, such as dedicated security roles (including Security Champions) and community structures like the security guild, are considered especially important in scaled Agile environments for building and sharing security knowledge [68]. In addition, the authors describe regulatory and organisational conditions of the regulated finance setting and identify multiple drivers and obstacles for adopting Agile security approaches, but they do not claim a general resolution of tensions between security and agility beyond the studied organisation [68].

Security Master/Security Officer: The Security Master role has been proposed to take responsibility for identifying features with security risks, documenting associated risks in a Security Backlog, and conducting security testing [23]. An industrial study at Ericsson AB reports the introduction of the Security-Enhanced Agile Process (SEAP) in the development of a mobile money transfer system that handles large amounts of money and is treated as security-critical [69]. SEAP extends the existing Agile method by adding a security group with four roles (security manager, security architect, security master, penetration tester) and by performing risk analyses per Business Use Case within sprints, allowing the security group to act on serious issues in ongoing deliveries [69]. The study reports that, relative to the earlier Agile process, SEAP reduced unattended risks from 50% to 22%, raised corrected risks from 12.5% to 67.1%, and lowered average analysis hours per corrected risk from 21.6 to 1.72, while increasing explicit security personnel cost from about 1% to about 5% of the project, which the authors describe as a justified investment in light of the improved security-related results [69].

Security Teams and Specialists: At the individual level, Oyetoyan et al. examined software security usage, competence, and training needs through a survey in two Agile organisations [70]. Their findings demonstrate that skill level is strongly correlated with the use of security activities and that secure design represents the most important training need across roles. In particular, the organisation that had established a security expert group exhibited higher security awareness and performed more core security activities, such as secure design and coding. The authors argue that effective security adoption in Agile settings is not automatic but requires a driver, and that skill rather than process is the primary determinant of sustained security practice adoption.

Beyond formal role definitions, the way in which developers and security experts interact and communicate also shapes how security responsibilities are carried out in practice. This relational aspect of security governance in Agile settings has gained increasing attention in recent research. Naji et al. conducted a qualitative interview study involving 14 developers and 13 security experts working in Scrum environments to explore these dynamics [46]. The findings show that relationships between developers and security experts are often characterised by interaction difficulties, limited direct communication with reliance on indirect communication mechanisms, and conflicting priorities between business goals and security needs [46]. Furthermore, the study finds that key Scrum values, such as openness, respect, and courage, are often missing in these interactions, which contributes to collaboration challenges [46]. These tensions contribute to viewing security as a barrier to development rather than a shared responsibility, hence indicating that effective security integration depends not only on technical expertise but also on better collaboration between developers and security experts [46].

4.2.4.2 Coordination and Checkpoint Mechanisms

Beyond role-based adaptations, the literature highlights the importance of recurring coordination mechanisms that act as checkpoints for maintaining continuous security assurance in Scrum. These mechanisms ensure that security concerns remain visible, are regularly prioritised, and are revisited throughout iterative development cycles.

Security Prioritisation Meetings: Tøndel and Cruzes' study on continuous software security through security prioritisation meetings introduces regular, security-focused meetings as a way to support ongoing security work in Agile development contexts [71]. Their investigation of three small and medium-sized companies indicates that such meetings, held on a regular basis, can reach key stakeholders, make security issues more visible, and contribute to continuous security prioritisation [71]. In these meetings, participants review the current state of security, discuss security-related needs and concerns, and agree on concrete actions for the next period, with roles such as POs, developers, architects, and managers involved rather than only security specialists [71]. The approach builds on the Security Intention Meeting Series concept, which focuses on intention setting, status assessment, and follow-up, and is designed to fit Agile practices that rely on meetings for coordination and problem solving [71]. The authors report that, especially in small and medium-sized development companies with basic security competence, these meetings can offer benefits such as increased visibility of security work and support for ongoing prioritisation of security alongside other development concerns [71].

Security Considerations in Sprint Planning: Rindell et al. mapped security engineering activities from Microsoft SDL, the ISO/IEC Common Criteria, and OWASP SAMM to commonly used Agile processes, practices, and artefacts, using terminology from Scrum [72]. Although the mapping covers multiple Agile events, iteration planning is presented as one of the main points for coordinating security-

related activities. The mapping includes design-time activities such as threat modelling, vulnerability analysis, and design verification, which are linked to Agile structures like iterations, iteration planning meetings, daily meetings, and iteration reviews or retrospectives across the security development lifecycle phases [72]. Iteration planning meetings and the product backlog are described as key points for defining and managing security requirements, while daily meetings and continuous integration help coordinate implementation and testing work, including security-related tasks [72]. Iteration reviews and retrospectives are presented as general quality improvement activities that also apply to security engineering and can be revisited at the end of each iteration [72]. Overall, their study describes a framework in which security engineering activities are integrated into recurring Agile activities so that security objectives, derived from the Common Criteria, can be addressed together with business objectives within the iterative development process [72].

4.2.4.3 Organisational and Process-Level Adaptations

In addition to roles and coordination mechanisms, the literature shows that effective security integration in Scrum requires changes to the underlying development process itself. These changes focus on how security activities are structured, sequenced, and embedded within Agile workflows, rather than on individual roles or isolated practices.

Process Misalignment Between Agile and Security Engineering: Organisational and process-level adaptations go beyond the introduction of roles or coordination mechanisms and instead focus on how security is structurally embedded within Agile delivery models. The literature suggests that Agile development and security engineering often follow conflicting approaches, where Agile methods are value-driven, iterative, and light-weight, while security engineering is more risk-driven and often organised around planned sequences of activities [72].

Process Alignment Through Integrated Security Activities: As a result, organisations may run a separate security development lifecycle alongside Agile development. Prior work argues that security objectives should be integrated directly into Agile delivery rather than treated as external activities. In this context, activities from Microsoft SDL, OWASP SAMM, and the ISO Common Criteria have been mapped into Agile processes, practices, and artefacts to support security throughout iterative development cycles [72].

Embedded Risk Management in Iterative Development Processes: At the organisational level, the industrial study by Baca et al. highlights structural challenges in integrating security into Agile development [69]. In the Security-Enhanced Agile Process (SEAP), security is embedded through an integrated risk analysis process that operates within development iterations rather than as a separate phase, hence allowing earlier identification and resolution of risks [69]. This type of integration enables organisations to move away from reactive "after-the-fact" security handling toward systematic, iteration-level risk management while maintaining development flow.

Process Adaptation Under Resource Constraints: In resource-constrained environments such as SMEs, software development processes are influenced by limited capacity and financial resources. Security mechanisms may introduce additional overhead and can affect the agility of development processes. To address these challenges, prior work proposes a structured approach for evaluating and selecting security elements, such as roles, activities, and artefacts, based on their measured impact on key dimensions, including provided security, cost efficiency, and retained agility. This evaluation enables organisations, particularly SMEs, to assess individual security elements and determine their suitability for integration into existing Agile

development practices without adding excessive overhead or reducing development efficiency [53].

4.2.5 Domain-Specific and Contextual Variations

While the previous subsection identified general organisational requirements for continuous security assurance, empirical studies indicate that their implementation varies significantly across domains and organisational contexts. Scrum and Agile practices have been applied in domains such as finance, healthcare, and SMEs, each characterised by distinct regulatory, organisational, and technical conditions. These differences demonstrate the adaptability of secure Agile frameworks while also highlighting domain-specific constraints that shape their implementation and scalability.

Cost–Agility Trade-offs in SME Security Integration: While large-scale regulated organisations institutionalise security through formal roles and governance structures, research on SMEs (SMEs) highlights that such organisations often face limited capacity and financial resources, as previously identified in SME contexts (see Section 4.2.2). As a result, they benefit from security method elements that help balance security, agility, and cost when adapting Agile methods [53]. Work by Mihelič et al. further indicates that SMEs can systematically evaluate different security roles, activities, and artefacts based on their impact on these dimensions, enabling the selection of context-appropriate security elements that align with organisational constraints while still supporting secure software development [53].

Secure Scrum Adaptation for Smart Aging Care Systems (SACS): Chakraborty et al. propose Scrum-based guidelines for secure software development in Smart Aging Care Systems (SACS), an emerging domain where smart IoT technologies are used to support older adults [73]. Their work adapts security controls from established standards and recommendations, including the NIST Cybersecurity

Framework, ISO/IEC 27001 and 27002, IEC 62304, and FDA cybersecurity guidance for medical devices, so that these controls can be applied within a Scrum process for SACS [73]. The guidelines organise these adapted controls into activities aligned with Scrum events, such as feature backlog definition, sprint planning, implementation and review, and sprint retrospectives, with a focus on continuously identifying and addressing vulnerabilities and security-related actions throughout the software lifecycle [73]. The paper also describes domain-specific human roles involved in SACS security, including Trusted Immediate contacts, service providers, and emergency responders, and explains how they contribute to secure operation and maintenance [73]. In addition, the authors incorporate usability and privacy considerations for older adults by linking their Scrum-based method to human factors engineering guidance and NIST security and privacy controls, suggesting that these guidelines provide structured support for applying Agile practices to security-sensitive SACS software [73].

Regulatory Influence on Security Engineering Practices in Agile Contexts: Rindell et al. report an industrial survey of software and security professionals in Finnish software companies to examine security engineering practices in the context of Agile software development processes and to compare these practices with BSIMM and Microsoft's SDL [48]. The authors state that the security engineering activities currently used in Finland are largely consistent with BSIMM-based practices observed in the global software industry. They also indicate that regulation plays a major role in driving security engineering in software development [48].

Taken together, these domain-specific studies show that adapting Agile and Scrum for security is feasible across a range of different contexts, from highly regulated industries to smaller, resource-constrained organisations. However, the findings also highlight limits to generalizability, as each sector has its own priorities and constraints. Overall, the evidence suggests that while Scrum can be adapted for

security-sensitive domains, its effectiveness depends on contextual tailoring rather than applying a single universal framework.

4.2.6 Synthesis of Secondary Studies

A growing body of secondary research, including SLRs, mapping studies, and broad practitioner surveys, has helped bring together the previously fragmented understanding of how security fits within Agile and Scrum methods. Collectively, these studies provide an evidence-based foundation that clarifies the main challenges, compares different integration frameworks, and highlights the research gaps that continue to shape this evolving field.

Datta et al. presented a structured literature survey on security and human-related challenges in Agile software deployment, focusing on large-scale Agile development [74]. In this survey, 16 peer-reviewed studies published between 2015 and 2020 were selected using defined inclusion and exclusion criteria, and 11 human-related challenges for scaling Agile methodologies were identified, including lack of training, insufficient communication, limited customer involvement, unclear roles and responsibilities, and lack of management commitment [74]. The review also reports that security practices in Agile software development can be difficult to adopt because security activities tend to be given lower priority or their relevance is not clearly perceived, and it highlights additional challenges related to Agile security and organisational factors [74]. Overall, the study summarises recurring human and organisational challenges, along with reported solutions, and identifies critical barriers to adopting Agile methodologies such as software project management problems and lack of requirements [74].

Oueslati et al. conducted an SLR to identify the reported challenges of developing secure software using Agile methods and to evaluate the causes of these challenges with respect to Agile values, Agile principles, and security assurance practices [7].

Their review summarised 20 challenges drawn from 10 publications and found that 14 of these were valid according to their criteria, while 6 were not attributed to Agile values, principles, or security assurance practices [7]. The 20 challenges were organised into five categories: software development life-cycle challenges, incremental development challenges, security assurance challenges, awareness and collaboration challenges, and security management challenges [7]. The analysis indicates that characteristics of Agile methods such as short iterations, frequent changes to requirements and code, and a preference for light documentation are associated with difficulties in areas like traceability, security verification and validation, and the use of detailed documentation in security assessment [7]. The authors report that security activities, including risk assessment and various assurance practices such as security testing, need to be performed in each iteration, and that the limited duration of iterations can make these time-consuming activities difficult to accommodate [7]. They conclude that developing secure software using Agile methods is challenging and state that their results justify the need for further research and for adapting Agile development methods and security assurance practices so that secure software can be developed more smoothly using Agile approaches [7].

Rindell et al. conducted an empirical practitioner survey to examine the use and perceived impact of software security engineering activities in the context of Agile software development [8]. The study analysed answers from 61 software practitioners in Finland about their use of 40 security engineering activities together with 16 Agile software development items and practices, and about the perceived security impact of these activities [8]. The security activities were drawn from security development and maturity models including the ISO/IEC Common Criteria, the Microsoft Security Development Lifecycle (SDL), BSIMM, and the Finnish VAHTI guidance [8]. The results indicated that Agile items and activities had a measurable effect on the selection of security practices, and that for many activities the perceived security

impact was lower than the rate of use, pointing to a discrepancy between how often practices are used and how effective they are considered to be [8]. The authors reported that proactive security activities, especially those in the early phases of the life cycle, such as requirements and implementation, were both widely used and often seen as impactful. In contrast, activities such as static analysis tools and some review practices showed notable gaps between how often they were used and how beneficial they were perceived to be [8]. Overall, Rindell et al. observed that security activities were most common in the requirements and implementation phases and that the identified discrepancy between usage and perceived impact motivates further methodological work on integrating security engineering activities into Agile software development processes [8].

Selva-Mora and Quesada-López carried out a mapping study to identify and categorise security practices used in Agile software development [75]. Their study reports 252 security practices extracted from 35 primary studies, which are organised according to BSIMM security practices and mapped to stages of the Software Development Life Cycle (SDLC) [75]. The authors report 38 benefits, covering areas such as security awareness, alignment with agility, and team organisation, and 95 challenges, including lack of knowledge and complexity, associated with these practices [75]. They further note that many of the practices appear in proposals, theoretical models, experiments, examples, and scenarios, and that about 65.7% of the papers discuss practices not applied in real development environments, which they indicate makes it difficult to understand the actual impact of these practices in industry and highlights the need for more empirical evaluation in real-world Agile projects [75].

Riisom et al. conducted a literature review to examine challenges and solutions related to integrating software security in Agile software development contexts [76]. The review included 16 primary studies. Their results indicate that there are vari-

ous reported efforts to incorporate security practices into Agile methods, yet security integration still poses difficulties and the identified activities are diverse and fragmented across different development phases. Commonly reported challenges include ensuring that security activities fit Agile principles, dealing with security tasks that require early planning and design, and addressing issues such as lack of experience, knowledge, and awareness regarding security among developers. The authors compiled a set of mitigation strategies, for example assigning specific security-focused roles, weighting security activities to support selection decisions, using misuse stories, and applying iterative and incremental risk analysis as well as other security-related practices. At the same time, they note that some documented challenges, such as time constraints, intangible security, incomplete security requirements, first-time integration of security, missing security knowledge, prioritisation of security quality, and continuous motivation to fix vulnerabilities, have no clearly mapped solutions in the reviewed studies. Overall, the study concludes that security is present in Agile software development but continues to create problems, and that not all integration challenges have been resolved, suggesting that further research on unsolved challenges and their solutions is needed [76].

Rindell et al. state that their objective is to examine the widely discussed "myth" of an inherent incompatibility between Agile methods and traditional security processes by conducting an SLR of Agile software development methods with explicit security engineering activities [6]. They identify a core set of 11 secure-Agile methods, extract the documented security activities, and organise these activities into a six-phase software development lifecycle, showing that the observed security work spans all phases of the security development lifecycle [6]. The reviewed methods draw extensively on established security processes such as OWASP CLASP, Microsoft SDL, Cigital Touchpoints, and ISO Common Criteria as sources of security activities that can be adapted and integrated into Agile development, and the au-

thors argue on this basis that the notion of the "inherent insecurity" of Agile software development can be regarded as a myth rather than a substantiated property of Agile methods [6].

Finally, Daneva and Wang reported on a documentary multi-case study that examined seven Agile security requirements engineering frameworks used in industrial settings [77]. Through qualitative thematic analysis of the available documentation, they discerned 46 solution practices that organisations propose for integrating security requirements into Agile project delivery [77]. These practices focused on introducing additional artefacts (such as security-oriented story types and risk-related documentation), organisational and technical security roles, specific competencies, and dedicated security-related activities, so that security requirements engineering is treated in a more systematic way in Agile projects [77]. Their results indicate that such practices add to project documentation and highlight the importance of security training for development teams and other stakeholders [77].

Taken together, these secondary and mapping studies reveal several clear patterns. First, human and organisational factors, such as limited security expertise, tight delivery schedules, poor visibility of security work, and weak communication, continue to be the main challenges in secure Agile adoption. Second, there is growing alignment between Agile practices and established security maturity models such as BSIMM[9], SAMM[10], and SSE-CMM[11], which are increasingly used as benchmarks for structured integration. Third, although the conceptual maturity of this research field is high, long-term industrial validation and large-scale empirical studies remain limited. Finally, the literature consistently points to a shared set of practices, including artefacts, such as Security Backlogs, S-Tags, roles, such as Security Champions, and process adaptations, such as Sprint Zero, secure retrospectives, as the most

[9]Building Security In Maturity Model; see `https://www.bsimm.com/`

[10]Software Assurance Maturity Model; see `https://owaspsamm.org/`

[11]Systems Security Engineering Capability Maturity Model, standardised as ISO/IEC 21827.

practical means of embedding security in iterative workflows. Overall, this body of secondary research demonstrates that Agile and security are not inherently conflicting, but their integration can be achieved depending on structured adaptation, cultural readiness, and empirical grounding. The synthesis of insights from these studies provides a foundation for moving secure Agile development from conceptual discussion toward validated and repeatable practice across different domains and organisational contexts.

4.2.7 Research Gaps and Limitations

Terpstra et al. conducted a qualitative grounded theory study to explore how Agile practitioners reason about security requirements and the strategies they use to address them, analysing posts from two professional LinkedIn discussions in the "Agile and Lean Development" group [20]. Their analysis indicates that existing proposals for handling security requirements in Agile projects have largely been developed in academic settings with little practitioner involvement and limited empirical evaluation in real-life contexts [20]. The study suggests that practitioners participating in these discussions tend to consider contextual factors such as ownership of security requirements, the definition of "done", the business case, team attitudes, organisational setup, and differing perceptions of priority when dealing with security-related issues [20]. It also reports that practitioners describe several coping strategies. These include embedding security concerns into Agile artefacts, such as the DoD, user stories, estimates, and AC. They also involve adjusting roles and improving awareness through education and support, as well as introducing security-focused practices such as prioritising security risks, conducting code reviews, and using automated checks [20]. Overall, the authors highlight that the observed solution strategies emphasise people and non-technical aspects rather than complex, mathematically

grounded techniques, and they point to the need for further empirical work to understand security requirements engineering in Agile settings more deeply [20].

Mohino et al. proposed a Secure Software Development Life Cycle (S-SDLC) that integrates security-related activities into all phases of software development while leveraging Agile practices such as short iterations and collaborative work [44]. The model includes activities like defining abuse cases, performing threat modelling and risk analysis, applying secure coding guidelines, and promoting systematic verification and security testing across the life cycle [44]. Their work draws on a prior comparative analysis of multiple secure development processes and maturity models, including BSIMM, CLASP, and Microsoft SDL, which are used as references when defining the new S-SDLC [44]. The proposed methodology is illustrated using a checklist, an application scenario, and an expert-based assessment, rather than reports of industrial deployment or detailed measurements from real projects [44]. The article describes these validation steps but does not present empirical industrial trials or a comprehensive quantitative evaluation of the S-SDLC in operational Agile environments [44].

Overall, the reviewed studies show that research on integrating security into Agile methods is conceptually rich but offers limited empirical validation. While many frameworks, such as ScrumS, Secure Scrum, and S-SDLC adaptations, present promising theoretical models, most have not been validated beyond academic or small-scale contexts. A recurring issue is that, although these models provide strong conceptual depth, they offer limited empirical evidence of their effectiveness and often rely on selective or controlled evaluations. Consequently, this body of work underscores the need for systematic empirical studies that validate these security extensions to Scrum across different domains and organisational scales [14], [21], [44]. Furthermore, while automation mechanisms offer potential to alleviate manual security work, their adoption and effectiveness remain highly uneven across differ-

ent Agile contexts. The field still lacks long-term empirical studies, cross-industry comparisons, and standardised metrics that could demonstrate measurable improvements in software security. This imbalance between conceptual design and real-world validation underscores the field's early stage of maturity. Future research should focus on systematic empirical testing, industry collaboration, and unified evaluation criteria to advance secure Agile development from theory to consistent, evidence-based practice.

4.2.8 Thematic Analysis

This section presents a thematic synthesis of the reviewed studies, focusing on identifying recurring patterns in research approaches and reported security practices. The analysis moves beyond descriptive classification by showing how security integration in Agile/Scrum is conceptualised and applied across studies. The resulting themes provide a structured basis for understanding common strategies, variations, and gaps in the literature.

4.2.8.1 Study Classification by Research Method

Studies were classified as theoretical, empirical, or mixed based on whether they (1) proposed conceptual models without validation, (2) reported empirical data from case studies or surveys, or (3) combined both conceptual proposals and empirical evaluation. Borderline cases, such as those relying on scenario-based evaluations or prototype demonstrations, were classified as mixed.

Table 4.3 summarises the methodological distribution of the 42 primary studies. Mixed-method studies constitute the largest proportion (50%), followed by empirical-only (31%) and theoretical-only studies (19%).

The predominance of mixed-method research suggests that many studies attempt to combine conceptual proposals with some form of validation. However, the nature

Type	Count (%)	Papers
Theoretical-only	8 (19%)	[72], [73], [12], [23], [44], [61], [57], [58]
Empirical-only	13 (31%)	[68], [20], [51], [70], [50], [78], [52], [49], [46], [28], [8], [48], [45]
Theoretical & Empirical	21 (50%)	[69], [54], [53], [24], [71], [55], [56], [15], [47], [22], [66], [21], [64], [60], [65], [67], [79], [14], [62], [63], [59],

Table 4.3: Method basis of primary studies (n=42)

and rigor of these evaluations vary widely across studies. Empirical studies provide insights into real-world practices, while theoretical studies mainly contribute conceptual frameworks without direct validation.

4.2.8.2 Identification of Security Practices

Security practices were identified using explicit inclusion criteria. Practices were included if they were either (1) proposed as structured mechanisms for integrating security into Agile/Scrum, or (2) empirically observed in use within the reviewed primary studies.

Practice	Supporting Papers
Core Artefact Practices	
Security Backlog / Repository	[69], [54], [68], [53], [72], [73], [55], [56], [15], [47], [22], [12], [23], [44], [64], [67], [79], [59]
S-Tags / S-Marks	[54], [68], [53], [55], [14]
Security User Stories	[69], [54], [68], [53], [20], [55], [56], [50], [52], [21], [8], [64], [60], [65], [67]
Misuse / Abuse Stories	[68], [53], [24], [70], [55], [56], [78], [15], [8], [48], [44], [64], [60], [79], [14], [57], [59], [45], [58]
DoD (with security checks)	[54], [68], [20], [12], [49], [46], [21], [64], [60], [14], [63]
Security AC	[69], [20], [47], [60], [65], [67], [61], [63],

Practice	**Supporting Papers**
Security Assurance Cases	[54], [55], [56], [65], [45]
Security Guidelines / Policy Artefacts	[69], [68], [53], [73], [55], [56], [50], [78], [47], [22], [46], [21], [8], [48], [44], [60], [65], [62], [61], [63], [59], [45]
Process / Event Practices	
Sprint Zero (Security Preparation)	[54], [21]
Security Integration in Sprint Planning	[69], [54], [68], [53], [73], [55], [56], [50], [52], [12], [46], [21], [44], [64], [60], [67], [79], [14], [63], [57], [59]
Threat Modelling Workshops	[69], [54], [68], [53], [24], [72], [70], [71], [55], [56], [78], [22], [52], [46], [8], [48], [44], [64], [60], [65], [67], [79], [63], [57], [59], [58]
Security Considerations in Retrospectives	[54], [24], [72], [73], [52], [12], [46], [21], [60]
Governance Practices	
Security Champion / Advocate	[69], [68], [70], [71], [55], [47], [22], [52], [46], [23], [44], [64]
Security Team / Specialist Group	[69], [54], [53], [70], [73], [71], [55], [56], [50], [47], [46], [28], [48], [44], [60], [79], [59], [58]
Security Prioritisation Meetings	[71], [55], [50], [47], [12], [66], [44], [60], [79], [59]
Security Decision Influence Mechanisms	[69], [54], [53], [73], [71], [55], [56], [47], [22], [12], [66], [46], [28], [48], [44], [64], [60], [65], [79], [57], [59]
Automation & Technical Practices	
DevSecOps Automation (in-sprint scans)	[69], [54], [68], [24], [20], [72], [51], [55], [56], [50], [22], [52], [49], [66], [46], [28], [8], [44], [60], [65], [79], [63], [58]

Practice	**Supporting Papers**
Automated Prioritisation	[66], [67], [63]
Security Training / Awareness	[69], [54], [68], [53], [20], [72], [51], [70], [73], [71], [55], [56], [50], [78], [15], [47], [52], [12], [49], [46], [28], [23], [44], [64], [79], [14], [62], [59], [45], [58]

Table 4.4: Security practices in Agile/Scrum and supporting primary studies

Table 4.4 details the security practices identified across the 42 primary studies included in this review. The identified practices are organised into four categories: core artefact practices, process and event practices, governance practices, and automation and technical practices. This classification reflects different integration points of security within Agile development.

Artefact-based practices are widely represented in the reviewed studies as mechanisms for incorporating security concerns into Agile development. As detailed in Table 4.4, commonly reported examples include the Security Backlog or Repository, which is used to manage security risks and mitigation tasks alongside standard development work. Other identified techniques include misuse or abuse stories, which are used to represent potential malicious behaviour, and extensions of the DoD to include security-related checks. These findings indicate that artefact-level mechanisms are frequently used to address the visibility of security requirements, which was identified as a recurring challenge in Agile development (see Section 4.2.2). The strong representation of backlog-based and story-based approaches suggests that integrating security into existing Scrum artefacts is a preferred strategy, as it aligns with Agile principles while minimising disruption to established workflows.

Process and event-based practices focus on incorporating security activities within Scrum ceremonies and iterative workflows. Practices outlined in Table 4.4,

such as Sprint Zero, threat modelling workshops, and the inclusion of security considerations in sprint planning and retrospectives are described in the literature as approaches for addressing security concerns throughout development cycles. These practices reflect an effort to embed security-related activities into existing Scrum events rather than introducing separate processes, suggesting that iterative and discussion-based mechanisms are used to support continuous identification and reassessment of security risks.

Governance-related practices emphasise the assignment and distribution of security responsibilities. As Table 4.4 indicates, the introduction of Security Champion or Advocate roles is widely reported, alongside the use of dedicated security teams or specialist groups and security-focused meetings for prioritisation. These practices are presented as organisational mechanisms for coordinating security-related activities within Agile environments. The presence of these governance mechanisms shows that security integration is not only a technical issue but also requires organisational coordination and clearly defined roles, aligning with earlier findings that highlight the importance of expertise, communication, and prioritisation structures in addressing security challenges.

In the area of automation and technical practices, the literature frequently reports the use of DevSecOps approaches, including the integration of security scans within development pipelines (see Table 4.4). In addition, numerous studies highlight the role of continuous security training and awareness in supporting the adoption of security practices within development teams. These findings suggest that automation and training mechanisms support continuous security assurance by reducing manual effort and increasing team awareness. However, their effectiveness may depend on the organisational context and the availability of technical expertise, as noted in earlier sections.

Overall, the identified practices show that security integration in Scrum is ad-

dressed through a combination of artefact, process, governance, and automation mechanisms. While this section has categorised and described these practices, the following section synthesises the findings to identify cross-study patterns, trade-offs, and limitations.

4.3 Synthesis of Findings

This section synthesises the findings from the reviewed primary studies and the thematic analysis to answer RQ1 and RQ2. While the previous sections focused on describing and categorising practices, this section integrates the results across studies to identify consistent patterns, trade-offs, and gaps in the literature.

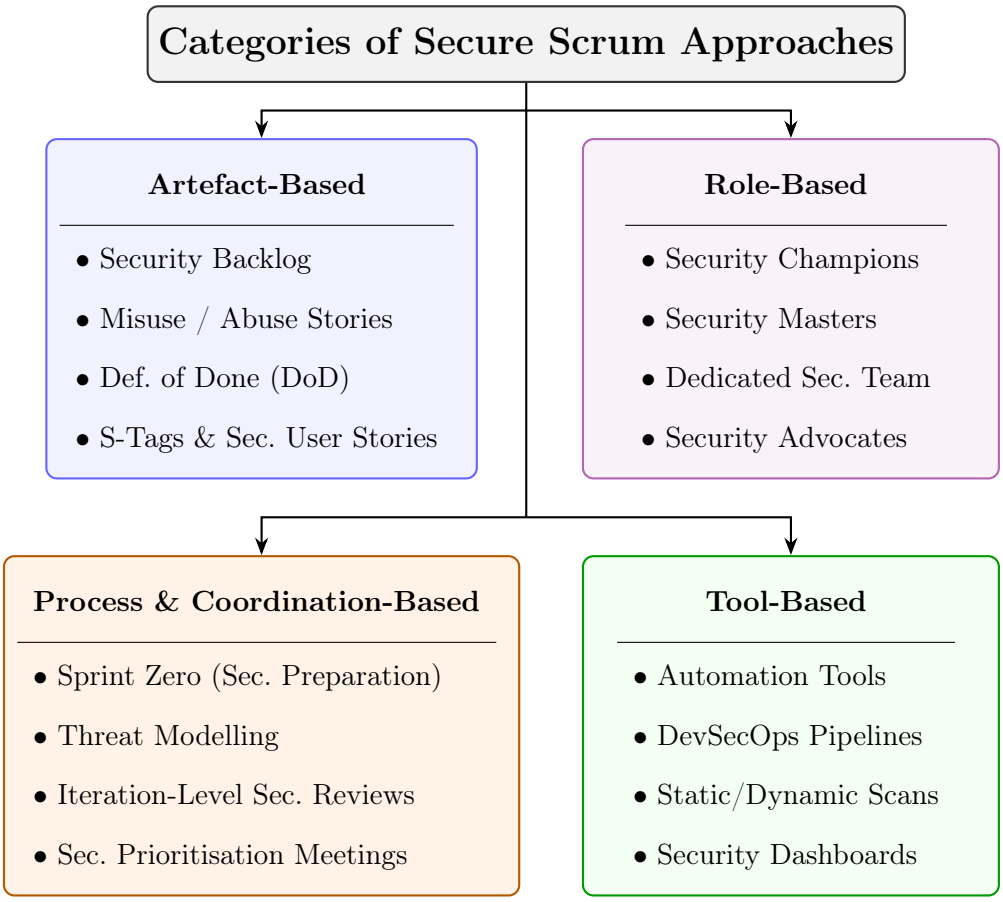

Figure 4.2: Taxonomy of Secure Scrum approaches identified in the literature

The taxonomy of secure Scrum approaches identified in the reviewed literature is presented in Figure 4.2. The classification groups mechanisms into artefact-based, process- and coordination-based, role-based, and tool-based categories, reflecting different integration points for security practices within Scrum artefacts, events, and organisational structures. Artefact-based approaches mainly improve the visibility of security requirements, while process and coordination mechanisms embed security activities into iterative workflows. Role-based mechanisms support expertise and governance, and tool-based approaches enable the automation of security assurance activities. Overall, the taxonomy shows that effective security integration depends on a combination of complementary mechanisms rather than a single approach.

4.3.1 Synthesis for RQ1: Integration of Security into Scrum

The reviewed studies show that security integration in Scrum is mainly achieved through a combination of artefact-level, process-level, and tool-supported mechanisms. Across the literature, artefact-based approaches emerge as a central mechanism for embedding security within the Agile workflows. These include practices such as Security Backlogs [54], [22], [23], [67], S-Tags [14], and security user stories [56], [63]. They enable security concerns to be represented within core Scrum artefacts, particularly the Product Backlog and the DoD. These approaches address the recurring issue of security visibility identified in multiple studies, where security requirements are often treated as implicit or secondary concerns.

Process-based and event-based mechanisms complement artefact-level integration by incorporating security activities within iterative development cycles. These include practices such as Sprint Zero [54], threat modelling approaches such as Threat Poker [64], and the inclusion of security considerations in sprint planning and retrospectives [72], hence enabling continuous identification and discussion of

security risks. These mechanisms reflect an effort to adapt traditional security engineering practices to the short and iterative cycles of Scrum.

In addition, automation-oriented approaches, particularly DevSecOps practices [15], aim to reduce the perceived trade-off between development speed and security assurance. Automated testing, static analysis, and continuous integration pipelines are used to integrate security verification into regular development workflows. However, the literature also indicates that the effectiveness of these automation mechanisms varies across contexts and is not consistently adopted.

Overall, the integration of security into Scrum is characterised by the combination of light-weight artefact modifications, iterative security activities, and automation mechanisms, all of which aim to align security practices with Agile principles while maintaining development agility.

4.3.2 Synthesis for RQ2: Roles, Checkpoints, and Process Adaptations

The findings indicate that achieving continuous security assurance in Scrum requires organisational and governance adaptations, in addition to technical and process-level changes. Across the literature, role-based mechanisms are commonly used to introduce or distribute security expertise within Agile teams. Roles such as Security Champions [47], Security Masters [69], and dedicated security teams [70] are described as ways to improve security awareness, support decision-making, and facilitate communication between developers and security experts.

Coordination and checkpoint mechanisms also play a key role in sustaining security practices across iterations. Regular security prioritisation meetings [71] and the integration of security considerations into sprint planning [72] are described as ways to maintain visibility and ensure that security requirements are continuously revisited. These mechanisms address previously identified prioritisation constraints

(see Section 4.2.2), where security often competes with functional requirements for limited development resources.

At a broader level, the literature shows that effective security integration in Scrum requires structural alignment between Agile development processes and security engineering practices. Agile methods emphasise iterative, value-driven development, whereas security engineering is typically risk-driven and organised around more structured activities. This creates a fundamental misalignment between the two processes [72]. To address this, prior work proposes embedding security activities directly within iterative development cycles rather than treating them as separate processes. For example, this can be achieved by integrating practices from established frameworks such as Microsoft SDL, OWASP SAMM, and the ISO Common Criteria into Agile workflows [72]. Empirical evidence further indicates that incorporating risk analysis within iterations supports earlier identification and mitigation of security issues, enabling a shift from reactive to continuous, iteration-level security management [69]. However, the feasibility of these process adaptations is constrained by organisational context, particularly in environments where resources are limited, and as a result, security mechanisms must be selectively integrated based on their impact on cost, agility, and effectiveness [53].

Overall, continuous security assurance in Scrum is supported by a combination of role-based expertise, recurring coordination mechanisms, and context-dependent organisational adaptations.

4.3.3 Cross-Cutting Insights

Across both research questions, several consistent patterns emerge. First, the lack of visibility of security work is a recurring challenge, and many proposed mechanisms, including backlogs, tagging systems, and meetings, aim to make security concerns more explicit within Agile processes. Second, security integration is not

achieved through a single mechanism but through a combination of artefacts, roles, and processes that operate together within Scrum environments. A key cross-study observation is that no single mechanism is sufficient to ensure effective security integration in Scrum. Artefact-level approaches improve visibility but do not guarantee prioritisation, role-based mechanisms introduce expertise but may not scale effectively, and automation supports verification but does not address organisational or cultural barriers. The literature consistently suggests that effective integration emerges from the combined use of multiple complementary mechanisms rather than isolated practices [12], [47], [76].

Third, despite the wide range of proposed mechanisms, empirical validation remains limited. Many approaches are evaluated through case studies, controlled environments, or expert-based assessments rather than long-term industrial deployment. This limits the ability to generalise the findings and assess the real-world effectiveness of proposed secure Scrum adaptations [20], [75], [76].

Lastly, rather than a strict trade-off, the literature suggests an ongoing tension between maintaining agility and achieving sufficient security assurance. While many approaches attempt to minimise disruption through light-weight adaptations, empirical findings indicate that additional effort, coordination, and cost are often unavoidable.

These findings highlight the need for approaches that (1) systematically integrate security into Scrum artefacts and events, (2) maintain visibility and traceability across iterations, and (3) reduce the overhead associated with security practices. This gap motivates the development of frameworks that use automation and intelligent assistance to support secure Scrum practices.

5 Proposed Framework for Enhancement of Secure Scrum with LLM

This chapter addresses RQ2 by proposing an LLM-supported framework to enhance the integration of security practices in Scrum.

5.1 What is Secure Scrum

Although many different Secure Scrum models exist, such as S-Scrum and others, one of the findings of the systematic literature review was that security practices which are lightweight and context-dependent tend to have a greater impact. Furthermore, there is a general lack of empirical evidence supporting the effectiveness of Secure Scrum frameworks. For this reason, only five security practices are considered for enhancement by the LLM in this thesis. It is also important to note that all of these are artefact-based enhancements rather than role-based. The reason for this is that extending or modifying roles is beyond the scope of this thesis.

In total, five security practices were selected for enhancement. As synthesised in the literature review (see Section 4.3.3), one of the primary reasons for this selection is that these artefact-based practices provide relatively strong empirical support while also remaining lightweight enough to be easily integrated into Scrum. The

selected practices include: security backlog practices (adapted into a unified backlog model), S-Tagging, Security Definition of Done (DoD), Security Acceptance Criteria (SAC) and Misuse and Abuser Stories.

5.1.1 Security Backlogs/Security Repositories

The concept of a dedicated Security Backlog, proposed by Azham et al. [23] and evaluated by Ghani et al. [22], was selected for this framework because it ensures that security tasks are visible and prioritised rather than overlooked. While these studies show that a Security Backlog improves transparency, they also highlight the challenge of integrating rigorous security risk management into short sprints. Earlier approaches relied on the manual identification of security tasks by a dedicated role. The framework proposed in this section addresses this limitation by automating the identification and entry of items.

Furthermore, instead of introducing a separate security backlog, some approaches integrate security-related items directly into the main Product Backlog [54], hence effectively creating a kind of 'virtual' security backlog within the same backlog structure.

As a result, unlike traditional security backlogs, the proposed LSS framework does not maintain a separate backlog. Instead, it embeds security directly within the Product Backlog through automated S-Tagging and repository linkage. Security-relevant PBIs are continuously identified and enhanced by the LLM component, hence improving visibility without introducing additional cost to process agility and resources.

This is implemented through the LLM-Driven Product Backlog Security Enrichment with Automated S-Tagging component, where each PBI is linked to security metadata, misuse and abuser stories, and SAC. As a result, security work remains

tightly coupled with its corresponding feature, preserving Scrum's backlog structure while improving traceability and actionability.

5.1.2 S-Tags

The involvement of S-Tags was selected for this framework based on the prior work of Mihelič et al. [53], who evaluate security tags as relatively cost-efficient in terms of development effort (man-hours) and implementation overhead, while preserving development agility, which is a critical factor for Scrum teams. The specific mechanism adopted is derived from the 'Secure Scrum' model by Pohl and Hof [14]. In this model, an S-Tag acts as a connector between security concerns and PBIs. To improve visibility, a visual indicator called an S-Mark is applied to any backlog item that has security implications [14]. The S-Tag links the marked item to a detailed description of the security issue, which may take the form of a user story, misuse story, or abuser story, and can also reference external knowledge bases [14].

In the baseline workflow described by Pohl and Hof, security-relevant PBIs are identified and marked by the team during backlog refinement [14]. This link is maintained throughout the lifecycle: when PBIs are broken down into tasks, the security context (S-Tag) remains connected [14]. However, this framework introduces a distinct innovation: while the original model relies on manual identification, the proposed framework automates the generation and assignment of S-Tags using an LLM component. This addresses the scalability challenges of manual tagging while retaining the visibility benefits. As described in the previous section, this mechanism also works with the security-related artefacts within the Product Backlog to increase the visibility of the security items.

5.1.3 Security Acceptance Criteria

In this proposed framework, the term SAC is used to denote specific, testable security conditions which are attached to individual user stories. These testable security conditions are generated by the LLM component of the framework. Although the exact label “SAC” is not standard in the literature, Agile security research recognises the practice of integrating security into story acceptance conditions as a recurring coping strategy. Villamizar et al. support a review-oriented approach in which user stories and their associated security specifications are examined against security properties and OWASP high-level security requirements, hence reinforcing the need for security expectations to be explicitly specified in a form that can be systematically reviewed [62]. More generally, Neugent conceptualises computer security AC as measurable or demonstrable features of required security functions that define how acceptability will be judged. This provides an early theoretical basis for treating security-related acceptance conditions as explicit evaluation criteria [61]. Furthermore, recent work on incorporating cybersecurity requirements into Agile development shows that user stories can include AC to represent security requirements, supporting their role as structured and testable conditions within Scrum artefacts [63].

5.1.4 Misuse and Abuser Stories

Misuse or Abuser Stories represent attack scenarios written from the perspective of a malicious actor, such as “As an attacker, I will attempt to brute-force the login page to gain access to the system" [57] [58]. These stories help teams identify security requirements by exploring how a system could be misused and then deriving appropriate countermeasures.

In this framework, misuse and abuser stories are explicitly adopted to operationalise threat modelling within the Scrum process. While the concept of expressing security requirements through negative scenarios is established in the literature, such

as Peeters' abuser stories [57] and Ardo et al.'s evil user stories [59], this framework formalises their role as a direct input for introducing LLM-driven threat modelling.

By systematically deriving these negative scenarios from functional stories, the framework supports the early identification of potential attacker goals. These identified threats are then transformed into specific, testable SAC that are attached to the original user story, hence ensuring that the corresponding mitigation is not only a theoretical exercise but a concrete, releasable requirement. Furthermore, these Abuser stories with additional threat details and advanced mitigation strategies are also created in the product backlog and attached via S-Tag to the functional story. This provides additional contextual details to the developer in addition to the derived SAC of the functional story.

5.1.5 Security Definition of Done

In this framework, the Security DoD is formally adopted as the global compliance baseline. While Kopczyńska et al. define the DoD as a collective quality measure [80], and Terpstra et al. identify it as a key artefact for security coping strategies [20], this framework leverages it to enforce continuous security assurance. Maier et al. similarly emphasise the necessity of such process-level security checks in Secure Scrum environments [60]. The DoD is then applied consistently to every increment. A story or PBI is not considered complete until all conditions in the DoD are met, at which point the increment is formally accepted.

The framework explicitly differentiates between SAC and DoD to avoid redundancy: "SAC" are used for feature-specific security requirements (e.g. this login must lock after 3 attempts), whereas the "Security DoD" ensures that every increment meets a non-negotiable security baseline (e.g. all code must pass static analysis and have no critical vulnerabilities). This differentiation addresses the common challenge

of 'security being intangible' [76] by making it a concrete, checkable gate for every sprint.

5.2 Proposed LLM-Supported Secure Scrum Framework

This section defines the working of the framework, its workflows, roles during the development workflow and design principles.

To provide a holistic view of the framework's design, Figure 5.1 presents a layered conceptual model illustrating how the LSS framework facilitates security integration within Agile development. The model captures the interaction between three layers: human actors (PO, Developer, Scrum Master), Scrum artefacts enhanced with security metadata, and the LLM assistance layer that drives automated analysis and artefact generation. This layered structure positions the framework not merely as a tooling solution, but as a structured socio-technical approach to embedding security into iterative development processes.

As illustrated in the model (Figure 5.1), the LLM layer does not operate autonomously but rather serves as an intermediary that enriches Scrum artefacts with security metadata, while human actors retain full control over prioritisation, implementation, and acceptance decisions. The bidirectional review link between the Developer and the Artefact Generation module reflects the human-in-the-loop design principle central to the framework. The resulting framework outcomes, namely improved security visibility, traceability, and risk awareness, emerge from this structured interaction between human expertise and automated assistance, rather than from either in isolation.

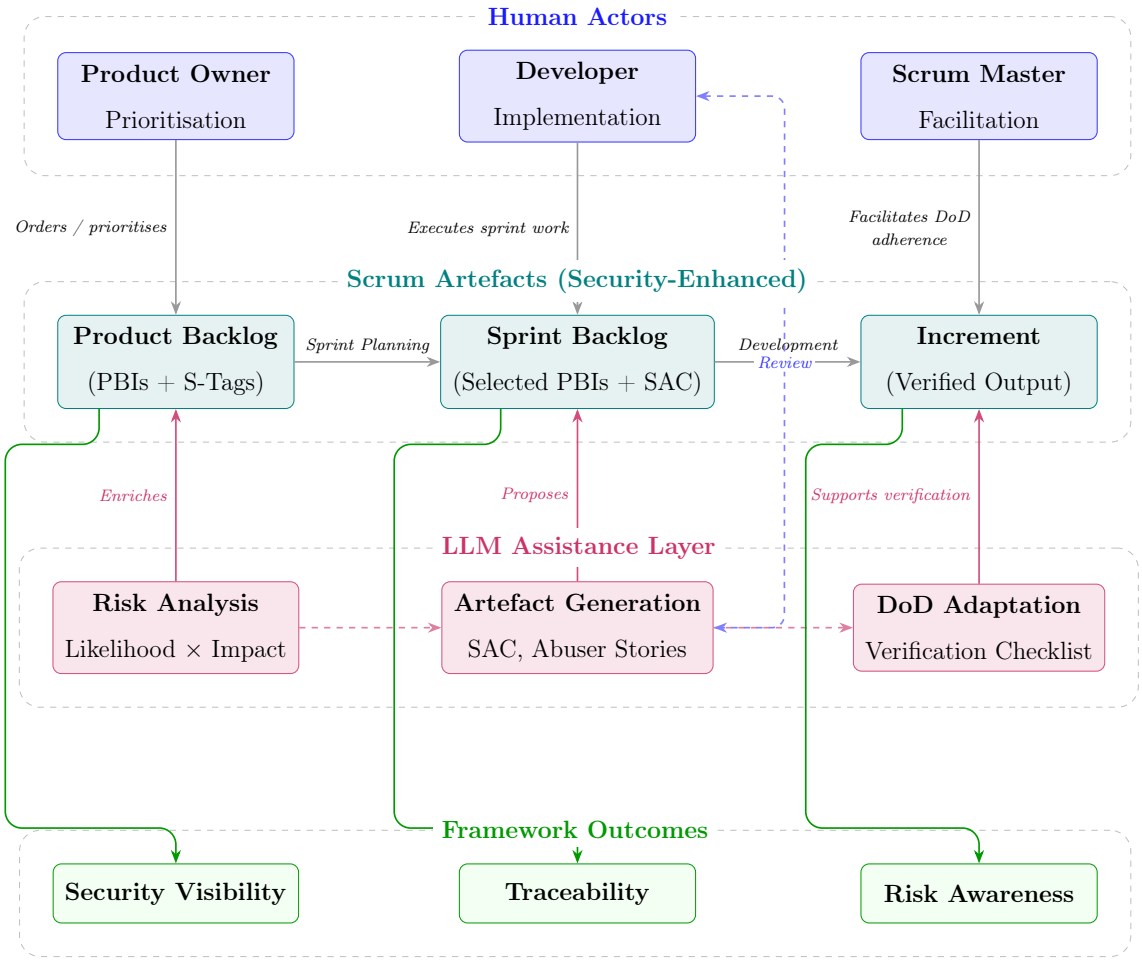

Figure 5.1: Scrum integration with LLM assistance

5.2.1 Design Motivation

When incorporating security principles into Scrum, there are numerous practices available, many of which were identified in the literature review in Chapter 4, and some more are added in this framework as well. These are different practices that introduce traceability and early discovery of the vulnerabilities. However, even if these practices succeed they are very time-consuming and may require additional personnel that many small teams cannot afford, hence forcing them to completely forgo the security aspect of the development.

This framework seeks to change that. It includes an LLM component responsible for the analysis, generation, and validation of security-related artefacts. The framework automates the discovery, risk assessment, and linkage of security work while maintaining Scrum's single ordered backlog and DoD. Although multiple artefacts are generated, the cognitive and operational overhead for the development team is minimised through automation, as the LLM component handles the analysis, generation, and linkage of security artefacts without requiring additional manual effort.

Operationally, these selected practices are implemented through five framework components. Four components directly correspond to the selected security practices, while the fifth, the unified dependency-driven prioritisation model, supports their integration into Scrum's single ordered Product Backlog.

The LSS framework embeds five key components into the Scrum lifecycle:

1. **LLM-Driven Product Backlog Security Enrichment with Automated S-Tagging**: automatically identifies, labels, and risk-assesses security-relevant PBIs within the unified Product Backlog.

2. **LLM-Supported SAC Generation**: produces security-specific AC for identified backlog items to support testable verification and validation.

3. **Misuse and Abuser Story Generation**: derives Misuse and Abuser Stories to capture threats and their mitigations within Scrum artefacts.

4. **Automated Security DoD Adaptation**: defines a baseline Security DoD and dynamically extends it with sprint-specific conditions that must be validated when an increment is marked as done.

5. **Unified, Dependency-Driven Prioritisation Model**: supports prioritisation of security and functional tasks based on risk value to enable balanced sprint planning.

5.2.2 Conceptual Product Backlog Model in LSS

This section presents the conceptual structure of the Product Backlog under the LSS framework, as illustrated in Figure 5.2. The diagram demonstrates how security artefacts are embedded directly within the Product Backlog rather than managed in a separate security backlog.

At the core of the model are Product Backlog Items (PBIs), each of which may contain functional Acceptance Criteria (AC) and, where applicable, LLM-generated Security Acceptance Criteria (SAC). The framework appends SAC directly to the original PBI, ensuring that security requirements remain closely linked to functional requirements and are validated together.

Security relevance is represented through S-Tags, which act as linking mechanisms between PBIs and associated security artefacts. These artefacts include threat-oriented artefacts, such as misuse and abuser stories, as well as derived mitigation actions that may be translated into Security Tasks (ST). The diagram shows that multiple PBIs can be connected to one or more security tasks through shared S-Tags. This forms a traceable relationship between features, threats, and mitigations.

Furthermore, the conceptual partition shown in the diagram separates functional

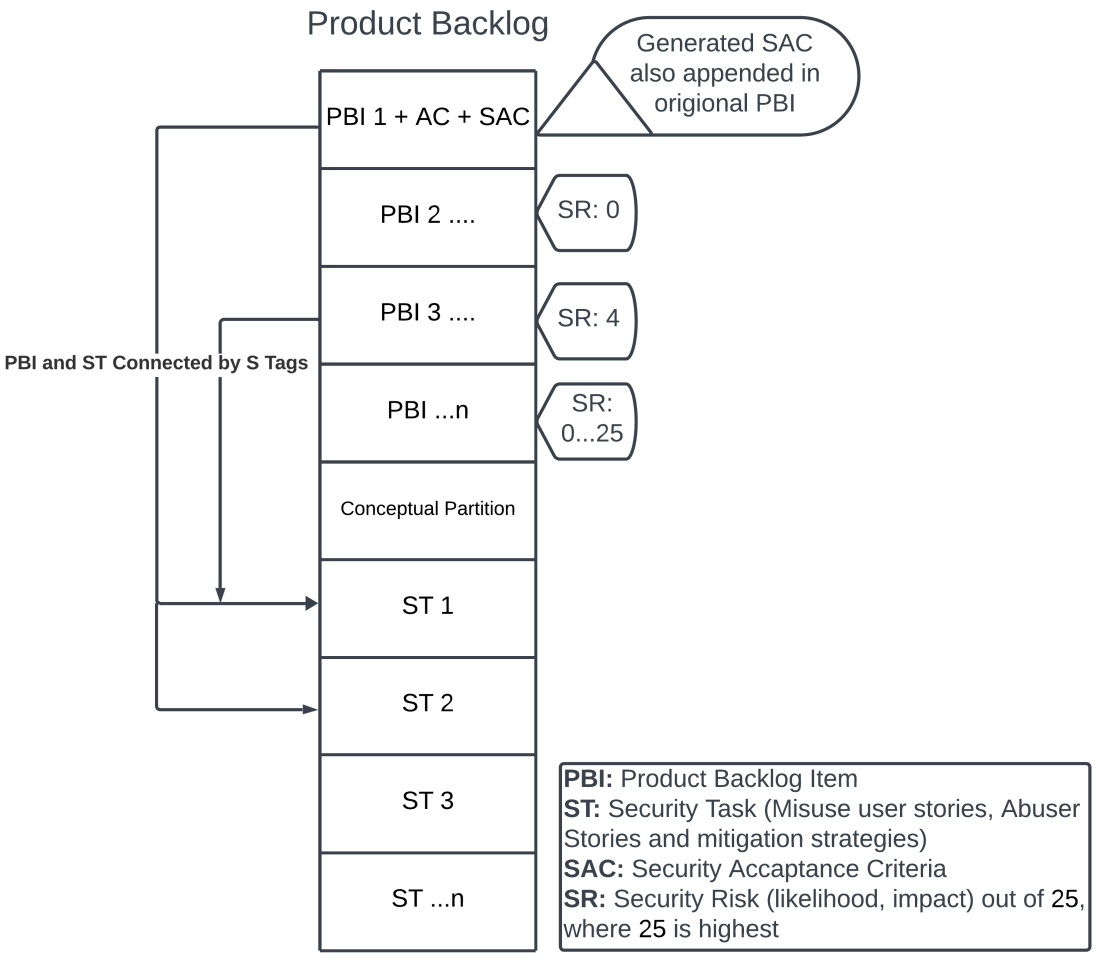

Figure 5.2: Conceptual product backlog structure in the LSS framework

PBIs from derived security tasks while maintaining a logical link through S-Tags. This ensures that security work remains visible, traceable, and aligned with feature development, without introducing additional backlog structures.

Additionally, each PBI is assigned structured security risk (SR) metadata by the LLM component. This metadata includes likelihood (1-5) and impact (1-5) scores, which are combined using a multiplicative model (Likelihood × Impact) to calculate an overall risk level that is then categorised as Low, Medium, High, or Critical. This risk-related information supports prioritisation decisions while maintaining Scrum's principle of a single ordered backlog.

5.2.3 Security Risk Estimation Model

In the LSS framework, security risk is estimated using a simplified likelihood-impact model, which is widely used in software security risk assessment. For each security-relevant PBI, the LLM component assigns two quantitative measures:

- **Likelihood (L):** the probability that a vulnerability can be exploited (scale 1 - 5)
- **Impact (I):** the severity of consequences if the vulnerability is exploited (scale 1 - 5)

The overall risk score is computed using a multiplicative model:

$$Risk = L \times I \tag{5.1}$$

The resulting score (range 1 - 25) is then mapped into qualitative categories to support decision-making:

- **Low:** 1 - 5
- **Medium:** 6 - 12

- **High:** 13 - 20
- **Critical:** 21 - 25

This formulation is consistent with established principles in information security risk assessment, where risk is understood as a function of likelihood and impact. Frameworks such as ISO/IEC 27005 and NIST SP 800-30 define risk as a combination of the probability of occurrence and the severity of its consequences. Similarly, the OWASP Risk Rating Methodology[1] follows a structured approach that assesses likelihood and impact through multiple contributing factors, reflecting the same underlying conceptual model [81], [82], [83].

However, while these frameworks include multiple detailed factors (e.g. threat agent capability, vulnerability exploitability, and asset value), the LSS framework uses a simplified model. This design choice is intentional and reflects the constraints of Agile development environments, where security practices need to remain lightweight, easy to understand, and compatible with short iteration cycles. This is consistent with findings from the literature review, which emphasise that lightweight and context-adaptable security practices are more suitable for Agile and Scrum environments (see Section 4.3.3).

Furthermore, although risk estimation is discussed in several studies as part of secure Agile practices [56], [67], its operationalisation within Scrum artefacts remains limited and often lacks standardisation. In particular, Ionita et al. propose a risk-driven approach in which risks are quantified based on likelihood and impact and then used to derive and prioritise security requirements [67]. Similarly, Voggenreiter and Schöpp introduce a scoring-based mechanism for prioritising security findings in Agile contexts [66]. These approaches highlight the importance of structured risk assessment as a basis for decision-making.

[1]OWASP Risk Rating Methodology; see `https://owasp.org/www-community/OWASP_Risk_Rating_Methodology`

Building on these insights, the LSS framework integrates risk estimation directly into PBIs and uses it as a supporting input for prioritisation. While Scrum typically prioritises PBIs based on business value, incorporating risk information allows for more informed trade-offs between functional requirements and security concerns. This is particularly important given the recurring challenge identified in the literature that security is often deprioritised due to its intangible nature and lack of explicit representation (see Section 4.2.2).

Therefore, the proposed model does not replace Scrum prioritisation mechanisms but extends them by providing structured, interpretable, and automatically generated risk metadata. This supports development teams in making more balanced decisions while preserving Scrum's principle of a single ordered backlog.

Overall, the model puts the integration of security into Scrum into practice by embedding security artefacts, risk information, and traceability mechanisms directly into the Product Backlog.

5.2.4 Conceptual Components of the LSS Framework

This section defines conceptual components that constitute the LSS Framework. It must be noted that these are not physical components of the application but rather ideological divisions of the blueprint of the framework.

To illustrate how these components are orchestrated, Figure 5.3 presents the LLM Interaction Model. This diagram highlights the core artefact-generation part of the framework, showing how raw inputs are transformed into S-Tags, security risk (SR) assessments, misuse stories, and SAC through the LLM processing engine, while explicitly maintaining a human-in-the-loop validation step.

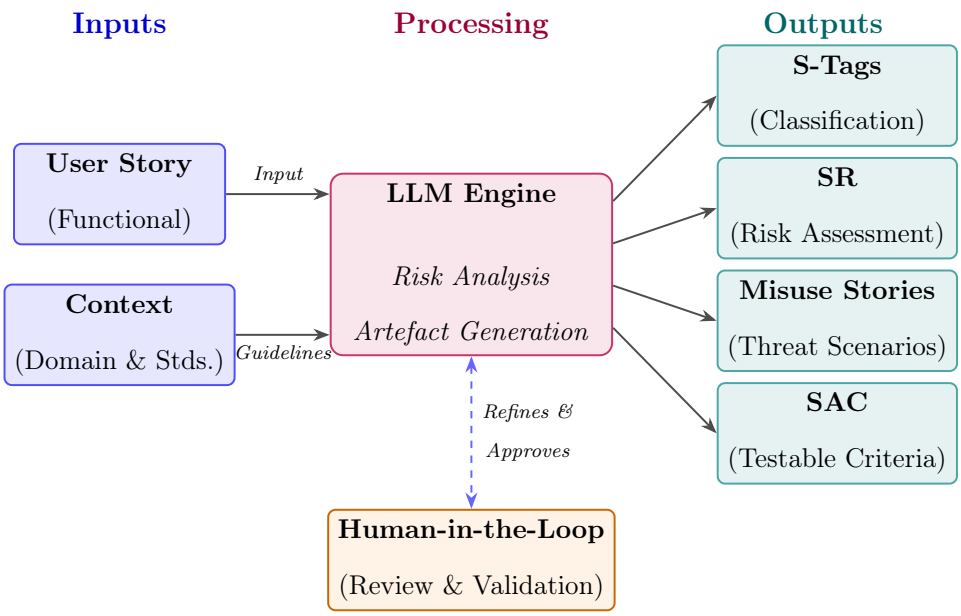

Figure 5.3: LLM interaction model in Scrum

5.2.4.1 LLM-Driven Product Backlog Security Enrichment with Automated S-Tagging

The purpose of this component is to continuously identify and structure the security-related work without additional manual overhead.

Mechanism:

- **Security Relevance Detection**: When a PBI is created or updated, the LLM component classifies its security relevance and assigns one or more S-Tags from a predefined taxonomy (such as Authentication, Input Validation, and Data Exposure). These tags map to Open Worldwide Application Security Project (OWASP) Top-10 [2] and Common Weakness Enumeration (CWE) IDs [3] to support standardisation.

[2]The OWASP Top 10 is a widely recognised list of the ten most critical web application security risks, published by the Open Web Application Security Project (OWASP). `https://owasp.org/Top10/`

[3]The Common Weakness Enumeration (CWE) is a community-developed list of software and hardware weakness types that provides a common language for describing vulnerabilities, enabling standardisation and auditability. `https://cwe.mitre.org/`

- **Contextual Enrichment**: After identifying a security-relevant backlog item, the LLM component performs contextual enrichment. This is done by attaching structured risk information and references to security standards to the item. The enrichment includes estimated likelihood and impact scores, a computed risk rating, and a natural-language explanation of why the story has security implications. In addition, each S-Tag is cross-referenced with OWASP Application Security Verification Standard (ASVS) control families (such as V2 Authentication and V5 Input Validation), providing traceability to widely recognised verification standards. This allows the PO and development team not only to prioritise work based on quantitative risk but also to ensure that each security-sensitive story aligns with established security controls.

  Furthermore, the framework also considers the overall project context. This is incorporated during the initial phases of project development. The domain of the project is fed into the framework when the project is initialised, where the PO defines the project context, which is stored in a database and subsequently provided to the LLM through structured prompting, enabling the model to consider this domain from a security perspective when making decisions. For example, if the domain of the project is medical related, then the framework will consider that context, incorporate domain-specific security considerations, and identify assets that need to be protected, ensuring that the generated security artefacts within Scrum take this domain into account.

  Finally, there is also flexibility to make the framework more customisable. This is done at the end of the sprint, usually in the sprint retrospective. If there is a certain way that the team needs the LLM component to act, then only the main points can be added to the framework system. It is also important to note here that only main points relevant to the workings of the framework

should be added, as priority is given to these points, as they reflect the current needs of the development team.

- **Repository Linkage**: Each S-Tag links the PBI to a logically grouped security entry within the unified Product Backlog, containing a concise description such as an abuser story and mitigation patterns. These entries do not constitute a separate backlog but represent structured artefacts embedded within the same backlog.

The overall goal of this component is to automatically highlight the security concerns of each backlog item and to establish the level of traceability for security concerns.

5.2.4.2 LLM-Supported SAC Generation

This component ensures that the security expectations of tagged PBIs are explicit and testable at the story level, aligning with AC norms.

Mechanism:

- **SAC proposal**: Given a PBI and its S-Tags, the LLM proposes SAC in a concise and testable form, grounded in ASVS and mapped to OWASP Top-10 risks for that domain.
- **Consistency Check**: Across similar stories, the framework ensures consistency in SAC.
- **Traceability**: The SAC is updated in the original backlog item, which is then linked through the S-Tag to misuse and abuser stories for verification.

Example: "As a user, I upload a profile picture" $\rightarrow$ SAC: only jpg/png; ≤ 5 MB; AV scan; metadata stripped; deny executable content (mapped to CWE/ASVS controls).

5.2.4.3 Misuse and Abuser Stories Generation

This component ensures that threats and derived mitigations are systematically highlighted at the level of user stories, following the misuse case approach and linking them into the backlog flow.

Mechanism:

- **Attacker-centric expansion**: For each functional PBI, the LLM component generates misuse and abuser stories that represent hostile intent, structured using the Actor-Action-Impact format.
- **Mitigation derivation**: For each misuse, the framework also generates a mitigation strategy (defender's view) (e.g. implement rate limiting, JWT expiry/rotation, server-side validation).
- **Risk ranking**: Misuse and Abuser Stories inherit risk from S-Tags and OWASP categories (A01-A10). This risk metadata serves as a decision-support signal to help the team address high-impact threats first, without overriding PO prioritisation of the parent PBI.
- **Link graph**: Misuse $\Leftrightarrow$ mitigation linkages are stored in the product backlog and linked back to PBIs via the S-Tag (per Secure Scrum's linkage model).

As a result, teams get early, story-level threat modelling and a direct path from threat to acceptance testing without separate heavy ceremonies.

5.2.4.4 Automated Security DoD

The framework defines a baseline Security DoD, which is then dynamically extended by the LLM based on sprint-specific security context. This ensures that each increment satisfies relevant security quality measures, consistent with Scrum's DoD commitment.

Mechanism:

- **Security DoD baseline**: Define a global Security DoD that includes tasks such as running static analysis, ensuring dependency scans are clean, performing secrets scans, and updating the threat model where relevant.
- **Adaptive checks**: For each sprint, the LLM tailors the DoD focus by analysing the sprint's S-Tag mix. For example, if multiple items relate to Authentication, it emphasises authentication testing and session handling.
- **Continuous Integration/Continuous Deployment (CI/CD) integration (optional, if CI/CD pipelines are available)**: Parse SAST[4]/DAST[5]/dependency reports and test logs. The LLM component does not independently determine whether a story is done. Instead, it summarises evidence from CI/CD outputs, test results, and security scans, and flags potential unmet Security DoD conditions for human review.
- **Audit trail**: Store the LLM's rationale and tool evidence alongside the PBI increment. This becomes useful for compliance and retrospectives.

As a result, DoD becomes an evolving, auditable gate: the moment a PBI meets functional AC + SAC + Security DoD, an Increment is accepted.

5.2.4.5 Dependency-Aware Security Prioritisation

Maintain Scrum's single ordered backlog while also ensuring security work is sprint-synchronous with its parent feature.

Mechanism:

[4]Static Application Security Testing (SAST) involves analysing source code or binaries to identify security vulnerabilities without executing the program; see `https://owasp.org/www-community/Source_Code_Analysis_Tools`

[5]Dynamic Application Security Testing (DAST) evaluates a running application from the outside to identify security vulnerabilities by simulating attacks; see `https://owasp.org/www-community/Vulnerability_Scanning_Tools`

- **PBI-centric prioritisation**: Only the Product Backlog Items (PBIs) are prioritised manually by the PO, hence preserving standard Scrum backlog ownership and decision flow.

- **Automatic attachment of security artefacts**: Security artefacts (misuse and abuser stories, SAC) are excluded from independent prioritisation and are automatically attached when their parent PBI is selected, hence ensuring that security is handled within the same development context.

- **Dependency graph management**: The framework maintains a dependency graph ($\text{PBI}_a \Leftrightarrow \text{PBI}_b$, $\text{S-Tag}_a \Leftrightarrow \text{S-Tag}_b$, security artefacts) and classifies security tasks as either dedicated (bound to a specific PBI) or shared between multiple PBIs. Dedicated tasks move together with their PBIs, while shared tasks can be scheduled independently when they support multiple near-term PBIs.

- **Integrated risk assessment**: For each selected PBI, the framework performs an automated risk assessment aligned with OWASP, producing structured outputs such as likelihood, impact, OWASP Top 10 categories, CWE mappings, and ASVS controls.

5.2.5 End-to-End Sprint Workflow

In this section, the proposed workflow describes how security is continuously integrated and managed throughout the Scrum lifecycle. Figure 5.4 illustrates the end-to-end security workflow within the LSS framework. The process begins when a PBI or user story is created, triggering an automated LLM analysis that classifies the item for security relevance. Based on this analysis, S-Tags are assigned to link the item to relevant security concerns and standards. The framework then automatically enriches the backlog item by generating a security risk score, misuse

and abuser stories to capture potential threats, and testable SAC. During Sprint Execution, functional development and security tasks (including SAC verification) are carried out in parallel. Before a sprint increment is accepted, the Security Definition of Done (DoD) gate is enforced to ensure that all security conditions have been met. Finally, a retrospective feedback loop enables continuous refinement of the framework by adapting constraints and prompts based on team observations and evolving project needs.

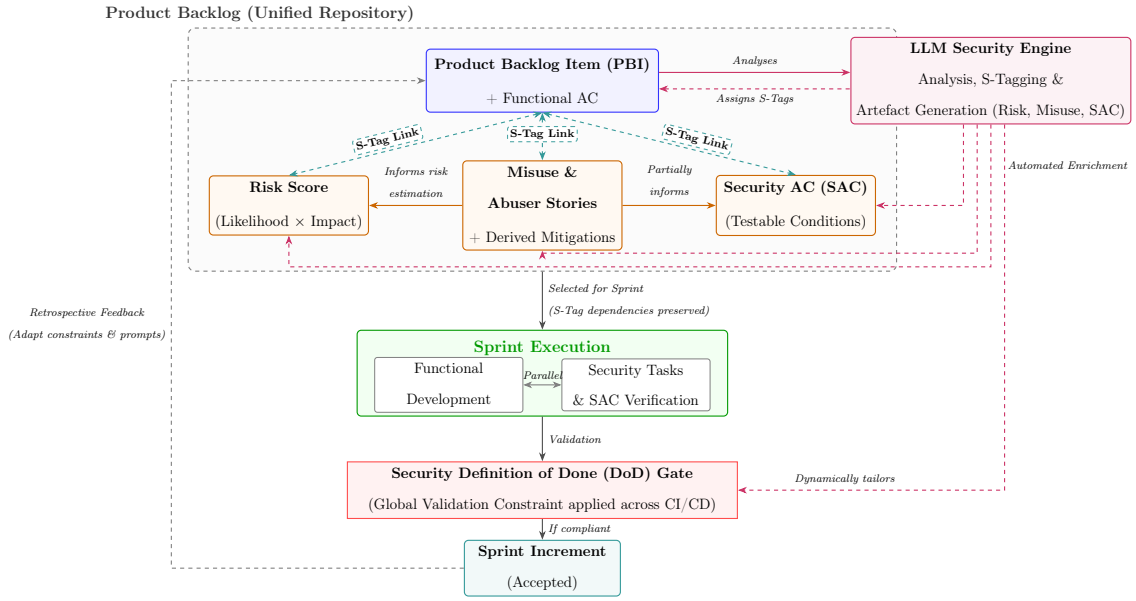

Figure 5.4: Conceptual end-to-end security workflow in the LSS framework

As detailed above, Figure 5.4 presents the operational workflow of the LSS framework, complementing the structural Product Backlog model shown earlier in Figure 5.2. While Figure 5.2 illustrates how security artefacts are structurally embedded and linked within the backlog, Figure 5.4 focuses on how these artefacts are generated, propagated, and validated throughout the Scrum lifecycle. It is important to note that although the diagram depicts the generation of S-Tags, risk scores, misuse stories, and SAC sequentially for visual clarity, these artefacts are actually generated

by the LLM Security Engine through an interdependent analysis rather than a strict linear pipeline.

5.2.5.1 Backlog Authoring (Continuous)

The development team creates or updates Product Backlog Items (PBIs). The framework uses an LLM with structured prompting to automatically assign S-Tags, map them to relevant CWE and OWASP categories, and add risk-related metadata. It also generates corresponding misuse and abuser stories, along with SAC. All related security artefacts are linked to their corresponding PBIs within the Product Backlog through S-Tag associations.

One security concern is that user stories may contain sensitive information that should not be shared with the LLM component. In such cases, it is the responsibility of the story author to omit these details, censor them, or exclude the story from being processed by the framework's LLM.

5.2.5.2 Backlog Refinement (Continuous or Regularly Scheduled)

Backlog refinement occurs continuously or during regular refinement sessions, depending on the team's Scrum practice. During refinement sessions, the team reviews the LLM-generated suggestions and refines the SAC as needed. The team can also split larger tasks into smaller and more manageable items. All related security artefacts remain linked across the backlog, and duplicate items are merged into shared security artefacts to maintain consistency.

5.2.5.3 Prioritisation (Ongoing)

The PO orders PBIs in the backlog based on business value, and LLM-generated risk information is used only as supporting input rather than as a strict control mechanism.

5.2.5.4 Sprint Planning

When a PBI is selected for a sprint, its associated misuse and abuser stories, and SAC are automatically proposed for inclusion in the Sprint Backlog to ensure joint implementation of functional and security tasks.

5.2.5.5 Development and Verification

During the Sprint, the developers implement both the feature and its associated security tasks in parallel. Automated tests and validation processes confirm that all SAC are satisfied before the story is marked as complete.

5.2.5.6 DoD (CI/CD Integrated)

The LLM component reviews continuous integration and testing outputs, summarising the results of scans and automated tests. Any unmet items from the Security DoD prevent the story from being marked as "done."

5.2.5.7 Sprint Review and Retrospective

At the end of each sprint, the LLM component generates a Sprint Security Report that summarises the implemented controls, residual risks, and recurring S-Tags. This information supports continuous improvement and backlog refinement. It also helps identify long-term security trends within the project.

5.2.6 Roles and Responsibilities

This section defines the Roles and their Responsibilities in the modified proposed Scrum Framework.

5.2.6.1 Product Owner

The PO prioritises Product Backlog Items (PBIs) based on business value, while also taking into account the risk metadata generated by the LLM component. The PO focuses on ordering PBIs rather than individual security tasks, and reviews completed PBIs against agreed functional AC and SAC, while the Developers remain responsible for ensuring that the Increment satisfies the DoD.

5.2.6.2 Developers

Developers implement both functional features and their related security tasks within the same sprint. They also provide domain-specific feedback to refine SAC definitions and improve reusable mitigation templates, hence supporting the continuous improvement of security artefacts.

5.2.6.3 Security Champion (Optional Organisational Support)

The Security Champion is an optional organisational support function that does not modify or replace any existing Scrum roles. This function ensures consistency in S-Tags, validates SAC for high-risk items, and oversees the logical grouping of security artefacts within the unified Product Backlog. It also helps keep security mappings to frameworks such as OWASP, CWE, and ASVS up to date and aligned with industry standards.

5.2.6.4 Scrum Master

The Scrum Master ensures that the LLM-supported workflow adheres to Scrum principles, events, and artefacts. They also support continuous improvement by facilitating the refinement of LLM prompts and validation processes within sprint activities.

5.2.6.5 LLM Security Engine (LLM component of the framework)

The LLM acts as a supporting analysis component by automatically detecting, linking, enriching, and verifying security-related items throughout the development lifecycle. This is done without changing the fundamental roles or responsibilities defined in Scrum.

5.2.7 How LSS Differs from the Traditional Security Backlog

Traditional security backlogs focus on collecting and managing security-related tasks derived from activities such as threat modelling, penetration testing, vulnerability scanning, incident reporting, and audit findings. These tasks are then manually refined, prioritised, and scheduled alongside functional Product Backlog Items (PBIs). While this approach highlights security as an important activity within Scrum, it largely remains manual, reactive, and dependent on periodic threat assessments. As a result, security issues are often identified only after development or during later testing phases, and the traceability between functional and security items is often limited [22].

In contrast, the LSS framework automates this process, making it more proactive and continuous. Instead of relying on post-development identification of issues, LSS uses Large Language Models (LLMs) to:

- Continuously analyse PBIs for security relevance and automatically assign S-Tags.
- Generate linked misuse and abuser stories to ensure that threats and corresponding countermeasures are identified early in the development process.
- Produce Security Acceptance Criteria (SAC) for each S-Tagged item and attach them directly to the corresponding user story.

- Enforce validation of the Security Definition of Done (DoD) through CI/CD integration, where the LLM-supported conceptual Product Backlog model is used to verify testing and security scanning results.
- Maintain unified prioritisation by ordering only PBIs, while automatically proposing related security artefacts for inclusion when a parent PBI enters a sprint.

This proactive mechanism transforms the security backlog from a static collection of post-discovery tasks into a dynamic, LLM-driven system that supports continuous risk detection, linkage, and validation. It preserves Scrum's principle of a single ordered backlog while significantly improving security visibility, traceability, and assurance.

6 Experimental Setup

The experimentation phase involves a systematic validation of the LSS framework using a dataset of user stories, with a fixed set of inputs and a consistent processing workflow. The process follows a linear pipeline: Input (Raw User Stories) → Framework Processing (LLM Component) → Output (Enhanced Security Artefacts). Each raw user story is processed to generate SAC, Abuser Stories, Security Risk Metadata (e.g. likelihood and impact scores, risk ratings, and rationale), and appropriate S-Tags, with the aim of transforming a functional requirement into a security-aware backlog item.

6.1 Selection of the Evaluation Dataset

To ensure a realistic evaluation, a dataset of 50 user stories was selected from the "Camp Administrator" project[1]. This open-source dataset represents a typical management system that includes diverse functional requirements, ranging from administrative tasks to the handling of sensitive data.

Example stories include:

- "As a camp administrator, I want to be able to add campers, so that I can keep track of each individual camper."

[1]G10 dataset available at: `https://zenodo.org/records/13880060`

- "As a parent, I want to be able to submit various types of forms to the camp management."

This diversity in user stories allows for testing the framework's ability to distinguish between benign features (e.g. viewing schedules) and high-risk operations (e.g. handling medical forms or payments).

6.2 Prototype Architecture and Technical Implementation

The technical core of the LSS framework is based on a specialised prompt engineering strategy that leverages the reasoning capabilities of LLMs. The implementation uses the OpenAI API (GPT-4o) integrated into a custom TypeScript application. The strategy applies three key techniques: **Role-Based Prompting** to establish a "Security Expert" persona, **Structured Output Generation** (JSON) to ensure machine readability, and **Contextual Constraint** by providing specific security taxonomies, such as OWASP Top 10 (2021) and OWASP ASVS v4.0 (V1–V14), as reference context within the prompts.

Since the framework relies on prompt-driven generation, the following sections detail the specific prompts used for each analysis component which are extracted from the tool's implementation. Figure 6.1 illustrates the system architecture of the LSS tool, showing how raw user stories are processed through the LLM component and its five specialised processing modules to produce security-enhanced backlog items.

The architecture shown in Figure 6.1 illustrates the processing pipeline of the LSS tool, where a raw user story is gradually enriched through a sequence of LLM-driven analysis stages. Although the components are presented as modular units, they are executed sequentially within the tool, with each stage building on the

outputs of the previous one. Specifically, the framework begins by establishing S-Tag classification which then informs risk baseline, and threat modelling through Abuser Stories. These artefacts are subsequently used to derive Security Acceptance Criteria (SAC) and, finally, a Security Definition of Done (DoD) checklist.

Each processing module corresponds to a specialised prompt template executed by the LLM component, rather than a separate computational service. As a result, the LLM is invoked multiple times in a controlled manner, with structured prompts ensuring consistency and alignment with security standards. Providing this knowledge as reference context (OWASP Top 10, ASVS, and CWE) ensures that all generated artefacts are grounded in recognised security practices. The final output is an enhanced Product Backlog Item (PBI) that combines functional requirements with structured and testable security artefacts.

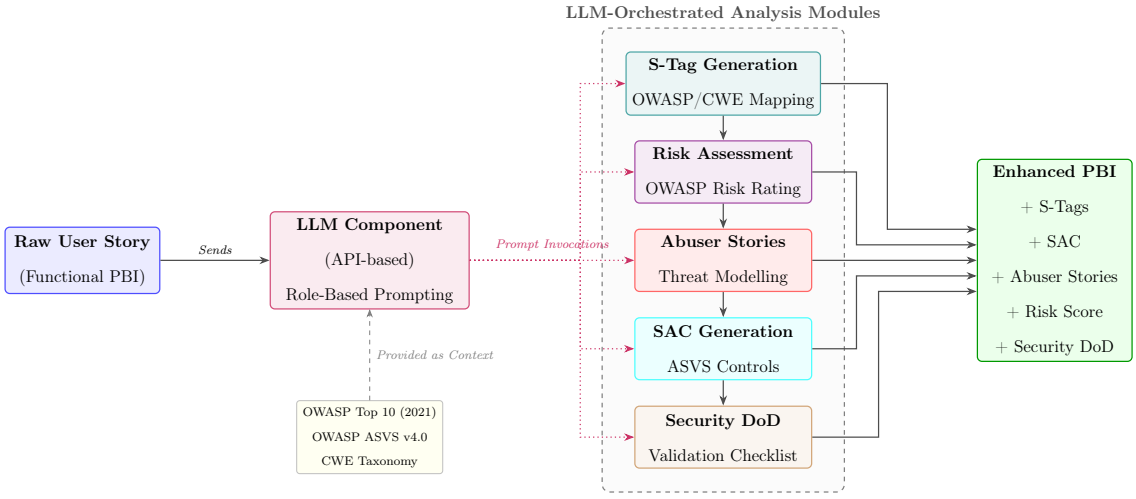

Figure 6.1: System architecture of the LSS tool

6.2.1 Risk Assessment Prompt

To establish an initial risk baseline, the tool sends the user story to the LLM with instructions to calculate Likelihood and Impact scores based on the OWASP Risk Rating Methodology. To ensure accurate and standardised categorisation, the defini-

tions for the OWASP Top 10 categories, CWE IDs, and ASVS controls are provided to the LLM as reference context within the system prompt.

```
"You are a security expert specialising in risk assessment for software development following the
    OWASP framework.
Analyse the following user story and provide a comprehensive risk assessment.
Issue Details: Title: ${title}, Description: ${description}

Perform a risk assessment with the following components:
1. Likelihood Score (1-5)
2. Impact Score (1-5)
3. OWASP Top 10 (2021) Categories
4. CWE IDs
5. OWASP ASVS v4.0 (V1-V14) Controls
6. Rationale

Output ONLY valid JSON in this exact format: { 'likelihood': 3, 'impact': 4, ... }"
```

Listing 6.1: Risk Assessment Prompt

6.2.2 S-Tag Generation Prompt

For S-Tag classification, the prompt provides a strict taxonomy of "S-Tags" and requires the LLM to map the story's functionality to these categories. As with the risk assessment, the LLM references the provided OWASP and CWE context to ensure standardised mapping.

```
"You are a security expert specialising in the LSS (LSS) framework.
Analyse the following user story and generate S-Tags for classification.
S-Tag Taxonomy: Authentication & Authorization, Input Validation, Data Protection...

Your Task:
1. Determine if this issue has security relevance
2. Assign appropriate S-Tags from the taxonomy
3. Map each S-Tag to relevant OWASP Top 10 (2021) categories and CWE IDs
4. Assign severity level (Critical/High/Medium/Low)

Output ONLY valid JSON..."
```

Listing 6.2: S-Tag Generation Prompt

6.2.3 Abuser Story Generation Prompt

To support threat modelling, the prompt adopts an adversarial persona. It explicitly constrains the model to focus on realistic, high-impact threats and instructs the LLM to identify potential attackers and their goals using a structured "Actor-Action-Impact-Mitigation" format.

```
"You are a security expert specialising in threat modelling...
Generate 2-4 Abuser Stories that follow the Actor-Action-Impact structure:
- Actor: Who is the malicious actor? (e.g. 'SQL Injection Attacker')
- Action: What specific attack or misuse do they perform?
- Impact: What is the potential damage?
- Mitigation: What specific security control prevents this?

Focus on realistic, high-impact threats. Output ONLY valid JSON..."
```

Listing 6.3: Abuser Story Generation Prompt

6.2.4 SAC Generation Prompt

Finally, to operationalise the security requirements, the prompt adopts the role of a Quality Assurance expert, generating testable criteria mapped to OWASP ASVS v4.0 (V1–V14). The LLM leverages the provided ASVS definitions to ensure these criteria align precisely with recognised security controls.

```
"You are a security expert specialising in the LSS framework and OWASP ASVS.
Generate SAC for the following user story...
1. Generate specific, testable SAC
2. Map each SAC to relevant OWASP ASVS v4.0 (V1-V14) controls
3. Mark if each criterion is testable
4. Prioritise using MoSCoW method

Output ONLY valid JSON..."
```

Listing 6.4: SAC Generation Prompt

6.2.5 Security Definition of Done (DoD) Generation Prompt

To ensure that security requirements are verified before completion of a user story, the prompt generates a structured Security DoD checklist. It combines baseline security practices with S-Tag-specific controls and ensures alignment with identified risks.

```
"You are a security expert working within the LSS (LSS) framework.

Given:

A user story
Assigned S-Tags
Relevant security context

Your Task:

Generate a Security DoD checklist
Include baseline security checks (e.g. SAST, dependency scan, secrets detection)
Add S-Tag-specific security verification items
Ensure each item is clear, testable, and actionable

Focus on practical verification steps that must be completed before marking the story as done.

Output ONLY valid JSON as a checklist..."
```

Listing 6.5: Security DoD Generation Prompt

6.3 Execution of the Experimental Procedure

The experiment was conducted by processing the full dataset of 50 "Camp Administrator" user stories using the LSS tool. The workflow was designed to reflect a real-world Scrum refinement session, in which raw functional requirements are transformed into "Ready" backlog items with complete security context.

6.3.1 Tool Overview

The validation utilised a custom-developed tool (Figure 6.2, see also Appendix B) that acts as a bridge between the Scrum Product Backlog and the LLM component of the framework. The tool provides a dashboard for tracking the security status of all stories, visualising relationships, and managing risk.

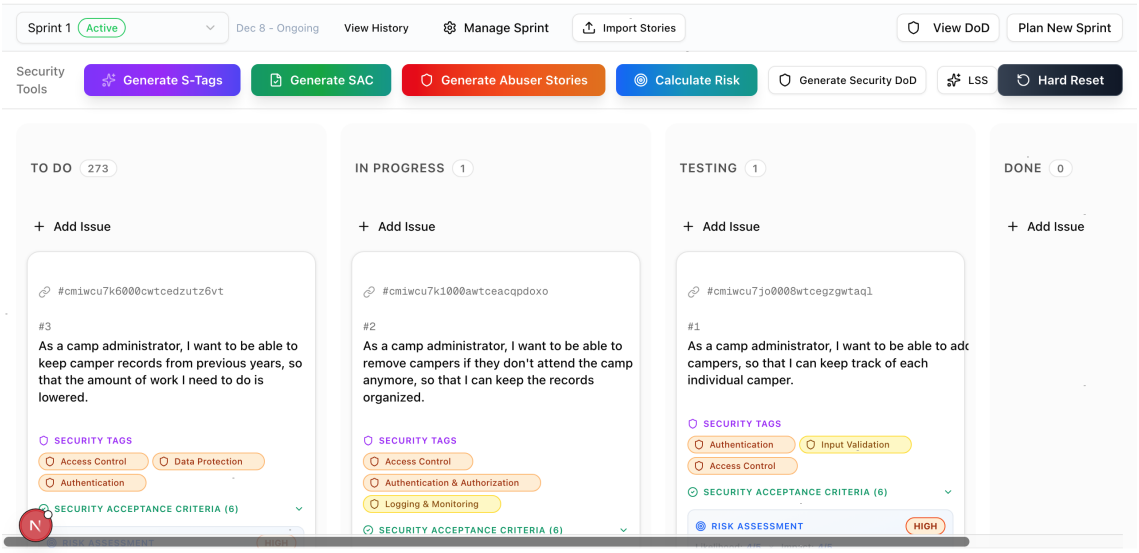

Figure 6.2: LSS tool overview dashboard showing the full backlog and analysis status

6.3.2 Detailed Walkthrough: "Add Camper" Use Case

To illustrate the detailed operation of the framework, we analyse the processing of the following user story: *"As a camp administrator, I want to be able to add campers, so that I can keep track of each individual camper."* Figure 6.3 provides an overview of how a single user story is progressively enriched through each processing stage, with the specific artefacts generated at each step annotated alongside.

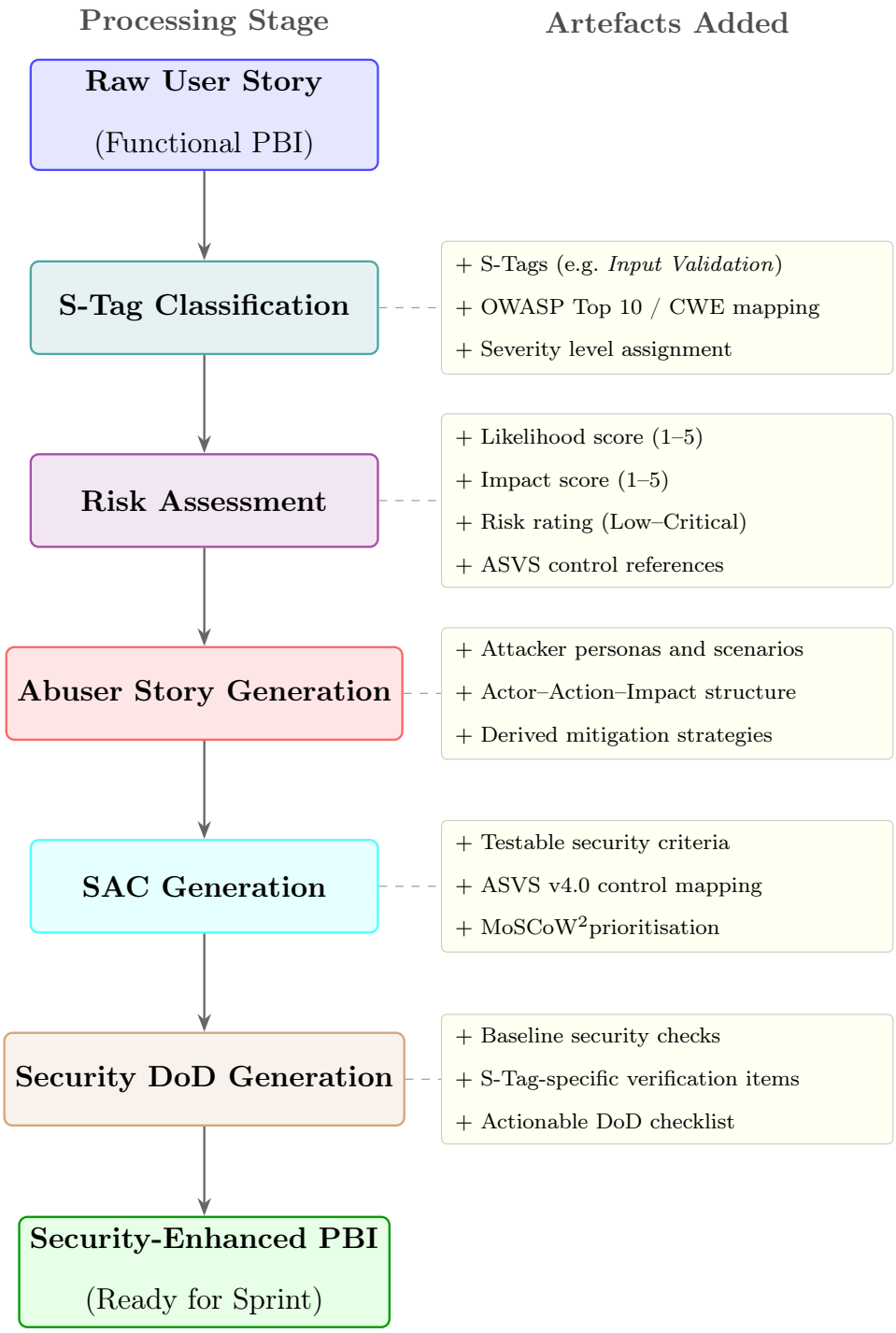

Figure 6.3: Data flow: step-by-step transformation of a user story

[2]The MoSCoW method (Must have, Should have, Could have, Won't have) is a popular prioritisation technique used in Agile software development to reach a common understanding with stakeholders on the importance of each requirement. See `https://en.wikipedia.org/wiki/MoSCoW_method`

The following walkthrough demonstrates the practical application of this transformation process within the LSS Tool. By following the 'Add Camper' use case, we illustrate how the system's internal logic, which is guided by the stages shown in Figure 6.3, manifests in the user interface to move a story from a raw state to a security-hardened and sprint-ready requirement.

Step 1: Input Analysis

System Processing Logic: At this stage, the raw user story is purely functional and lacks any definition of security constraints, which is typical of early-stage backlog items.

User Interaction Layer: The raw user story is selected from the backlog via the dashboard (Figure 6.4).

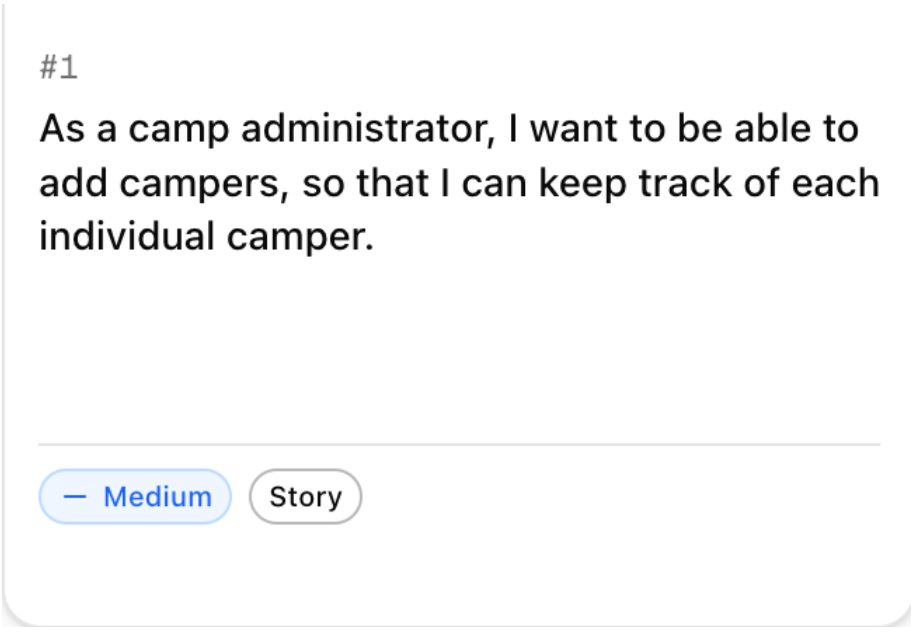

Figure 6.4: Raw user story state before LSS processing

Step 2: Execution Pipeline

System Processing Logic: The LLM component executes the analysis phases, hence generating security metadata, threat models, and explicit testable criteria.

User Interaction Layer: Upon initiation, the tool presents the execution phases as actionable buttons (Figure 6.5). These buttons allow the LLM component's analysis phases to be triggered sequentially: **S-Tag Generation**, **SAC Generation**, **Misuse Stories**, and **Risk Assessment**. The interface is designed to give users control over the analysis flow by enabling each phase to be initiated step by step from left to right.

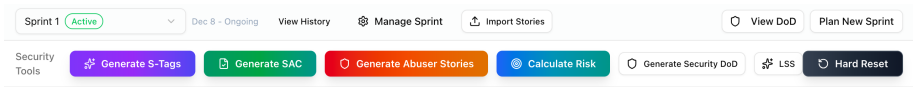

Figure 6.5: Analysis panel displaying the execution buttons for starting different analysis phases

The relationships between user stories and generated artefacts are further visualised in the Connection Graph (Figure 6.6), which maps each story to its corresponding security artefacts (e.g. OWASP A04: Insecure Design).

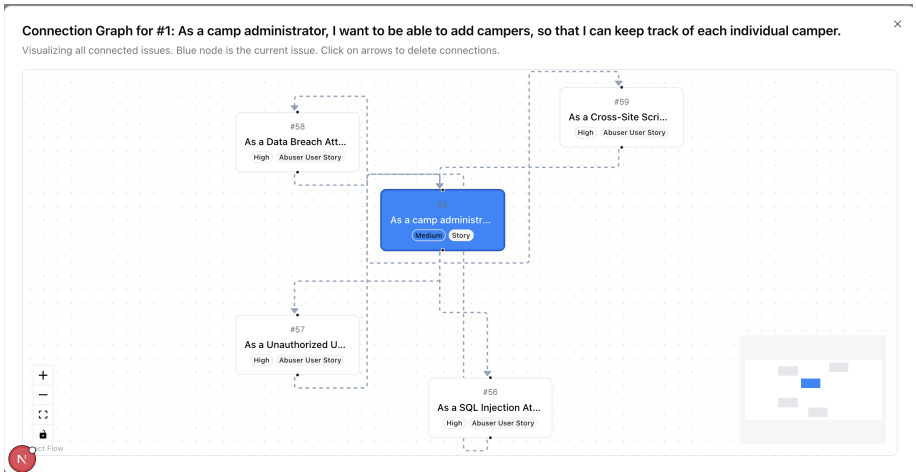

Figure 6.6: Connection graph visualizing the story's link to generated security artefacts

Step 3: Threat Modelling (Abuser Stories) Based on the identified risk tags, the tool generates Abuser Stories to model potential threats. Figure 6.7 presents the generated scenarios, such as an attacker attempting to inject malicious scripts through the camper name field (Stored XSS). This step demonstrates the application of the "Abuser Story" strategy by proactively identifying the "Attacker" persona and their specific "Action".

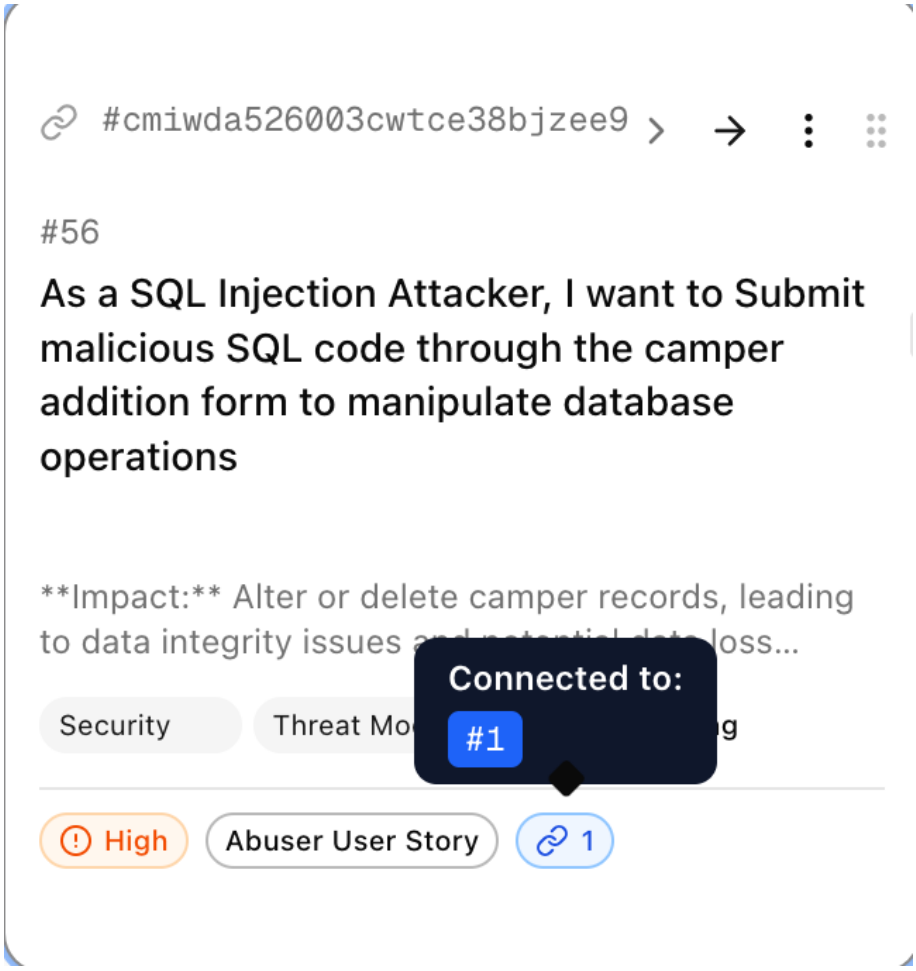

Figure 6.7: Abuser story identifying specific threat scenarios

Step 4: Requirement Generation (SAC) To mitigate the identified threats, the framework generates SAC (Figure 6.8). For example, to address the XSS threat, it specifies "Input validation for special characters" and references relevant controls

from OWASP ASVS v4.0 (V1–V14). This approach addresses the lack of concrete, testable security requirements by providing clearly defined and testable constraints.

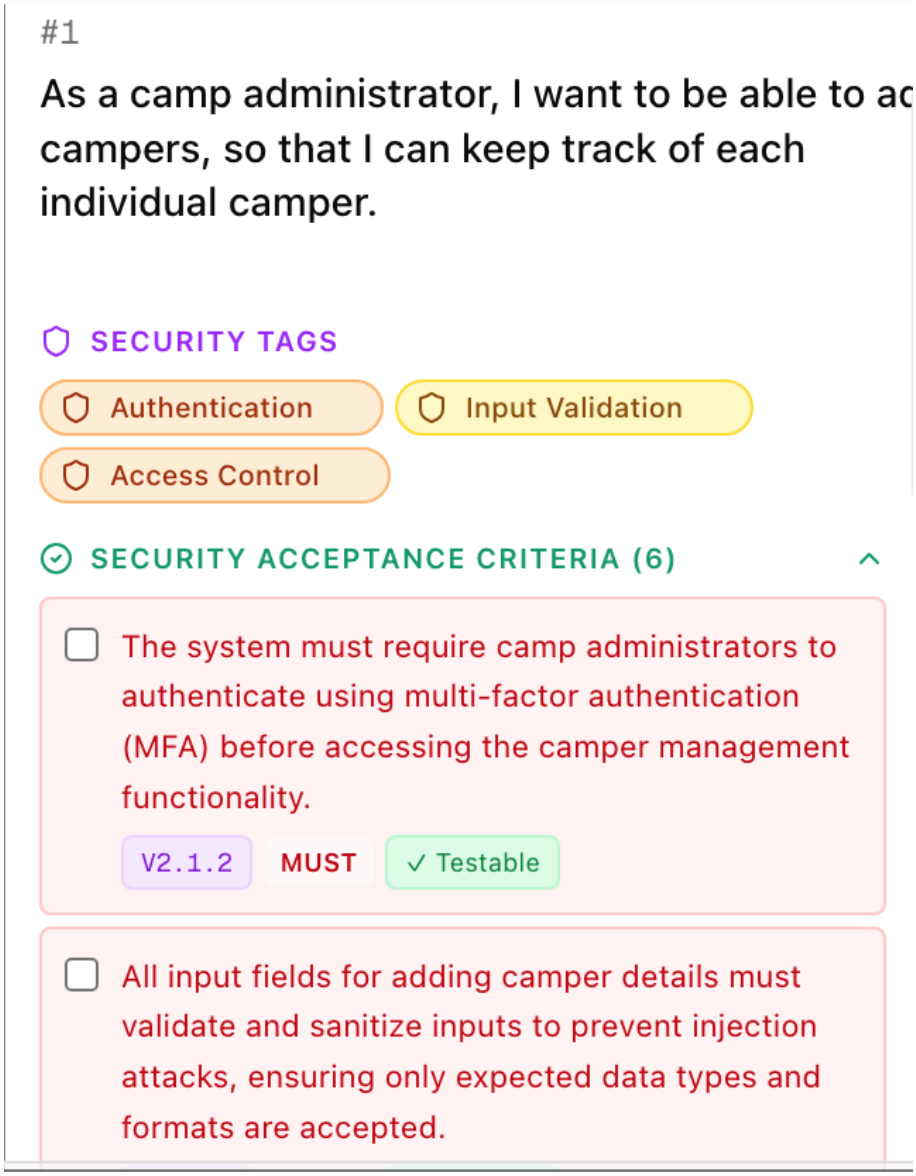

Figure 6.8: Generated SAC with ASVS mapping

Step 5: Final Output The process concludes with the fully enhanced user story (Figure 6.9). The story now includes a comprehensive security profile, consisting of a Risk Rating, S-Tags, SAC, and Abuser Stories, with minimal manual intervention beyond triggering the analysis steps.

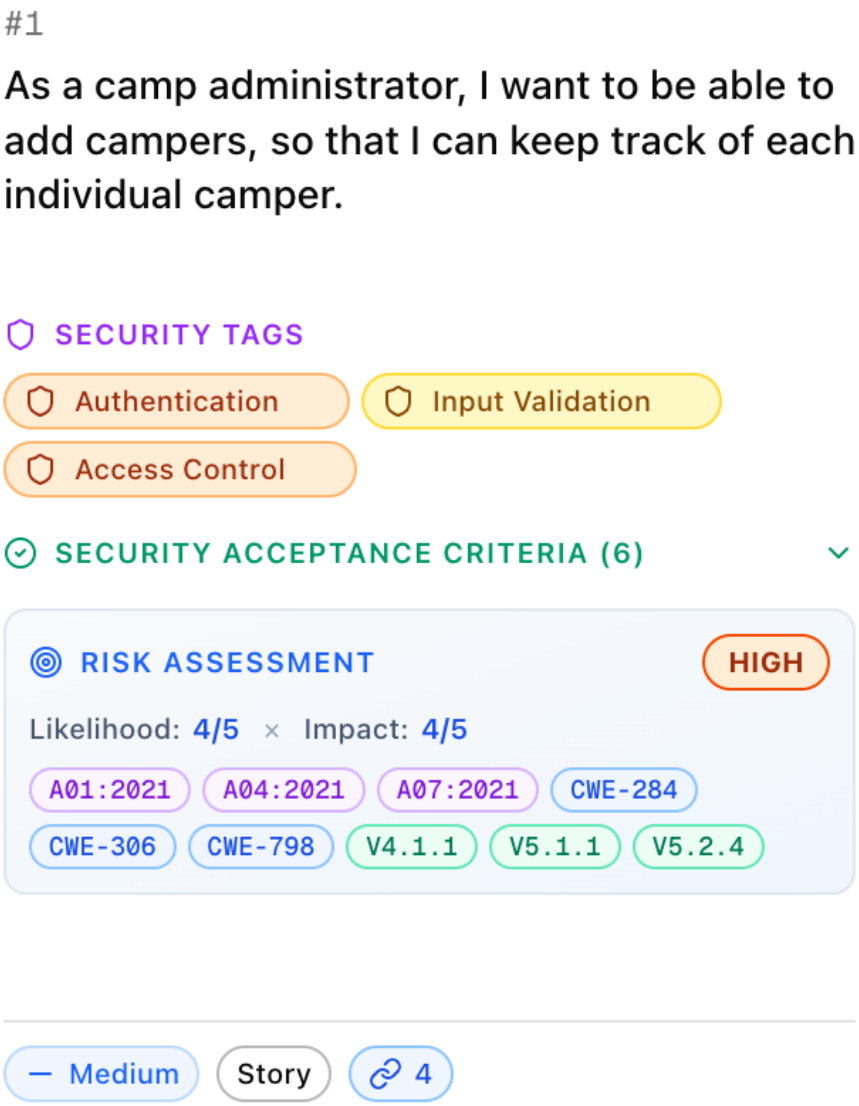

Figure 6.9: Final enhanced user story incorporating all security artefacts

Table 6.1 summarises the artefact-level transformation demonstrated in the walk-through. The purpose of this transformation is not to claim that the generated artefacts are complete or final security decisions, but to demonstrate that initially implicit security concerns can be converted into explicit, reviewable backlog artefacts.

This transformation demonstrates artefact-level feasibility: security concerns

Input artefact	**LSS-generated output artefacts**
Raw functional user story	Risk score, S-Tag, abuser/misuse story, security AC, ASVS mapping, Security DoD checklist, and enhanced user story

Table 6.1: Artefact-level transformation performed by the LSS tool

that would otherwise remain implicit are converted into reviewable backlog-level artefacts.

DoD Integration Finally, the tool generates a checklist for the DoD (Figure 6.10), ensuring that the security tasks are verifiable before the story is closed. These checkpoints are applied across all backlog items, meaning that every story must satisfy them to be considered a valid increment in Scrum.

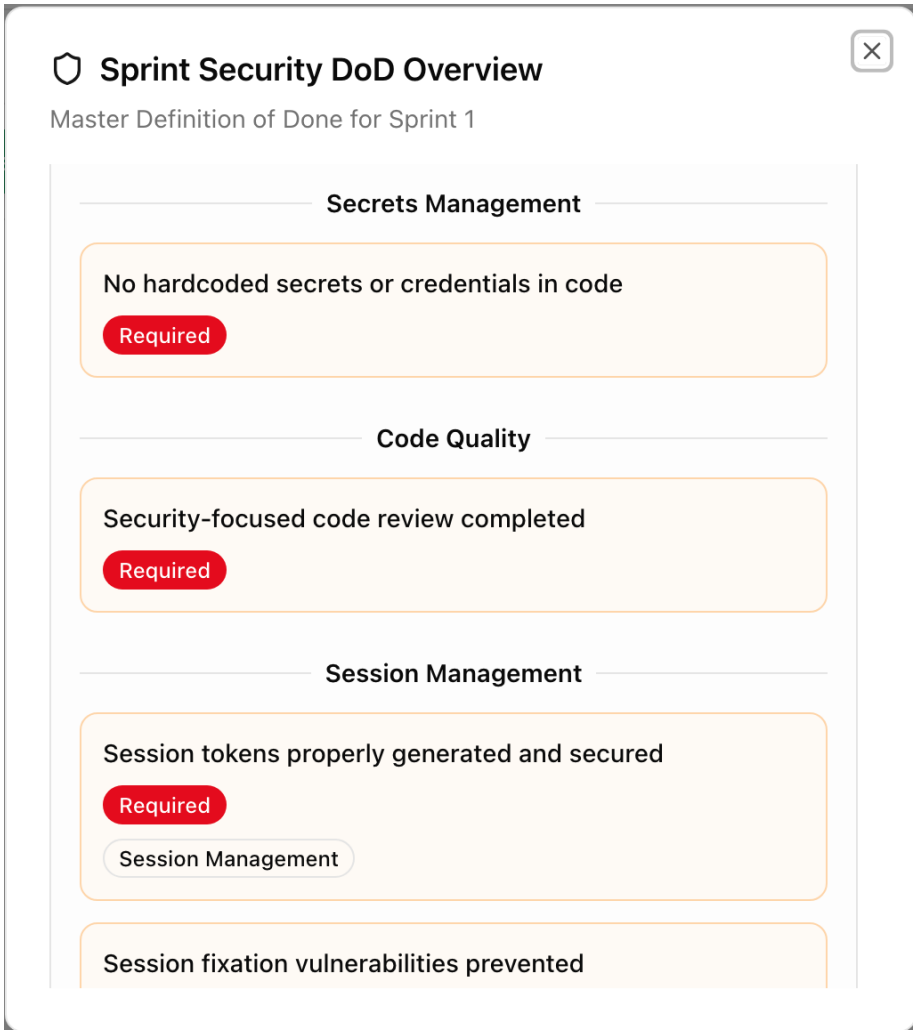

Figure 6.10: Generated security Definition of Done checklist

6.4 Validity and Rationale of Experimental Setup

The experimental setup is designed to support a controlled evaluation of the LSS framework in a context that reflects a practical Agile development environment. This evaluation mainly focuses on the framework's ability to systematically generate security artefacts from raw user stories. The results are complemented by a separate practitioner-based validation (Chapter 7), which examines perceived usefulness and feasibility in real-world Agile settings. The use of a publicly available

dataset consisting of 50 user stories from the Camp Administrator system supports reproducibility while also maintaining realism because it includes a range of functional requirements with varying levels of security relevance, such as data handling and user input scenarios.

The use of prompt-engineered LLM components promotes consistent and structured outputs. By providing established security standards, including OWASP Top 10 (2021) and OWASP ASVS v4.0 (V1–V14), as reference context within the prompts, the generated artefacts are guided by recognised security practices.

The processing pipeline, which transforms raw user stories into enriched security artefacts, is aligned with a typical Scrum backlog refinement process. This experimental design allows the evaluation to focus on how the framework supports the systematic integration of security into early-stage requirements without requiring manual security expertise.

The setup enables evaluation across three aspects: (1) identification of security-relevant requirements, (2) generation of structured and testable security artefacts, and (3) applicability within an Agile workflow. Overall, the experimental design provides an initial basis for assessing both the functional behaviour of the framework and its practical suitability for supporting secure Scrum practices.

7 Validation

This chapter presents the validation of the proposed LSS framework. The validation was carried out through a practitioner workshop involving industry experts from a specialised telecommunications and digital services context. The objective of this validation is not to measure empirical security outcomes, but to assess practitioner perceptions of the framework's usefulness, trustworthiness, and feasibility of integration. Tshe validation follows an exploratory and formative design, mainly focusing on evaluating the initial feasibility and perceived value of the framework rather than producing conclusive or generalisable results.

To clarify the scope of the validation, Table 7.1 summarises what the evaluation is intended to assess and what it does not claim to demonstrate. This distinction is important because the study evaluates the initial feasibility and perceived usefulness of the LSS framework rather than its long-term empirical effect on security outcomes.

Evaluation aspect	**In Scope**	**Out of Scope**
Framework usefulness	Practitioner-perceived usefulness of LSS artefacts such as S-Tags, misuse stories, SAC, and Security DoD	Statistically generalisable effectiveness
Tool feasibility	Whether an MVP can demonstrate the transformation of user stories into security-enriched backlog artefacts	Full production readiness or large-scale integration with tools such as Jira
LLM output value	Whether generated artefacts are perceived as useful when reviewed by humans	Fully autonomous security decision-making
Process fit	Whether experts perceive the approach as compatible with Scrum workflows	Measured long-term impact on sprint velocity or vulnerability reduction

Table 7.1: Scope and limitations of the LSS validation

7.1 Survey Design

The survey was designed to mainly evaluate the perceived usefulness, trustworthiness (expressed as practitioners' willingness to rely on LLM-generated outputs given manual review), and practical feasibility of the proposed LSS framework. In addition, the validation also assesses perceived alignment with the goals defined in this thesis,

particularly in terms of improving security visibility, integrating security practices into Scrum artefacts and events, and ensuring practical usability. These dimensions directly relate to RQ1 and RQ2, particularly by evaluating the practical applicability and usefulness of the proposed LLM-supported framework in supporting security integration in Scrum. Google Form was used to implement the survey and it was distributed to the participants.

The survey consists of three main sections. The first section captures participant background information to support the contextual interpretation of responses. The second section evaluates key components of the LSS framework using Likert-scale statements, in which respondents indicate their level of agreement on a five-point scale ranging from Strongly Disagree (1) to Strongly Agree (5). The final section includes open-ended questions designed to capture qualitative feedback.

The survey focuses on practitioners' perceptions rather than measuring post-deployment effectiveness, as the LSS framework is evaluated at a conceptual and prototype level. All responses were collected anonymously, and no personally identifiable information was recorded. Participation was voluntary, secured via a consent form, and data handling followed the institutional guidelines of the University of Turku and GDPR requirements.

Table 7.2 summarises the survey items in a condensed form. The full survey, including contextual descriptions for each construct and the complete questionnaire, is provided in Appendix A. The original questionnaire included additional contextual descriptions for each construct and used five-point Likert-scale response options ranging from 'Strongly Disagree' to 'Strongly Agree'. Minor wording simplifications have been applied for compactness while preserving the original meaning of each item.

Section	Question / Item
Consent	
Consent	I confirm that I have read the information and agree that my anonymised responses may be used for research purposes. (Yes / No)
Participant Context	
Role	Current role (Developer, Security Engineer, Product Owner, Scrum Master, Architect, Manager, Other)
Experience	Years of industry experience (0–2, 3–5, 6–10, 10+)
Context	Primary development context (Startup, Mid-size organisation, Large enterprise, Regulated industry, Other)
Visibility & Backlog Integration (S-Tags)	
S-Tags	S-Tags improve the visibility of security-related requirements within the product backlog.
Traceability	LLM-generated security artefacts linked to backlog items improve traceability and auditability.
Risk Awareness & Prioritisation	
Risk Scoring	Automated risk scoring supports informed prioritisation during Sprint Planning.
Centralisation	A centrally integrated LLM-based security framework is preferable to ad hoc LLM usage.
Open Question	What concerns do you have about LLM-generated risk assessments?
Security Acceptance & Verification	
SAC	SAC improve clarity and objectivity of security verification.

Section	Question / Item
Misuse Stories	Misuse and abuser stories improve understanding of potential threats.
Trust & Governance	
Trust	I am comfortable using LLM-generated security artefacts with manual review.
Process Impact	
DoD Debt	Security DoD helps prevent security-related technical debt.
DoD Governance	Security DoD provides governance for validating LLM-generated recommendations.
Velocity Concern	Security DoD may negatively impact development velocity.
Automation	LLM automation reduces effort to integrate security into Scrum.
Balance	The LSS framework balances security and Agile flexibility.
Context Awareness	Context-aware LLM recommendations (e.g. integrating the project domain into the prompt context) improve usefulness.
Adoption & Feasibility	
Barrier	Biggest barrier to adoption (open-ended).
Future Work	Desired future capabilities or extensions.
Comments	Additional comments or suggestions.

Table 7.2: Survey questionnaire items for the LSS framework

7.1.1 Demographics

The validation workshop involved four industry experts within the same company, with significant experience in Agile software development. The participants represented different roles within the Scrum process, hence ensuring evaluation from multiple perspectives, including management, development, and coaching. Although the sample size is small (N=4), the participants are experienced professionals, each with over ten years of industry experience. Table 7.3 provides details of the participant demographics. Following this sampling strategy, the survey was designed to collect structured feedback from the participants.

ID	Role	Experience	Industry Context
P1	Developer	10+ Years	Regulated (Finance/Healthcare)
P2	Manager	10+ Years	Large Enterprise
P3	Agile Coach	10+ Years	Large Enterprise
P4	Scrum Master	10+ Years	Large Enterprise

Table 7.3: Validation workshop participants

7.1.2 Methodology

The validation process followed an exploratory expert evaluation study, supported by a quantitative survey. The session was structured as follows:

1. **Framework Presentation**: A detailed walkthrough of the LSS framework and its components, including the S-Tagging mechanism, SAC generation, and the creation of misuse and abuser stories.

2. **Prototype Demonstration**: A live demonstration of an MVP tool (see Appendix B) implementing the core features of the framework, such as risk scoring and automated backlog tagging.

3. **Focus Group Discussion**: An open discussion in which participants provided qualitative feedback and shared practitioner perspectives on the framework's

applicability to their current workflows (e.g. Jira integration and backlog refinement), consistent with the established use of focus groups in software engineering research [84].

4. **Validation Survey**: A structured questionnaire used to quantify perceptions of utility, trust, and feasibility.

7.2 Analysis of Quantitative Survey Results

Following the presentation and demonstration, participants completed a survey evaluating specific components of the framework on a Likert scale (1=Strongly Disagree, 5=Strongly Agree). Descriptive statistics (mean values) were computed; no inferential analysis was performed due to the small sample size.

The survey results show consistent positive responses among participants regarding the framework's utility, particularly with respect to visibility and traceability. Figure 7.1 illustrates the average scores for key evaluation criteria. Due to the small sample size (N=4), variance and standard deviation are not reported.

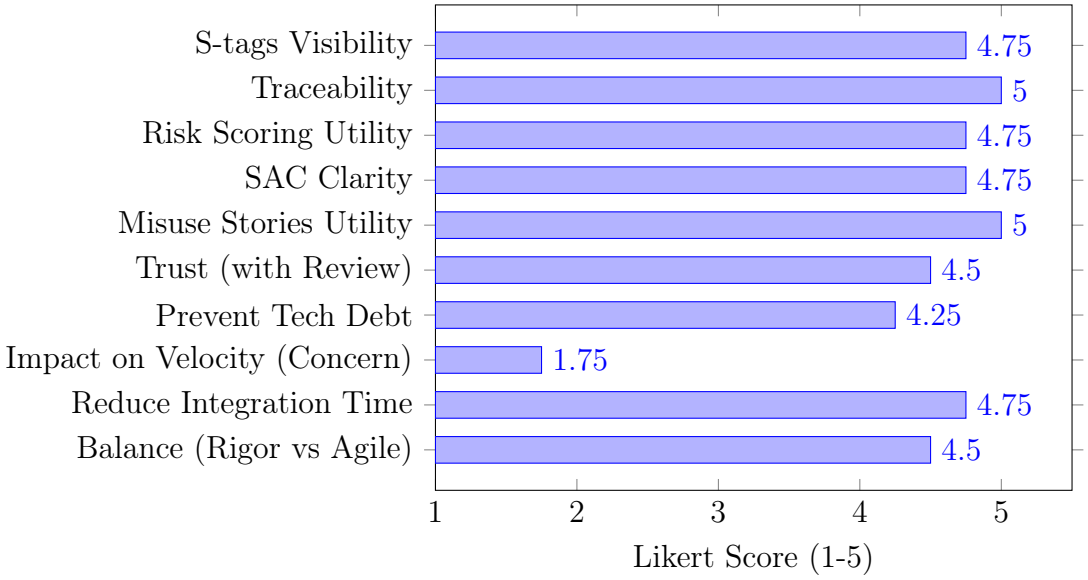

Figure 7.1: Average Likert scores from the validation survey (N=4, all participants with >10 years of experience). Note: For "Impact on Velocity", a lower score indicates lower concern (positive outcome).

The survey results indicate a generally positive evaluation of the framework across participants (N=4), particularly in terms of visibility and traceability. Within this small sample, all participants rated traceability at the maximum score (Mean = 5.0), while S-Tags and related artefacts received similarly high scores (Mean = 4.75), suggesting improved visibility of security requirements within the backlog.

Misuse stories were also rated at the maximum score by all participants (Mean = 5.0), indicating their perceived usefulness in representing security threats during requirements engineering. Concerns about development velocity were low (Mean = 1.75), suggesting minimal perceived impact from integrating security practices.

Trust in LLM-generated artefacts was high but conditional (Mean = 4.5), with responses highlighting the need for manual review. This aligns with the human-in-the-loop design of the framework, which requires manual developer review before accepting generated artefacts.

To further contextualise these results, Figure 7.2 groups the survey items by

evaluation dimension, distinguishing between perceived usefulness, trust and governance, and process-level impact. This grouping highlights that the framework scored consistently high on usefulness and visibility measures, while process-level concerns (such as impact on velocity) remained low.

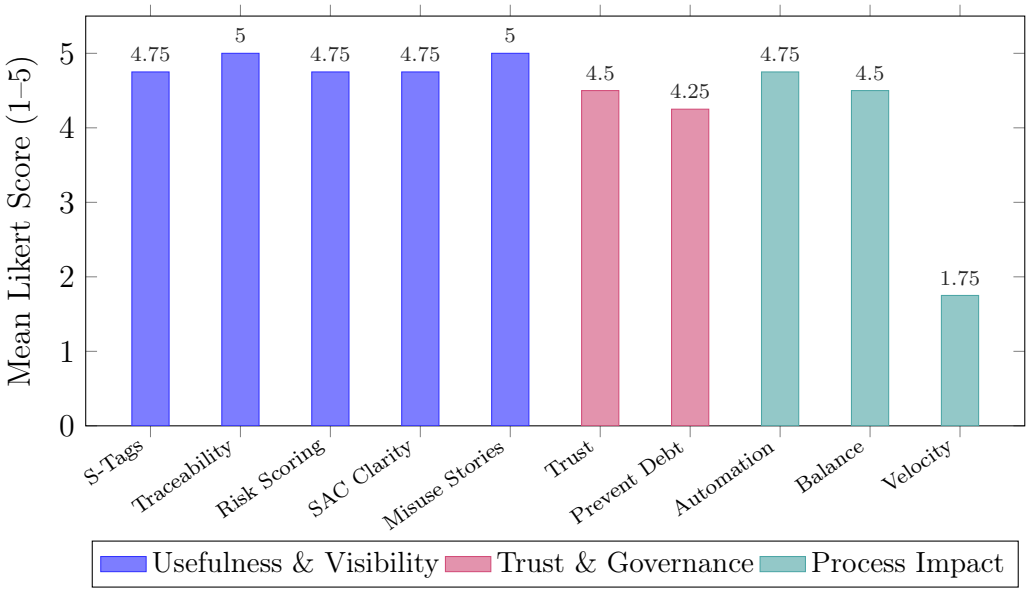

Figure 7.2: Validation scores grouped by evaluation dimension (N=4)

Table 7.4 presents a thematic synthesis of the validation results, highlighting a strong consensus on the framework's usefulness alongside certain technical and procedural prerequisites. As shown in the table, the most notable perceived improvements relate to the visibility of security requirements and the clarity provided by automated verification artefacts (SAC).

Category	Finding	Score
Key Perceived Improvements	Trust: Traceability & Auditability	5.00
	Awareness: Misuse Stories / Threats	5.00
	Visibility: S-Tags in Backlog	4.75
	Clarity: Verification via SAC	4.75
Adoption Conditions	Human-in-the-loop (Manual Review)	4.50
	Native Ecosystem Integration (Jira)	Qual.
Process Impact	Minimal Friction / Dev Overhead	1.75*

*Lower score indicates positive outcome

Table 7.4: Consolidated summary of validation findings showing high perceived utility alongside critical adoption requirements

7.3 Qualitative Feedback

The focus group discussion revealed several notable thematic insights regarding the practical application of the framework.

7.3.1 Integration with Existing Tools

A recurring theme was the necessity of seamless integration of the LSS framework with existing project management tools, specifically Jira[1]. Participant P1 (Developer) noted that while the concept is valuable, "Jira is so deeply integrated in the current development workflow that it would be impossible to start using some completely separate external tool." This observation was further reinforced by other participants, who highlighted that Jira is tightly coupled with surrounding tools and processes, including version control systems and documentation platforms, making any transition to an external standalone system impractical.

In addition to technical coupling, participants also emphasised the operational friction which is associated with introducing parallel tools. While workarounds such as exporting backlog data, processing it externally via the LSS framework tool, and re-importing enriched artefacts were discussed, these approaches were consistently characterised as "a hassle", hence indicating perceived usability barriers and lacking dynamic synchronisation with evolving project artefacts. This suggests that loosely coupled or manual integration strategies would likely hinder adoption due to added overhead and process disruption.

Despite these constraints, participants acknowledged potential intermediate use cases. For example, the framework could initially serve as a supporting tool during backlog refinement discussions, where AI-generated security artefacts provide additional context that can later be manually transferred into Jira. However, such usage was implicitly framed as a temporary or limited solution rather than a sustainable long-term approach.

Overall, the feedback indicates that successful adoption of the LSS framework depends on deep integration within existing ecosystems. Ideally, this can be achieved

[1]Jira is a widely used project management and issue-tracking tool developed by Atlassian, commonly used by Scrum teams to manage backlogs, sprints, and development workflows. See `https://www.atlassian.com/software/jira`

through native extensions such as plugins. Participants explicitly suggested leveraging Jira's plugin architecture and separating backend logic from the user interface to enable seamless embedding of LSS functionality into existing workflows. This reinforces the requirement that the framework should augment, rather than replace, established development environments, hence ensuring minimal disruption to current practices while enabling incremental adoption. This finding is consistent with empirical observations that security activities in Agile can introduce additional tools, documentation, and process overhead, and therefore need to be integrated in ways that preserve Agile teams' existing workflow and productivity [28].

7.3.2 Bridging the Knowledge Gap

Participants highlighted the framework's potential to bridge the gap between security expertise and generalist software development practices. As one participant noted, "People are experts on software development, but maybe not on application security, and we don't know what we don't know," pointing to a fundamental barrier to proactive risk identification. This reflects a recurring industry challenge, where developers have strong domain and implementation knowledge but lack specialised security expertise, leading to overlooked risks and late-stage issue discovery.

Several participants emphasised that security considerations are often not adequately discussed during backlog refinement and are instead treated as secondary or external concerns. In many cases, security input is introduced too late in the development lifecycle or remains abstract, requiring POs to make decisions without complete or structured information. This creates a disconnect between identified risks and actionable development practices, further reinforcing the knowledge gap within teams.

In this context, the proposed framework was perceived as a way to embed security knowledge directly into routine development activities. One participant described it

as functioning like a "security AI team member," bringing structured, domain-specific insights into backlog refinement sessions without requiring the continuous involvement of dedicated security experts. By generating artefacts such as S-Tags, SAC, and abuse stories, the framework provides developers with concrete guidance and reference points, enabling them to reason more effectively about security concerns.

Importantly, participants also noted that such support does not replace expert knowledge but instead enhances awareness and guidance. For example, automatically generated artefacts can direct developers to relevant standards and threat scenarios, helping them identify areas that require further investigation. This aligns with broader observations in the literature [20], [76], which indicate that a central challenge is often not only implementing security measures, but also recognising their necessity in the first place due to limited security awareness and the implicit nature of security requirements [12].

7.3.3 Risk Visibility for Product Owners

The framework's ability to make security risks explicit and standardised was perceived as particularly valuable for supporting decision-making by POs. Participants noted that, although security risks are often identified during development activities such as threat modelling, they are not always clearly communicated or prioritised in later backlog decisions. In practice, these risks are often reduced to simple labels or informal discussions, after which POs are expected to judge their importance without having sufficiently detailed or structured information.

This gap contributes to a recurring tendency to prioritise feature delivery over security concerns. As highlighted by participants, development teams often face situations where known risks are recognised but ultimately postponed due to business pressures or competing priorities. In such cases, the lack of clear and understandable representations of risk makes it easier to overlook or delay security-related work. This

dynamic corroborates prior studies noting that prioritisation mechanisms driven by business value frequently marginalise security requirements, particularly when security concerns are implicit, difficult to assess, or insufficiently visible to POs and other stakeholders [12], [47].

Within this context, the framework's use of explicit risk indicators, along with supporting artefacts such as misuse (abuse) stories, was seen as a way to improve risk visibility and communication. By translating abstract security concerns into more concrete, scenario-based representations, the framework helps make risks easier to understand for non-specialist stakeholders. One participant noted that such representations could make risks more "PO understandable," strengthening their role in prioritisation discussions and making them harder to ignore.

In addition, participants observed that increased risk visibility could support wider organisational communication. For example, clearly described risk scenarios and their potential impact can help bridge the gap between technical teams and business decision-makers, who might otherwise underestimate security issues. This supports the idea that improving the clarity and accessibility of risk information can influence prioritisation decisions, especially in environments where technical and security debt tend to build up over time.

Overall, the findings suggest that the framework supports more informed and transparent prioritisation by making security risks explicit, understandable, and directly connected to backlog artefacts.

7.3.4 Extensibility for Regulatory Compliance

Participants also emphasised that the framework is not limited to security, but can also be applied to other domains such as regulatory compliance and domain-specific requirements. In particular, the ability to use the same mechanism where backlog

items are enriched with structured, AI-generated artefacts was seen as useful for integrating compliance considerations directly into development workflows.

Several participants drew comparisons between security and compliance practices, noting that both are often treated as external or late-stage concerns. In many cases, teams depend on legal or compliance experts during the early planning phases but receive limited support during implementation. As a result, compliance issues are often identified only in later stages of development, sometimes close to release, where they can delay delivery and require costly rework.

In this context, the framework's extensibility was seen as a way to introduce regulation compliance knowledge earlier in the development lifecycle. By adapting the tagging and artefact generation approach (for example, replacing security tags with regulation-specific tags), teams could continuously integrate regulatory requirements into backlog refinement and development activities. This allows compliance considerations to evolve alongside functional requirements, rather than being introduced as a separate validation step.

Participants also noted that this approach could reduce the common problem of late-stage rejection by external stakeholders, such as legal teams. These legal teams may otherwise become involved only after significant development work has already been completed. By making compliance requirements visible and actionable at the backlog level, the framework supports earlier validation and reduces the risk of disruptive feedback near release.

The findings suggest that the framework offers a flexible foundation for integrating not only security but also broader regulatory and organisational requirements into Agile workflows, hence improving alignment between requirement definition, implementation, and validation. This aligns with prior literature indicating that regulation acts as a major driver for security engineering in Agile contexts and that

privacy and compliance requirements must be explicitly managed alongside functional development [48], [50].

7.4 Summary of Validation Findings

The validation results indicate a generally positive evaluation of the LSS framework across both quantitative and qualitative dimensions. The survey results show high perceived usefulness, particularly in terms of visibility, traceability, and threat awareness through artefacts such as S-Tags and misuse stories. Trust in LLM-generated artefacts was high but conditional on manual review, consistent with the human-in-the-loop design of the framework. Concerns about negative impact on development velocity were low.

The qualitative feedback highlighted the necessity of integration with existing tools such as Jira, the framework's potential to bridge the gap between security expertise and generalist development practices, its value in making security risks more visible and understandable for POs, and its extensibility towards regulatory compliance. These findings are discussed and interpreted in detail in the following chapter.

8 Discussion

This chapter interprets the validation findings presented in Chapter 7, positions them within the broader literature on secure Agile development, and discusses their practical implications and limitations.

8.1 Interpretation of Key Findings

The findings suggest that the framework may help address a key gap in Agile security, namely the lack of actionable and visible security requirements in the product backlog. The high utility scores for misuse stories and S-Tags indicate that structuring security knowledge into familiar Agile artefacts can be an effective approach in practical settings. However, successful adoption depends on integration with existing application lifecycle management tools (e.g. Jira) to avoid disrupting established workflows. Finally, the validation highlights the framework's extensibility, suggesting that the same LSS principles could also be applied to support regulatory compliance in highly regulated domains.

In addition, the results also indicate a potential for improving communication between technical and non-technical stakeholders. By representing security concerns through structured artefacts such as misuse stories, SAC, and risk indicators, the framework makes security information more accessible and easier to understand. This is particularly important in Agile environments, where prioritisation decisions are often made collaboratively and under time pressure.

These interpretations are further supported by the consolidated validation results (Table 7.4), which show consistently high scores across visibility and awareness dimensions, alongside critical adoption prerequisites such as human-in-the-loop review and native tool integration with existing development workflows. The combination of high perceived utility with low velocity concerns (Mean = 1.75) suggests that the framework's lightweight design is perceived as compatible with existing Scrum workflows, hence directly relating to RQ1's focus on integrating security practices without compromising agility.

A particularly interesting detail in the findings is the tension between the desire for automation and the strict requirement for human control. While the LSS framework is designed to lower cognitive overhead by automatically assisting in generating SAC and misuse stories, the expert validation opposed full autonomy and rated the human-in-the-loop review as highly critical (Mean = 4.50). This suggests that Agile teams value AI for drafting initial security artefacts and making security knowledge more visible, particularly when teams lack the expertise to create such artefacts from scratch. However, they insist that the final authority on risk acceptance must remain with human stakeholders. This positions the LLM as a support tool that helps generalist developers make better security decisions, rather than as an autonomous security system.

8.2 Alignment with Existing Literature

The validation findings align with the central observations synthesised in the literature review (Section 4.3.3), namely that security in Scrum is often weakened by low visibility, weak prioritisation, and limited developer security expertise. The positive expert feedback on S-Tags, misuse stories, SAC, and traceability suggests that the LSS framework addresses these issues by making security concerns explicit within existing Scrum artefacts rather than introducing a separate heavyweight process.

However, the findings also confirm concerns raised in prior work, including that security integration depends on human factors and organisational willingness to adopt new practices [20], as well as the challenge of fitting security activities into existing Agile workflows without disrupting them [12], [76]. For this reason, the LSS framework should be understood as a decision-support tool for Secure Scrum rather than a replacement for security expertise.

More specifically, the qualitative feedback on the framework's role in bridging the security knowledge gap supports the findings of Terpstra et al. [20], who found that security requirements are frequently poorly defined and owned within Agile teams. The LSS framework helps address this gap by generating structured artefacts such as misuse stories and SAC, which provide developers with clear, actionable security guidance without requiring deep security expertise. This is also in line with Bartsch's [49] finding that developer security awareness varies across teams and that development processes need to be adapted to make security knowledge easier to access and share.

Furthermore, the strong preference among participants for native tool integration connects to observations by Thool and Brown [28], who noted that security activities can be seen as cumbersome because they often involve more documentation and tools than Agile typically allows. Although their survey found that practitioners still viewed security activities positively overall, the emphasis on seamless integration suggests that how security is delivered matters as much as what security is delivered. The fact that experts in this study described Jira integration as a must-have for adoption reinforces this point, as adopting security in Scrum depends not only on the quality of the practices, but also on how smoothly they fit into the tools and workflows that teams already use.

The validation results also connect to the prioritisation challenges described by Türpe and Poller [12] and Tøndel et al. [47], who found that security requirements

can be neglected during backlog prioritisation because they are often implicit and remain invisible to key decision makers. The high scores for S-Tags (Mean = 4.75) and risk scoring (Mean = 4.75) suggest that the LSS framework's way of making security concerns visible and measurable can help address these prioritisation challenges, and support Product Owners in making better-informed decisions.

An important finding is how the framework relates to the common concern that security activities slow down Agile teams. Thool and Brown [28] found that while some practitioners reported increased time and occasional delays in feature deployment due to security tasks, most of those who responded to the optional velocity question experienced minimal impact on sprint velocity and viewed security activities positively. The LSS framework's low "Minimal Friction / Dev Overhead" score (Mean = 1.75, where lower is better) is consistent with this finding. Together, this suggests that security does not have to slow down Agile teams; rather, friction tends to arise when security is treated as a separate, heavyweight process. By embedding threat modelling (misuse stories) and verification (SAC) directly into the backlog items that developers already work with, the framework shows that strong security visibility can exist alongside a smooth development workflow.

8.3 Implications for Practice

The validation findings carry several practical implications for teams and organisations adopting security practices within Scrum.

As discussed above, the framework's integration of security into routine Scrum practices such as backlog refinement and sprint planning can help teams develop ongoing security awareness. At the same time, the strong preference for human-in-the-loop review confirms that the framework should be positioned as a decision-support tool, where human judgment remains central to validation and prioritisation.

From a practical standpoint, these findings suggest several considerations for

teams seeking to adopt LLM-supported security practices. First, organisations should ensure that any security integration framework is deployed as a plugin or extension within existing project management tools rather than as a standalone system, as the validation strongly indicated that adoption depends on seamless workflow integration. Second, the human-in-the-loop requirement implies that teams should establish clear review protocols for LLM-generated artefacts, treating them as initial drafts that require practitioner validation before being accepted into the backlog. This positioning as a decision-support mechanism rather than an autonomous tool directly supports RQ2's focus on how LLMs can practically assist security integration without replacing human expertise.

Building on the roles defined in Section 5.2.6, the validation findings highlight specific ways each role can benefit from the framework in practice:

- **Product Owners** can use the generated Risk Scores and S-Tags not as absolute mandates, but as data-driven input to justify prioritisation trade-offs between feature delivery and security debt.
- **Developers** can rely on the generated SAC as clear, testable boundaries for when a feature is considered secure, helping to prevent security-related scope creep.
- **Scrum Masters** should ensure that the human-in-the-loop review step is part of Sprint Planning or Backlog Refinement, and facilitate discussions around misuse stories so the team builds a shared understanding of threats before development begins.

8.4 Addressing the Research Questions

In relation to the research objectives, this thesis addressed the integration of security practices into Scrum (RQ1) and the role of Large Language Models (LLMs) in sup-

porting and enhancing these practices (RQ2). The findings indicate that security can be effectively embedded into Scrum workflows by aligning lightweight security practices with existing Agile artefacts, rather than introducing separate or heavyweight processes. In addition, the findings of this thesis suggest that LLM-based support can enhance these practices by improving consistency, accessibility, and traceability of security-related information.

Building on the identified challenges of visibility, prioritisation, and integration, this thesis developed a lightweight framework that uses LLMs to integrate these security practices into Scrum with minimal disruption. The framework focuses on embedding security directly into existing Scrum artefacts and workflows, with the aim of improving the continuity, visibility, and traceability of security without introducing heavy processes that could reduce process agility.

To demonstrate the practical applicability of the framework, a web-based tool was developed to simulate a Jira-like environment and illustrate the intended operational workflow. The tool shows how security artefacts, such as misuse stories, SAC, and dynamic DoD extensions, can be semi-automatically generated and integrated into the development workflow with human oversight.

To assess the framework, a preliminary evaluation was conducted with industry practitioners. The results indicate that the framework shows potential for use in real-world development environments, particularly in improving the visibility of security, reducing the security knowledge gap among developers, and supporting POs in understanding security risks. Importantly, the findings also suggest that these improvements may be achieved without significantly affecting development velocity, thereby addressing one of the key challenges identified in the literature.

Taken together, these findings and design outcomes demonstrate the main contribution of this thesis, which is the design and initial validation of a practical and adaptable LLM-supported Secure Scrum (LSS) framework that integrates security

practices directly into Agile workflows. The results demonstrate that combining lightweight security mechanisms with LLM-assisted support can improve security visibility, awareness, and decision-making while preserving the agility of Scrum. In this way, the framework contributes towards bridging the gap between existing Secure Scrum research and its practical adoption in real-world development environments.

8.5 Limitations and Challenges

Despite the positive feedback, the findings also highlight potential challenges. In particular, the reliance on accurate input data and context-aware LLM outputs may affect the quality of the generated artefacts. If backlog items are poorly defined, the usefulness of the resulting security artefacts may be reduced. This suggests that the effectiveness of the framework is closely linked to the quality of existing Agile practices and artefacts.

In addition, several LLM-specific limitations should be acknowledged. The quality and relevance of generated security artefacts may vary depending on the underlying model, prompt design, and the specificity of the project context provided [31], [32]. LLM outputs can also be affected by issues such as hallucination, where generated content appears plausible but is factually incorrect or contextually inappropriate [27], [30]. Although the human-in-the-loop design mitigates this risk by requiring manual review, it also introduces a dependency on the reviewer's ability to identify such errors, which may be limited in teams lacking dedicated security expertise. Furthermore, the cost and latency associated with LLM API calls may present practical barriers in environments with strict budget or performance constraints.

Overall, the validation provides initial evidence that the LSS framework is practical and valuable from a practitioner perspective, suggesting that integrating LLM-supported security practices into Scrum artefacts may improve visibility, awareness,

and decision-making without significantly disrupting existing workflows. These findings support RQ1 by illustrating how security practices can be embedded into Scrum workflows, and support RQ2 by showing how LLM-based assistance can support the practical implementation of these practices.

8.6 Threats to Validity

The classification of validity threats in this study follows established guidelines in empirical software engineering [85].

- **Internal Validity**: The validation was carried out through expert evaluation using a prototype demonstration rather than a real-world deployment. As a result, the observed outcomes are based on participants' perceptions and may be influenced by the controlled setting. In addition, no baseline comparison (e.g. traditional Secure Scrum practices without LLM support) was included, which limits the ability to attribute any improvements solely to the proposed framework.

  The absence of a baseline comparison was a deliberate consequence of the exploratory and design-oriented scope of the study. The purpose of the validation was not to prove causal superiority over existing Secure Scrum approaches, but to assess whether the proposed LSS framework is understandable, usable, and perceived as valuable by experienced practitioners. Therefore, the findings should be interpreted as evidence of initial feasibility and perceived utility rather than empirical proof of effectiveness. A future controlled comparison between ordinary Scrum, Secure Scrum, and LSS-supported Scrum would be required to evaluate relative effectiveness.

- **External Validity**: The validation was conducted with a small group of experts (N=4). Although the participants hold senior roles and have relevant

experience, the statistical significance of the quantitative results is limited. Furthermore, all participants were drawn from a specific organisational context (large enterprise/telecom), which may affect generalisability. The results may differ in smaller startups or in other domains.

- **Construct Validity**: The validation relied on expert feedback collected through questionnaires and discussions, which may introduce subjective bias. Furthermore, the evaluation of the framework's output was based on publicly available user stories rather than actual production user stories from the participants' organisation. Since real-world proprietary user stories may exhibit different levels of complexity and domain-specific ambiguity, the perceived usefulness of the generated artefacts might vary when applied to a live production backlog. In addition, the evaluation was conducted using an MVP prototype. Full integration into a production Jira environment may reveal additional challenges that were not captured in this study.

9 Conclusion

This thesis examined key limitations in integrating security into Scrum, namely that security concerns are often not visible, are poorly prioritised, and are difficult to incorporate without affecting agility. The literature review highlighted these challenges and provided evidence that security can be systematically integrated into Scrum through lightweight artefacts such as S-Tags, misuse stories, Security Acceptance Criteria (SAC), and Definition of Done (DoD) extensions. Furthermore, the review suggested that organisational changes, such as improving visibility and promoting shared responsibility, are necessary to support continuous security assurance.

To address the research objectives, this work focused on how security practices can be systematically integrated into Scrum (RQ1) and how Large Language Models (LLMs) can support and enhance their implementation (RQ2). A lightweight framework was proposed that embeds security directly into existing Scrum artefacts and workflows through LLM-based support, and a web-based prototype tool was developed to simulate a Jira-like environment and demonstrate the intended operational workflow with human oversight.

A preliminary evaluation with industry practitioners indicates that the framework shows potential for use in real-world development environments, particularly in improving the visibility of security, reducing the security knowledge gap among developers, and supporting POs in understanding security risks, without significantly affecting development velocity. Taken together, these findings demonstrate

the main contribution of this thesis, which is the design and initial validation of a practical and adaptable LLM-supported Secure Scrum (LSS) framework that integrates security practices directly into Agile workflows. The results demonstrate that combining lightweight security mechanisms with LLM-assisted support can improve security visibility, awareness, and decision-making while preserving the agility of Scrum. In this way, the framework contributes towards bridging the gap between existing Secure Scrum research and its practical adoption in real-world development environments.

This work has several limitations. As discussed in the experimental setup (Chapter 6) and validation (Chapter 7) sections, the evaluation was conducted on a limited scale with a small sample size and specific participant seniority, which restricts the generalisability of the findings. Ideally, the framework should be evaluated across multiple sprints in a real software development environment to better capture its practical impact over time. In addition, the framework was tested in a controlled prototype setting, and its effectiveness in large-scale or highly regulated industrial environments requires further investigation.

Addressing the identified limitations, future work should focus on extending this framework to support regulatory and compliance requirements, which was also highlighted as an important direction during the validation phase. This includes integrating standards and guidelines directly into the framework so that security practices are not only consistent but also aligned with industry regulations. Additionally, further research can explore large-scale industrial adoption, improved reliability of LLM-generated outputs, and deeper integration with existing development tools to enhance automation and usability.

Building on recent research in prompt engineering, core prompts could also be automated to some extent, reducing reliance on manual trial and error and supporting more adaptive refinement of prompts during and after sprints based on developer

and PO reviews. Furthermore, future work could also include tightly integrating the principles of the proposed framework with existing development tools such as Jira.

Ultimately, this thesis demonstrates the potential of combining lightweight Secure Scrum practices with LLM-assisted support to improve the integration of security within Agile software development. As software systems continue to grow in complexity and security requirements become increasingly important, approaches that maintain both security and agility will become more essential in modern development environments. The proposed framework represents a step towards more practical, accessible, and continuously integrated security practices within Scrum-based development.

References

[1] Digital.ai, "17th annual state of agile report", Digital.ai, Tech. Rep., 2023. [Online]. Available: https://digital.ai/resource-center/analyst-reports/state-of-agile-report.

[2] H. Dong, N. Dacre, D. Baxter, and S. Ceylan, "What is agile project management? developing a new definition following a systematic literature review", *Project Management Journal*, vol. 55, no. 6, pp. 668–688, 2024.

[3] K. Beck et al., *Manifesto for agile software development*, https://agilemanifesto.org, 2001.

[4] A. Mishra and Y. I. Alzoubi, "Structured software development versus agile software development: A comparative analysis", *International Journal of System Assurance Engineering and Management*, vol. 14, no. 4, pp. 1504–1522, 2023. DOI: 10.1007/s13198-023-01958-5.

[5] Y. Valdés-Rodríguez, F. Author2, F. Author3, and F. Author4, "Analysis of strategies for the integration of security practices in agile software development: A sustainable sme approach", *IEEE Access*, vol. 12, pp. 35 204–35 230, 2024.

[6] K. Rindell, S. Hyrynsalmi, and V. Leppänen, "Busting a myth: Review of agile security engineering methods", in *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES)*, ACM, 2017, pp. 1–10.

[7] H. Oueslati, M. M. Rahman, and L. B. Othmane, "Literature review of the challenges of developing secure software using the agile approach", in *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security (ARES)*, IEEE, 2015, pp. 540–547.

[8] K. Rindell, J. Ruohonen, J. Holvitie, S. Hyrynsalmi, and V. Leppänen, "Security in agile software development: A practitioner survey", *Information and Software Technology*, vol. 131, p. 106 488, 2021.

[9] A. Fan et al., "Large language models for software engineering: Survey and open problems", in *Proceedings of the 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, IEEE, 2023, pp. 31–53.

[10] A. Kuzior, I. Tiutiunyk, A. Zielińska, and R. Kelemen, "Cybersecurity and cybercrime: Current trends and threats", *Journal of International Studies*, vol. 17, no. 2, 2024.

[11] T. C. Zagita and T. Raharjo, "Information security integration with agile software development: Systematic literature review and expert judgement", *The Indonesian Journal of Computer Science*, vol. 12, no. 6, 2023.

[12] S. Türpe and A. Poller, "Managing security work in scrum: Tensions and challenges", in *Proceedings of the 9th International Workshop on Secure Software Engineering (SecSE@ESORICS 2017)*, CEUR-WS, 2017, pp. 34–49.

[13] K. Schwaber and J. Sutherland, *The scrum guide: The definitive guide to scrum: The rules of the game*, `https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf`, Scrum.org, 2020.

[14] C. Pohl and H.-J. Hof, "Secure scrum: Development of secure software with scrum", in *Proceedings of the 9th International Conference on*

Emerging Security Information, Systems and Technologies (SECURWARE), arXiv:1507.02992v1 [cs.CR], IARIA, Jul. 2015.

[15] A. Sharma and R. K. Bawa, "Identification and integration of security activities for secure agile development", *International Journal of Information Technology*, vol. 14, no. 2, pp. 1117–1130, 2022.

[16] G. McGraw, *Software Security: Building Security In*. Addison-Wesley, 2006.

[17] S. Samonas and D. Coss, "The cia strikes back: Redefining confidentiality, integrity and availability in security", *Journal of Information System Security*, vol. 10, no. 3, 2014.

[18] Y. Valdés-Rodríguez, J. Hochstetter-Diez, J. Díaz-Arancibia, and R. Cadena-Martínez, "Towards the integration of security practices in agile software development: A systematic mapping review", *Applied Sciences*, vol. 13, no. 7, p. 4578, 2023. DOI: 10.3390/app13074578.

[19] H. Villamizar, M. Kalinowski, M. Viana, and D. Méndez Fernández, "A systematic mapping study on security in agile requirements engineering", in *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2018, pp. 454–461. DOI: 10.1109/SEAA.2018.00074.

[20] E. Terpstra, M. Daneva, and C. Wang, "Agile practitioners' understanding of security requirements: Insights from a grounded theory analysis", in *Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, IEEE, 2017, pp. 439–442.

[21] Maria, R. Esteves, L. A. R. Jr, and N. A. Pinto, "Scrums: A model for safe agile development", in *Proceedings of the 7th International Conference on Management of Computational and Collective Intelligence in Digital EcoSystems (MEDES)*, ACM, 2015, pp. 43–47.

[22] I. Ghani and Z. Azham, "Integrating software security into agile-scrum method", *KSII Transactions on Internet and Information Systems*, vol. 8, no. 2, pp. 1–20, 2014.

[23] Z. Azham, I. Ghani, and N. Ithnin, "Security backlog in scrum security practices", in *Proceedings of the 2011 Malaysian Conference in Software Engineering (MySEC)*, IEEE, 2011, pp. 414–417.

[24] D. Baca and B. Carlsson, "Agile development with security engineering activities", in *Proceedings of the 2011 International Conference on Software and Systems Process (ICSSP)*, ACM, 2011, pp. 149–158.

[25] S. Minaee et al., *Large language models: A survey*, arXiv preprint arXiv:2402.06196, 2024.

[26] H. Cheng et al., "Generative ai for requirements engineering: A systematic literature review", *Software: Practice and Experience*, vol. 56, no. 2, pp. 141–170, 2026. DOI: 10.1002/spe.70029.

[27] Z. Ji et al., "Survey of hallucination in natural language generation", *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023. DOI: 10.1145/3571730.

[28] A. Thool and C. Brown, "Securing agile: Assessing the impact of security activities on agile development", in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, ACM, 2024, pp. 668–678.

[29] H. Naveed et al., *A comprehensive overview of large language models*, ACM Transactions on Intelligent Systems and Technology, vol. 16, no. 5, pp. 1–72, 2025.

[30] S. S. Rahman et al., "Hallucination to truth: A review of fact-checking and factuality evaluation in large language models", *Artificial Intelligence Review*, 2026.

[31] Y. Chang et al., *A survey on evaluation of large language models*, ACM Transactions on Intelligent Systems and Technology, vol. 15, no. 3, pp. 1–45, 2024.

[32] M. T. R. Laskar et al., "A systematic survey and critical review on evaluating large language models: Challenges, limitations, and recommendations", in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024, pp. 13 785–13 816.

[33] I. O. Gallegos et al., "Bias and fairness in large language models: A survey", *Computational Linguistics*, vol. 50, no. 3, pp. 1097–1179, 2024. DOI: 10.1162/coli_a_00524.

[34] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, *A systematic survey of prompt engineering in large language models: Techniques and applications*, arXiv preprint arXiv:2402.07927, 2024.

[35] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, *Unleashing the potential of prompt engineering for large language models*, Patterns, 2025.

[36] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 24 824–24 837.

[37] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic chain of thought prompting in large language models", *arXiv preprint arXiv:2210.03493*, 2022.

[38] Z. Wang, A. Liu, H. Lin, J. Li, X. Ma, and Y. Liang, "Rat: Retrieval-augmented thoughts elicit context-aware reasoning in long-horizon generation", *arXiv preprint arXiv:2403.05313*, 2024.

[39] S. Yao et al., "Tree of thoughts: Deliberate problem solving with large language models", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023, pp. 11 809–11 822.

[40] Q. Ye, M. Ahmed, R. Pryzant, and F. Khani, *Prompt engineering a prompt engineer*, Findings of the Association for Computational Linguistics: ACL 2024, pp. 355–385, 2024.

[41] B. Kitchenham and S. Charters, *Guidelines for performing systematic literature reviews in software engineering*, Tech. Rep. EBSE-2007-01, Jul. 2007.

[42] J. Thomas and A. Harden, "Methods for the thematic synthesis of qualitative research in systematic reviews", *BMC Medical Research Methodology*, vol. 8, no. 1, p. 45, 2008.

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

[44] J. de Vicente Mohino, J. B. Higuera, J. R. B. Higuera, and J. A. S. Montalvo, "The application of a new secure software development life cycle (s-sdlc) with agile methodologies", *Electronics*, vol. 8, no. 11, p. 1218, 2019.

[45] M. Herrmann et al., "From missile warhead to smart fridge: Interviews with industry experts on tracing safety- and security-relevant artifacts", *Journal of Systems and Software*, vol. 230, p. 112 551, 2025.

[46] H. Naji, M. Gutfleisch, and A. Naiakshina, "Relationship status: "it's complicated" developer-security expert dynamics in scrum", in *Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, IEEE Computer Society, 2025, pp. 657–657.

[47] I. A. Tøndel, D. S. Cruzes, M. G. Jaatun, and G. Sindre, "Influencing the security prioritisation of an agile software development project", *Computers & Security*, vol. 118, p. 102 744, 2022.

[48] K. Rindell, J. Ruohonen, and S. Hyrynsalmi, "Surveying secure software development practices in finland", in *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES)*, ACM, 2018, pp. 1–7.

[49] S. Bartsch, "Practitioners' perspectives on security in agile development", in *Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security (ARES)*, IEEE, 2011, pp. 479–484.

[50] E. D. Canedo, A. T. S. Calazans, I. N. Bandeira, P. H. T. Costa, and E. T. S. Masson, "Guidelines adopted by agile teams in privacy requirements elicitation after the brazilian general data protection law (lgpd) implementation", *Requirements Engineering*, vol. 27, no. 4, pp. 545–567, 2022.

[51] A. van der Heijden, C. Broasca, and A. Serebrenik, "An empirical perspective on security challenges in large-scale agile software development", in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ACM, 2018, pp. 1–4.

[52] C. Ascencão, H. Teixeira, J. Gonçalves, and F. Almeida, "Large-scale agile security practices in software engineering", *Information & Computer Security*, 2024.

[53] A. Mihelic, S. Vrhovec, and T. Hovelja, "Agile development of secure software for small and medium-sized enterprises", *Sustainability*, vol. 15, no. 1, p. 801, 2023.

[54] G. Kagombe, R. W. Mwangi, and J. M. Wafula, "Achieving standard software security in agile developments", in *Proceedings of the 11th International Conference on Information Communication and Management (ICICM)*, ACM, 2021, pp. 24–33.

[55] A. Mihelic, S. Vrhovec, B. Markelj, and T. Hovelja, "Delegation-based agile secure software development approach for small and medium-sized businesses", *IEEE Access*, 2024.

[56] L. B. Othmane, P. Angin, H. Weffers, and B. Bhargava, "Extending the agile development process to develop acceptably secure software", *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 6, pp. 497–509, 2014.

[57] J. Peeters, "Agile security requirements engineering", in *Symposium on Requirements Engineering for Information Security (SREIS)*, 2005.

[58] G. McGraw, P. Hope, and A. I. Anton, "Misuse and abuse cases: Getting past the positive", *IEEE Security & Privacy*, vol. 2, no. 3, pp. 90–92, 2004.

[59] A. A. Ardo, J. M. Bass, and T. Gaber, "Towards secure agile software development process: A practice-based model", *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 14, pp. 5159–5178, 2022.

[60] P. Maier, Z. Ma, and R. Bloem, "Towards a secure scrum process for agile web application development", in *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES)*, ACM, 2017, pp. 1–8.

[61] W. Neugent, "Acceptance criteria for computer security", in *AFIPS Conference Proceedings*, 1982.

[62] H. Villamizar, M. Kalinowski, A. F. Garcia, and D. Méndez, "An efficient approach for reviewing security-related aspects in agile requirements specifications of web applications", *Requirements Engineering*, 2020.

[63] G. C. B. C. Dalpra, V. B. L. de Paula, M. A. S. Belisário, M. J. M. Schettini, J. E. Fernandes, and T. Pedrosa, "Incorporating cybersecurity requirements in agile development processes", in *Proceedings of the 28th Workshop on Requirements Engineering (WER 2025)*, 2025.

[64] H. Rygge and A. Jøsang, "Threat poker: Solving security and privacy threats in agile software development", in *Proceedings of the Nordic Conference on Secure IT Systems (NordSec)*, Cham: Springer International Publishing, 2018, pp. 468–483.

[65] L. B. Othmane and A. Ali, "Towards effective security assurance for incremental software development: The case of zen cart application", in *Proceedings of the 2016 11th International Conference on Availability, Reliability and Security (ARES)*, IEEE, 2016, pp. 564–571.

[66] M. Voggenreiter and U. Schöpp, "Prioritizing industrial security findings in agile software development projects", in *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, IEEE, 2023, pp. 375–379.

[67] D. Ionita, C. V. D. Velden, H. K. Ikkink, E. Neven, M. Daneva, and M. Kuipers, "Towards risk-driven security requirements management in agile software development", in *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, Cham: Springer International Publishing, 2019, pp. 133–144.

[68] S. Nägele, L. Korn, and F. Matthes, "Adoption of information security practices in large-scale agile software development: A case study in the finance industry", in *Proceedings of the 18th International Conference on Availability, Reliability and Security (ARES)*, ACM, 2023, pp. 1–12.

[69] D. Baca, M. Boldt, B. Carlsson, and A. Jacobsson, "A novel security-enhanced agile software development process applied in an industrial setting", in *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security (ARES)*, IEEE, 2015, pp. 11–19.

[70] T. D. Oyetoyan, D. S. Cruzes, and M. G. Jaatun, "An empirical study on the relationship between software security skills, usage and training needs in agile settings", in *Proceedings of the 2016 11th International Conference on Availability, Reliability and Security (ARES)*, IEEE, 2016, pp. 548–555.

[71] I. A. Tøndel and D. S. Cruzes, "Continuous software security through security prioritisation meetings", *Journal of Systems and Software*, vol. 194, p. 111 477, 2022.

[72] K. Rindell, S. Hyrynsalmi, and V. Leppänen, "Aligning security objectives with agile software development", in *Proceedings of the 19th International Conference on Agile Software Development: Companion (XP 2018)*, ACM, 2018, pp. 1–9.

[73] N. Chakraborty, S. Iqbal, and M. Zulkernine, "Building secure software for smart aging care systems: An agile approach", in *Proceedings of the 2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS)*, IEEE, 2024, pp. 562–571.

[74] N. Singh, P. Patel, and S. Datta, "A survey on security and human-related challenges in agile software deployment", in *Proceedings of the 2021 IEEE International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2021, pp. 1976–1982.

[75] A. Selva-Mora and C. Quesada-López, "Security practices in agile software development: A mapping study", in *Proceedings of the 7th ACM/IEEE International Workshop on Software-intensive Business (IWSiB)*, ACM/IEEE, 2024, pp. 56–63.

[76] K. R. Riisom, M. S. Hubel, H. M. Alradhi, N. B. Nielsen, K. Kuusinen, and R. Jabangwe, "Software security in agile software development: A literature review of challenges and solutions", in *Proceedings of the 19th International*

Conference on Agile Software Development: Companion (XP 2018), ACM, 2018, pp. 1–5.

[77] M. Daneva and C. Wang, "Security requirements engineering in the agile era: How does it work in practice?", in *Proceedings of the 2018 IEEE 1st International Workshop on Quality Requirements in Agile Projects (QuaRAP)*, IEEE, 2018, pp. 10–13.

[78] T. Ayalew, T. Kidane, and B. Carlsson, "Identification and evaluation of security activities in agile projects", in *Proceedings of the Nordic Conference on Secure IT Systems (NordSec)*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 139–153.

[79] A. A. Ardo, J. M. Bass, and T. Gaber, "Towards secure agile software development process: A practice-based model", in *Proceedings of the 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2022, pp. 149–156.

[80] S. Kopczyńska, M. Ochodek, J. Piechowiak, and J. R. Nawrocki, "On the benefits and problems related to using definition of done – a survey study", *Journal of Systems and Software*, vol. 193, p. 111 479, 2022. DOI: 10.1016/j.jss.2022.111479.

[81] OWASP Foundation, *Owasp risk rating methodology*, https://owasp.org/www-community/OWASP_Risk_Rating_Methodology, 2021.

[82] *Iso/iec 27005:2018 information technology – security techniques – information security risk management*, International Organization for Standardization, 2018.

[83] Joint Task Force Transformation Initiative, "Guide for conducting risk assessments", National Institute of Standards and Technology (NIST), Tech. Rep. NIST Special Publication 800-30 Revision 1, 2012.

[84] J. Kontio, L. Lehtola, and J. Bragge, "Using the focus group method in software engineering: Obtaining practitioner and user experiences", in *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE)*, IEEE, 2004, pp. 271–280. DOI: 10.1109/ISESE.2004.1334901.

[85] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering* (Computer Science). Berlin: Springer, 2012, vol. 236.

Appendix A LSS Survey Instrument

LSS Survey

This survey evaluates the proposed LLM-Supported Secure Scrum (LSS) framework, which integrates Large Language Models (LLMs) into Scrum to enhance security practices through automated S-Tags, risk assessment, Security Acceptance Criteria (SAC), misuse/abuse story generation, and Security Definition of Done (DoD) support.

As part of this research, prototype tools were developed to automatically analyse Product Backlog Items (PBIs), classify security relevance, generate structured security artefacts aligned with OWASP and ASVS standards, and provide decision-support metadata for prioritisation and governance.

The survey was implemented using Google Forms and distributed to software professionals with experience in Agile development and/or application security.

The form consists of three sections:

1. Participant background
2. Evaluation of LSS components (Likert-scale)
3. Open-ended feedback

Likert-scale questions range from Strongly Disagree (1) to Strongly Agree (5).

No personally identifiable information is collected. Participation is voluntary

and responses are anonymised in accordance with GDPR and University of Turku policies.

Consent: I confirm that I have read the above information and voluntarily agree that my anonymised responses may be used for academic research purposes.

Participant Context

Current role:

- Developer
- Security Engineer
- Product Owner
- Scrum Master
- Architect
- Manager
- Other: _____

Years of industry experience:

- 0–2
- 3–5
- 6–10
- 10+

Primary development context:

- Startup / Small team
- Mid-size organisation
- Large enterprise
- Regulated industry (finance, healthcare, etc.)
- Other: _____

Visibility & Backlog Integration

S-Tags

S-Tags (Security Tags) are lightweight classification markers attached to Product Backlog Items (PBIs) to indicate security-relevant concerns.

S-Tags improve the visibility of security-related requirements within the product backlog.

1	**2**	**3**	**4**	**5**
SD		N		SA

Having AI-generated security artefacts linked directly to backlog items would improve traceability and auditability.

1	**2**	**3**	**4**	**5**
SD		N		SA

Risk Awareness & Prioritisation

Automated risk scoring would support more informed prioritisation of security-relevant backlog items during Sprint Planning.

1	**2**	**3**	**4**	**5**
SD		N		SA

A centrally integrated LLM-based security framework is preferable to developers using ad hoc LLM prompts.

1	**2**	**3**	**4**	**5**
SD		N		SA

Concerns about AI-generated risk assessments:

Security Acceptance & Verification

Explicit Security Acceptance Criteria (SAC) make security verification clearer and more objective.

	1	2	3	4	5
	SD		N		SA

Misuse / Abuse Stories help identify potential threats better than standard requirements.

	1	2	3	4	5
	SD		N		SA

Trust & Governance

I would feel comfortable using AI-generated security artefacts with manual review.

	1	2	3	4	5
	SD		N		SA

Process Impact

Security Definition of Done (DoD) helps prevent security-related technical debt.

	1	2	3	4	5
	SD		N		SA

Security DoD provides an effective governance mechanism for validating AI-generated outputs.

	1	2	3	4	5
	SD		N		SA

Security DoD may negatively impact development velocity.

	1	2	3	4	5
	SD		N		SA

LLM-based automation can reduce the effort of integrating security into Scrum.

	1	2	3	4	5
	SD		N		SA

The LSS framework balances security and Agile flexibility.

1	**2**	**3**	**4**	**5**
SD		N		SA

Project-specific context would improve LLM-generated recommendations.

1	**2**	**3**	**4**	**5**
SD		N		SA

Adoption & Practical Feasibility

Biggest barrier to adoption:

Desired future capabilities:

Additional comments or suggestions:

Appendix B LSS Tool Code

The web tool for the LSS application can be found at: LSS Scrum Tool Web Repository.

Alternate Link: https://github.com/W44/lss-scrum-tool-web