*Master Thesis*
*Computer Science*
*January 2026*


# A Multi-Agent Approach for a Security-Aware Translation of Software Business Requirements

**Savvas Mantzouranidis**

Dept. Computer Science & Engineering
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Department of Computer Science & Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full-time studies.

**Contact Information:**
Author:
Savvas Mantzouranidis
E-mail: samn24@student.bth.se

University advisor:
Prof. Ricardo Britto
Dept. Computer Science & Engineering

Dept. Computer Science & Engineering
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se/didd
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

# Abstract

**Context**. The translation of high-level business requirements into detailed, security-compliant software specifications is a critical yet often neglected phase of the software development lifecycle. Traditional Security Requirements Engineering (SRE) is frequently manual, error-prone, and disconnected from the rapid pace of modern development, creating a "translation gap" that leaves software vulnerable to security threats.

**Objectives**. In this study, we investigate how a Multi-Agent System (MAS) powered by Large Language Models (LLMs) can automate this translation process. The primary objective is to design and validate an architecture that systematically bridges the gap between abstract business needs and actionable security controls, ensuring alignment with industry standards such as OWASP ASVS, NIST SP 800-53, and ISO 27001.

**Methods**. Following the Design Science Research Process (DSRP), a prototype system comprising ten specialized agents was developed, utilizing Retrieval-Augmented Generation (RAG) to ground outputs in verified security knowledge. The system was empirically evaluated through 14 industrial use case generations and validated via a survey of 15 software engineering experts who assessed the utility and quality of the generated security reports.

**Results**. The automated system achieved a mean quality score of 0.85 out of 1.0, with particularly high performance in consistency (0.91) and standards alignment (0.89). The parallel execution of specialized agents reduced analysis time by approximately 40% compared to sequential processes. Expert practitioners reported a high intent to adopt the system (4.13/5), confirming its practical utility, though feedback indicated a need for better integration with development tools to improve implementability.

**Conclusions**. We conclude that decomposing SRE tasks into specialized agent roles significantly enhances the reliability and efficiency of security analysis compared to generic LLM approaches. We also conclude that the integration of RAG is essential for preventing hallucinations and ensuring factual compliance. Moreover, while the generation of requirements is effectively solved, future work must address the direct integration of these agents into DevSecOps pipelines to fully realize "Shift Left" security.

**Keywords:** Large Language Models, Security Requirements Engineering, Multi-Agent Systems, Threat Modeling

# Acknowledgments

I would like to express my sincere gratitude to Dr. Ricardo Britto, my supervisor at BTH, for his continuous support and expert guidance throughout this research. His advice on shaping the research direction and aligning the thesis with practical industry needs has been essential to its completion.

I am also deeply grateful to my colleagues at the University of Bristol and the European Patent Office, as well as my software developer friends from various companies who generously agreed to participate in the evaluation survey. Their willingness to contribute their time and professional expertise has been fundamental to the validation of this research.

Lastly, I would like to express my gratitude to my family and friends for their support throughout the course of this project. Their encouragement has been invaluable to me in completing this research. To all, thank you for being pillars of support throughout this journey.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| **A2A** | Agent2Agent Protocol |
| **AI** | Artificial Intelligence |
| **ASVS** | Application Security Verification Standard |
| **CPS** | Cyber-Physical Systems |
| **CVE** | Common Vulnerabilities and Exposures |
| **DFD** | Data Flow Diagram |
| **DSRP** | Design Science Research Process |
| **HNSW** | Hierarchical Navigable Small World |
| **ISO** | International Organization for Standardization |
| **IEC** | International Electrotechnical Commission |
| **ISMS** | Information Security Management Systems |
| **ISTDRE** | Identification of Security Threats during Requirement Engineering |
| **IT** | Information Technology |
| **KGMAF** | Knowledge-Guided Multi-Agent Framework |
| **LLM** | Large Language Model |
| **MARE** | Multi-Agents Collaboration Framework for Requirements Engineering |
| **MAS** | Multi-Agent Systems |
| **MCP** | Model Context Protocol |
| **MFA** | Multi-Factor Authentication |
| **MOSRE** | Model Oriented Security Requirements Engineering |
| **NIST** | National Institute of Standards and Technology |
| **NLP** | Natural Language Processing |
| **OWASP** | Open Worldwide Application Security Project |
| **PII** | Personally Identifiable Information |
| **RAG** | Retrieval-Augmented Generation |

| | |
|---|---|
| **RE** | Requirements Engineering |
| **ReAct** | Reasoning and Acting |
| **SAST** | Static Application Security Testing |
| **DAST** | Dynamic Application Security Testing |
| **SIEM** | Security Information and Event Management |
| **SLA** | Service Level Agreement |
| **SRE** | Security Requirements Engineering |
| **SRS** | Software Requirements Specification |
| **STORE** | Security Threat Oriented Requirements Engineering Methodology |
| **STRIDE** | Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege |
| **TLS** | Transport Layer Security |

# Chapter 1

# Introduction

The rapid evolution of software-intensive systems has profoundly reshaped industries and societies, embedding complex functionalities into nearly every aspect of modern life. This digital transformation, while offering unprecedented opportunities, also introduces significant challenges in software product management, where balancing innovation with quality and security remains a persistent struggle [31]. Central to navigating this complexity is the discipline of Requirements Engineering (RE), which serves as the foundational phase for defining the capabilities and constraints of a software system. However, the process of eliciting, analysing, and specifying requirements is fraught with difficulties, including stakeholder communication gaps, changing requirements, and ensuring completeness and correctness, particularly in large-scale projects [45].

Among the myriad of considerations in RE, ensuring robust security has emerged as a paramount concern. Failing to integrate security requirements effectively from the outset can lead to severe vulnerabilities, significant financial losses, and a decline in user trust. organizations are increasingly dependent on information technology applications for critical business operations, online transactions, and communication, making software security a necessity rather than a desirable feature [30, 8]. However, traditional approaches to Security Requirements Engineering (SRE) often struggle to keep pace with the dynamic threat landscape and the agile nature of modern software development [35]. A persistent challenge in current practice is the translation gap: high-level business requirements frequently lack the necessary actionable security controls, and the manual, error-prone nature of traditional SRE processes makes the development of secure software inefficient and costly [14, 5].

In recent years, remarkable advancements in Artificial Intelligence (AI), particularly in the domains of Large Language Models (LLMs) and multi-agent systems, have presented transformative potential for automating and enhancing various facets of software engineering [5, 21]. LLMs, trained on vast text datasets, have demonstrated impressive capabilities in tasks such as text generation, understanding, and reasoning [14, 26, 27]. These models offer promising opportunities for automating aspects of RE, including requirements generation, analysis, and specification. Meanwhile, multi-agent systems provide structural advantages over

single-agent approaches: task specialization enables agents to develop domain-specific expertise, parallel processing improves efficiency, and collaborative validation mechanisms create natural error-checking [11, 13, 12]. However, despite these capabilities, existing multi-agent systems have primarily focused on code development and general requirements engineering, largely neglecting the specialized and critical domain of security requirements engineering.

This research explores how an intelligent, multi-agent system can be engineered to systematically translate high-level business requirements into detailed, security-compliant software requirements, thereby addressing a critical gap in current practice. The thesis investigates the integration of LLMs with established security methodologies and frameworks, such as Security Threat Oriented Requirements Engineering Methodology (STORE) and Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege (STRIDE), within a multi-agent architecture designed specifically for security-centric requirements engineering. The research further addresses the security considerations inherent in deploying LLM-integrated multi-agent systems for sensitive SRE tasks, ensuring that the solution is not only effective but also trustworthy and robust. Through this comprehensive approach, the thesis aims to advance the state of the art in automated security requirements engineering and provide practical solutions for enhancing software security in industrial contexts.

## 1.1 Problem Statement

The escalating dependency on complex Information Technology (IT) applications highlights an urgent need for robust software security, engineered systematically from the earliest stages of development [30, 8]. However, a fundamental challenge persists: the inefficient and error-prone translation of high-level business requirements into detailed, actionable, and security-compliant software requirements [5, 2, 45]. Traditional SRE methodologies often fail to integrate security consistently, treating it as an afterthought or a separate activity, leading to critical security aspects being overlooked and costly vulnerabilities emerging late in the development life cycle [33, 30, 18]. This issue is particularly acute in large-scale industrial settings where manual processes are time-consuming, prone to human error, and contribute to significant project delays and resource inefficiencies [5].

The advent of LLMs offers a promising avenue for automating and enhancing RE tasks, including generation, analysis, and specification [14, 10, 26, 15]. However, the direct application of generic LLMs to complex SRE challenges is hampered by inherent limitations such as a lack of domain-specific knowledge, the potential for generating erroneous or uncontextualized information (hallucinations), and the substantial computational resources required [26, 10, 14]. Moreover, while multi-agent systems are emerging as powerful tools in software development, their current designs largely prioritise code generation and have not yet

adequately addressed the nuanced and critical tasks of security requirements elicitation and translation [15, 27, 17, 37, 9]. This creates a "translation gap" where high-level strategic documents frequently lack the necessary actionable security controls, and existing solutions fall short in providing a systematic, efficient, and robust automated mechanism [14, 5].

While single LLM approaches have shown promise in automating various software engineering tasks, they face significant limitations when applied to the complex, multi-dimensional challenge of requirements translation and security compliance. Multi-agent systems offer several critical advantages over monolithic single-LLM solutions [32]. First, task specialization enables individual agents to develop domain-specific expertise in areas such as requirements analysis, security policy interpretation, and compliance verification, leading to improved accuracy and quality of outputs [11, 13]. Second, parallel processing capabilities allow multiple agents to work simultaneously on different aspects of the translation process, significantly reducing response times and improving overall system efficiency [12, 7]. Third, multi-agent architectures provide enhanced fault tolerance and redundancy, where if one agent encounters issues, other agents can continue functioning and compensate for failures, ensuring robust system operation [4, 23]. Additionally, collaborative validation between agents creates natural error-checking mechanisms, with research showing that multi-agent systems achieve higher accuracy compared to single LLMs in complex reasoning tasks [43, 19]. Finally, the modular and scalable nature of multi-agent systems allows for easier extension and adaptation as new security requirements or compliance frameworks emerge, making them particularly suitable for the evolving landscape of software engineering requirements [22].

Therefore, the core problem addressed by this research is the absence of a systematic, automated, and empirically validated approach to bridge the gap between high-level software requirements and their security-compliant, detailed specifications, particularly in industrial contexts. This gap is a result of the limitations of manual processes, the unaddressed challenges of applying generic LLMs, and the underdeveloped application of multi-agent systems specifically for security requirements engineering. Resolving this problem is crucial for developing secure software efficiently and effectively, thereby enhancing the overall quality and trustworthiness of IT systems.

## 1.2 Research Objectives

The aim of this thesis is to design, develop, and evaluate a novel multi-agent system that automates the translation of high-level software requirements into detailed, security-compliant software requirements. To achieve this aim, the following objectives have been defined:

- **OBJ1: To identify and conceptualise the challenges** in translating

high-level requirements into security-compliant specifications, drawing from a rapid literature review [29] and practitioner insights [45, 26, 10, 18].

- **OBJ2: To design and develop a multi-agent system architecture** and its collaborating agents, integrating relevant security frameworks and knowledge bases to facilitate systematic security requirements elicitation.
- **OBJ3: To empirically evaluate the effectiveness, efficiency, and accuracy** of the developed multi-agent system in generating security-compliant requirements compared to established manual and AI-augmented methods.
- **OBJ4: To assess the practical utility and feasibility** of the multi-agent system from an industry perspective through qualitative feedback.

## 1.3 Scope and Limitations

This thesis focuses on the automated translation of high-level software requirements into detailed, security-compliant software requirements using a multi-agent system powered by LLMs. The research encompasses the design, development, and evaluation of a prototype multi-agent system that integrates prominent threat-oriented methodologies, such as STORE [2] and STRIDE [30], along with other relevant security frameworks [17, 15, 41]. The application and validation of this system draw upon industry-representative cases, with a particular focus on scenarios prevalent in industry verticals such as e-commerce, healthcare, and financial services, to ensure the practical relevance and generalizability of the findings [5, 2, 45, 21]. The primary emphasis remains on enhancing the early phases of the software development process, specifically the requirements engineering stage, to ensure security is embedded from conception [15, 30].

This work contributes to the field of security requirements engineering in several ways:

1. A novel multi-agent architecture and methodology specifically tailored for automated security requirements elicitation, demonstrating the integration of LLMs with established security frameworks for systematic threat analysis and requirement generation.
2. Empirical evidence and insights into the applicability, performance, and limitations of AI-driven approaches for security requirements engineering within industrial contexts, addressing a current gap in validation.
3. A validated framework for integrating security standards and best practices into an automated RE process, providing a pathway for more rigorous and comprehensive security considerations from the outset.

4. A prototype multi-agent system that significantly enhances the efficiency and accuracy of generating security-compliant requirements, reducing time and effort for identifying and incorporating security aspects early in the software development lifecycle, along with guidance and actionable recommendations for practitioners on leveraging multi-agent LLM systems for improved software security development.

### 1.3.1 Limitations

This research acknowledges several inherent limitations that warrant consideration. The inherent probabilistic nature of LLMs leads to variations in output across different runs, posing a threat to reproducibility [14, 1, 6]. While this study mitigated this by fixing random seeds where feasible, future industrial deployments should enforce reproducibility by "freezing" specific model versions (e.g., gpt-5-mini-2025-08-07) or utilizing local, deterministic open-source models. The baseline security requirements used for evaluation are derived from human experts, whose interpretations may vary, introducing a potential bias in the expert baseline. This internal validity threat is addressed by involving multiple experts, facilitating consensus-building processes, and providing standardized instructions for baseline generation [14].

The evaluation environment, while striving to be industry-representative, does not perfectly replicate all complexities of real-world project conditions. Efforts were made to include industry-relevant cases to enhance ecological validity, but a full-scale deployment validation is beyond the scope of this thesis [26, 21]. The developed system is a prototype that demonstrates the feasibility and utility of the proposed approach, rather than a fully developed, enterprise-ready solution. While the research integrates several key security frameworks and standards, it does not cover every existing methodology exhaustively, with the selection based on their prominence and relevance to the identified problem.

Finally, two methodological limitations regarding evaluation metrics must be noted. First, while expert reviews were utilized, a formal inter-rater agreement metric (e.g., Cohen's kappa) was not calculated. Given the subjective nature of requirements engineering, the evaluation relied on expert consensus regarding the utility of the outputs rather than statistical agreement on specific phrasing. Second, this study functioned as a feasibility analysis of the Multi-Agent Systems (MAS) architecture and did not include a direct manual control group (Human vs. AI) for efficiency comparison. Consequently, performance claims regarding time savings are calculated against sequential automated processes rather than human baselines.

## 1.4 Thesis Outline

The remainder of the thesis is organized as follows:

- **Chapter 2: Background** provides a comprehensive overview of the core concepts, including RE, key Security Standards and Frameworks, the architecture and application of MAS, and the role of LLMs in RE.
- **Chapter 3: Related Work** surveys existing research on security requirements engineering methodologies, AI-augmented RE, and multi-agent systems in software development, highlighting current approaches and identifying limitations and gaps.
- **Chapter 4: Research Methodology** details the research methodology, adopting a Design Science Research Process (DSRP). It outlines the research design, data collection and evaluation methods, and discusses potential threats to validity and the strategies employed to mitigate them.
- **Chapter 5: Solution Design and Implementation** presents the design and implementation of the proposed multi-agent system. It covers the high-level architecture, the specific roles and designs of each agent, their communication and coordination mechanisms, the structure of the security knowledge base, and details of the implementation.
- **Chapter 6: Evaluation Design and Results** describes the evaluation design and presents the results. This includes the evaluation methodology, the analysis of the system's effectiveness in relation to RQ1, an examination of the architecture and collaboration for RQ2, a performance evaluation against RQ3, and findings from expert and industry validation.
- **Chapter 7: Discussion and Analysis** provides an in-depth discussion and analysis of the research findings. It analyses the results in the context of each research question, discusses the theoretical and methodological contributions, addresses the system's limitations, and explores the implications for both practice and future research.
- **Chapter 8: Conclusions and Future Work** summarises the research, reiterates the key findings and contributions, discusses the limitations, and proposes directions for future research.

# Chapter 2

# Background

This chapter provides the foundational knowledge necessary to understand the core concepts underlying this thesis. It begins by introducing RE and its security-specific considerations, followed by an overview of relevant security standards and frameworks. Subsequently, it presents LLMs and MAS, highlighting their capabilities and general challenges. Finally, the chapter elaborates on the DSRP, the methodological approach adopted for this study.

## 2.1 Requirements Engineering

RE is a fundamental aspect of the software development life cycle, defining the framework and primary objectives that guide the creation of software applications [10]. It encompasses a disciplined and structured approach to consistently define, document, and maintain requirements throughout the software development process [10]. Requirements are crucial at various stages, from high-level stakeholder analysis and system requirements to detailed component specifications [26]. Traditional RE approaches involve a systematic progression of stages, starting with the elicitation of high-level customer requirements, followed by detailed feasibility analysis, and concluding with their verification and strategic allocation to sub-system modules for development [5]. This process is often time-consuming and human-intensive, heavily relying on manual practices, especially in industrial settings [15, 5]. The manual nature of these practices can lead to inconsistencies and prolonged feedback cycles, negatively impacting the quality of the developed system [5].

RE can be broadly categorized into requirements development, which includes elicitation, analysis, and specification, and requirements management, a continuous process covering change requests, documentation, and traceability [10]. Requirement engineers traditionally tend to focus predominantly on functional requirements, often overlooking or treating security requirements as secondary [33]. This overemphasis on functional aspects can lead to critical security issues being missed in the early stages of development [33, 30]. As modern software complexity and system interconnection increase, the RE process mandates alignment with standards, infrastructure specifications, and customer expectations

[5]. The rapidly changing business environments further challenge traditional RE approaches, necessitating more effective methodologies [35].

### 2.1.1 Security Requirements Engineering

Security requirements are a type of non-functional requirement, primarily concerned with confidentiality, integrity, and availability [2, 33]. They are necessary when stakeholders identify valuable objects within the software system that require protection [2]. SRE should be systematic, repeatable, and capable of eliciting complete, reliable, clear, simple, and easy-to-analyse requirements [2]. However, security requirements are often developed independently of other RE activities, leading to critical security aspects being overlooked as engineers focus on functional requirements [2, 33, 18, 35]. Security, specifically, is recognized as an essential non-functional requirement that has historically proven difficult to address, particularly within agile development contexts [35].

Given the increasing dependence on information technology applications for business, online transactions, and communication, software security has become a critical necessity [30, 2]. Software security vulnerabilities and flaws are outcomes of poorly built software that can lead to easy exploitation by attackers [2]. Inappropriate security requirements engineering is one reason for developing software products of poor quality [2]. SRE aims to integrate security considerations from the early phases of software development to ensure the creation of secure and high-quality software products [30, 2, 33]. Eliciting all security requirements before product development is a complex task [2, 33]. Most requirement engineers are proficient in functional requirements but lack deep knowledge in authentic security requirements engineering [2]. Requirement engineers frequently concentrate on functional requirements, often failing to adequately cover security aspects, especially when they have limited knowledge in this domain [33]. This can lead to ambiguous information and an overall lack of well-assembled security requirements engineering approaches [33].

### 2.1.2 The Requirements Translation Challenge

A persistent challenge in SRE is the "Translation Gap" which is a distinct phenomenon from requirements ambiguity. While ambiguity refers to unclear or vague input text, the Translation Gap refers to the semantic disconnect between clear high-level business goals and the specific, actionable technical security controls required to satisfy them. Business stakeholders rarely articulate the technical countermeasures (e.g., 'implement TLS 1.3' or 'enforce bcrypt hashing') needed to secure their features, creating a void that manual translation often fills inconsistently [5, 14].

## 2.2 Security Standards and Frameworks

To ensure robust security in software systems, various standards and frameworks provide essential guidelines and best practices. These structured approaches help organizations identify, implement, and manage security controls effectively. Security standards serve the critical purpose of providing guidelines for security implementation, ensuring consistency, and facilitating compliance across different organizations and contexts [30, 39]. Three major frameworks are particularly relevant to this work: International Organization for Standardization (ISO) 27001, National Institute of Standards and Technology (NIST) Special Publication 800-160, and the Open Worldwide Application Security Project (OWASP) Application Security Verification Standard (ASVS).

ISO/International Electrotechnical Commission (IEC) 27001 is an international standard that provides a framework for Information Security Management Systemss (ISMSs) [39]. It helps organizations manage the security of assets such as financial information, intellectual property, employee details, and information entrusted by third parties [39]. Mapping security requirements to standards like ISO 27001 is a common challenge for organizations [39, 30]. The NIST provides a suite of documents and frameworks that are highly influential in guiding system security engineering. NIST Special Publication 800-160 Volume 1 Revision 1, titled *Engineering Trustworthy Secure Systems*, provides comprehensive guidance for integrating security throughout the system development life cycle, from conception and design to deployment and maintenance [30]. This document emphasizes a systems-oriented approach to security, addressing both technical and organizational aspects [30].

The OWASP provides a wealth of resources, guidelines, and methodologies for improving software security. The OWASP ASVS provides a basis for testing application technical security controls and serves as a structured framework for security verification [39]. Threat modeling is a crucial concept within these guidelines, serving to identify potential security risks early in the development lifecycle and incorporate mitigations into the core design of the system [18]. Frameworks like STRIDE, which categorize threats into Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege, are commonly utilized for threat modeling [41, 30]. These standards and frameworks form the knowledge base that must be integrated into automated SRE approaches to ensure security-compliant requirements.

## 2.3 Multi-Agent Systems

MASs are composed of multiple interacting intelligent agents that collaborate to achieve complex goals [15, 17]. In an MAS, complex problems are often decomposed into smaller tasks, with specialized agents collaborating to complete each

part of the workflow [9]. This division of labor allows MASs to handle tasks that would be difficult or inefficient for a single agent to handle [15, 17]. Agents in a MAS typically have defined roles and communicate through shared representations or direct message passing [17, 15]. The benefits of using MASs for complex tasks include parallel processing, specialization of agents for specific sub-tasks, and the ability to maintain a coordinated workflow through shared knowledge structures [17, 15].

MASs typically involves agents with defined roles that communicate and share information to achieve tasks. For instance, agents might upload intermediate artifacts to a shared workspace where properties such as content, role, and flow among agents facilitate coordination [17]. Alternatively, agents might continuously monitor a shared artifacts pool to plan and execute actions based on its state [15]. These architectural patterns, whether based on shared workspaces, blackboard mechanisms, or direct communication, enable agents to collaborate effectively on multi-stage workflows [17, 15]. Each agent can be equipped with specific functionality, predefined actions, and domain knowledge, allowing it to contribute its specialization to the overall goal of the system [15].

However, the interaction among intelligent agents in MASs also introduces novel challenges and security vulnerabilities distinct from those in traditional systems, including emergent behaviors, potential for covert collusion, coordinated attacks, and cascade failures [9]. Safeguarding MASs requires careful design considerations, such as hierarchical information management and memory protection, which involve classifying information by security levels to restrict sensitive data access to authorized agents [20]. These foundational concepts of MASs form the basis for understanding how they can be applied to automate complex software engineering tasks, including requirements engineering.

## 2.4 Large Language Models

LLMs represent a significant advancement in AI, particularly within the domain of Natural Language Processing (NLP). Trained on vast text datasets, LLMs demonstrate impressive capabilities across a wide range of tasks, including text summarization, question answering, and code generation [14, 26, 27, 10]. These models offer transformative potential for SRE by enhancing precision and reducing ambiguities in requirement specifications [14, 10, 26]. LLMs, with their advanced NLP capabilities, are being explored for tasks such as generating requirements, classifying them, and ensuring their quality [10, 26]. They can generate requirements in natural language or structured forms based on input text [26]. LLMs can also translate requirements into equivalent forms, such as formal languages, semi-structured formats, or requirements modeling languages [26]. This includes improving the completeness of requirements by using mechanisms to predict missing elements based on context [26, 14].

Effective prompt engineering, involving strategically designed task-specific instructions, is crucial to guide LLM behavior without altering model parameters and improving output reliability [10]. The concept of prompt engineering involves crafting inputs that help LLMs produce more accurate and contextually appropriate outputs for specific tasks [10, 28]. Automating various RE tasks through LLMs is seen as a way to expedite development processes [5, 15, 17]. However, integrating AI, particularly LLMs, into SRE introduces its own set of challenges. These include domain ignorance, hallucinations that generate factually incorrect but plausible text, and high computational costs, which can hinder their widespread adoption [10, 14, 26]. LLMs may also struggle with domain-specific subtleties, bias from training data, difficulties in explainability, and a general lack of contextual understanding, necessitating human oversight [10, 14]. Such limitations highlight the need for continued investigation and the development of effective mitigation strategies [10].

The integration of LLMs into applications also introduces new attack surfaces and security risks [16, 8]. These include prompt injection, where malicious input manipulates LLM behavior, and data leakage, where sensitive information is revealed [8, 16, 1]. Other vulnerabilities include bias, toxic outputs, privacy breaches, and disinformation [8, 16]. Defenses for LLM-integrated applications require properties such as integrity, source identification, attack detectability, and utility preservation [16]. These foundational concepts of LLMs are essential for understanding how they can be leveraged within multi-agent architectures for automated requirements engineering tasks.

# Chapter 3

# Related Work

This chapter reviews the literature pertinent to the core areas of this thesis: the application of LLMs in RE, SRE methodologies, MASs for software engineering tasks, and security challenges in LLM-integrated MASs. The aim is to establish the current state of the art, identify existing limitations and gaps, and position the contributions of this research.

## 3.1 Large Language Models in Requirements Engineering

LLMs and Generative AI are fundamentally reshaping the software development life cycle, holding significant potential for transforming RE [10, 3]. LLMs excel at generating human-like content in response to complex prompts, offering unique capabilities that span various RE tasks [25]. Research suggests that LLMs can augment and streamline tasks across the primary stages of RE: elicitation, specification, analysis, and validation [15, 3].

In the area of requirements generation, LLMs can generate requirements in natural language or structured forms based on input text [26, 3]. They enhance brainstorming, foster creativity, and facilitate the provision of real-time feedback [10]. For instance, LLMs can generate refined requirements as user stories, potentially incorporating details inferred from their extensive domain knowledge, which contributes to the maturation and enrichment of requirements [44, 3, 10, 28]. Studies show that LLMs can generate use case specifications, sometimes presenting more comprehensive alternative flows compared to human-generated content [28]. Beyond generation, LLMs are capable of translating requirements into equivalent forms, such as semi-structured formats conforming to templates or requirements modeling languages [26]. This translation capability extends to converting high-level requirements into more detailed representations that can support downstream development activities [26, 3].

LLMs also demonstrate capabilities in quality assessment and management. They can assess quality at the individual requirement level, evaluating factors such as ambiguity and conformance to guidelines, as well as at the document

level, finding redundancy or conflicts [26]. This includes extracting glossary terms, checking conformance to templates, and potentially handling anaphoric ambiguity [3]. The capability of LLMs to detect semantic similarity can be applied to find conflicting or redundant requirements [26]. Furthermore, LLMs can list requirement identifiers related to features [14], and by simulating continuous feedback loops, they can enhance change impact analysis and proactively predict requirement changes [3]. LLMs are also crucial in generating technical documentation, plans, and reports, ensuring accuracy, consistency, and accessibility of technical content [10].

To overcome the inherent limitations of LLMs, such as domain ignorance and hallucinations, specific prompt engineering techniques have been developed [10, 28]. These techniques emphasise providing context, examples, and specific instructions to guide LLM behavior [10]. A taxonomy of prompt patterns for RE tasks has been proposed, including cognitive verifier patterns for classification and tracing tasks that instruct the LLM to break down complex tasks into smaller subtasks, and context manager patterns that require the LLM to provide explanations, reasoning, and address potential ambiguities in its response [28, 14]. These patterns add layers of transparency and validation to LLM outputs, improving reliability for RE applications [28].

Despite their transformative potential, LLMs are not a silver bullet for RE problems [3]. Critical challenges hinder their adoption. Technical challenges include domain ignorance, the risk of hallucinations that generate inaccurate or irrelevant information, high computational costs, and inherent biases from training data [10, 3]. LLMs often suffer from inconsistencies among generated artifacts, particularly between design and implementation, which lead to traceability concerns [28]. They may struggle with deep-seated domain nuances, leading to outputs lacking contextual understanding [10, 3]. A significant issue is the transferability of models: while models may excel on datasets similar to those on which they were trained, their performance can drastically decrease when applied to external domains [24]. Process risks, such as over-automation and over-specification, can lead to a loss of the human-centric view of requirements, emphasizing the necessity of human oversight [10, 3]. These limitations suggest that highly rigorous and specialized domains, such as SRE, require hybrid validation pipelines and techniques to mitigate domain-specific gaps [10].

## 3.2 Multi-Agent Systems in Software Engineering

The complexity of software engineering tasks, particularly RE, which traditionally involves iterative collaboration among multiple roles such as analysts and stakeholders [17, 15], has led to the emergence of MASs as a promising automation paradigm. This section examines MASs for software engineering, beginning with foundational concepts, then progressing to general applications in software

engineering, and finally focusing on security-specific tasks.

### 3.2.1 Foundations of Multi-Agent Systems

MASs are computational systems composed of multiple intelligent agents that interact and collaborate to solve complex problems that individual agents would find difficult to address [15, 17]. The fundamental principle underlying MASs is the decomposition of complex tasks into smaller, manageable sub-tasks, with specialized agents responsible for different aspects of the overall workflow [9]. This division of labour enables parallel processing and leverages the specialization of individual agents, improving both efficiency and the quality of outcomes [17, 15].

Agent communication and coordination are central to MAS architectures. Agents typically coordinate through shared representations, such as a shared workspace or artifacts pool, where intermediate results and knowledge are stored and accessed by multiple agents [17, 15]. In such architectures, artifacts possess properties that detail their content, the role of the agent that created them, and their flow among agents, facilitating structured collaboration [17]. Alternatively, agents may continuously monitor the state of a shared repository, such as a blackboard mechanism, to plan and execute actions based on changes to that state [15]. These architectural patterns enable agents to work asynchronously yet cohesively towards a common goal, with each agent contributing its specialized capability to the larger task [17, 15].

The benefits of MASs for complex tasks are manifold. By distributing work across multiple specialized agents, MASs can handle tasks that exceed the capacity or context limits of individual agents [34]. Specialization allows each agent to focus on a narrow domain, potentially improving the quality of its contributions [15]. The collaborative nature of MASs also enables iterative refinement, where agents can review and improve each other's outputs, leading to more robust final results [17, 27]. These foundational concepts of task decomposition, agent specialization, and coordinated collaboration provide the basis for applying MASs to software engineering domains.

### 3.2.2 Multi-Agent Systems for General Software Engineering

MASs have demonstrated significant potential in various software engineering tasks, primarily focusing on code development and requirements engineering [15, 27, 17]. Examples of general-purpose MAS frameworks include MetaGPT and ChatDev, which automate aspects of code generation by assigning different development roles to different agents [15, 17]. However, recent research has increasingly focused on designing MASs specifically for the upstream task of RE, recognising that RE complexities have often been neglected in earlier code-generation-focused systems [15].

The Multi-Agents Collaboration Framework for Requirements Engineering (MARE) framework is a prominent example of LLM-based MAS designed to achieve end-to-end RE automation [17]. MARE divides the RE process into four macro-tasks: elicitation, modeling, verification, and specification, and employs five specific agents: Collector, Modeler, Checker, Stakeholder, and Documenter [17]. These agents perform nine defined actions, and collaboration is facilitated via a shared workspace where agents upload intermediate artifacts such as user stories and requirement drafts [17]. Each artifact possesses properties that detail its content, the role associated with it, the action that caused its creation, and information about which agents sent and should receive it, ensuring a structured artifact flow [17]. Agents in MARE iteratively collaborate to establish a common understanding of the envisioned product, with each agent's action implemented by designing specific prompts for the underlying LLM [17]. Evaluation of MARE demonstrated superior results in requirements modeling compared to three state-of-the-art baselines, showing an improvement in F1 score by up to 15.4% [17].

Another significant framework is Knowledge-Guided Multi-Agent Framework (KGMAF), which has been envisioned to automate the requirements development process through a knowledge-guided approach [15]. KGMAF is composed of six specialized agents (interviewer, end-user, analyst, archivist, reviewer, deployer) and an artifacts pool inspired by the blackboard mechanism [15]. The artifacts pool is designed to store intermediate requirements artifacts such as Original Idea, Product Goals, and Software Requirements Specification (SRS), and its state dictates the workflow and collaboration among agents [15]. Agents continuously monitor the artifacts pool and plan actions based on its status, with each agent equipped with specific functionality, predefined actions, and knowledge to guide its behavior [15]. For instance, an interviewer agent might conduct research and interview end-users to gather requirements, while an end-user agent responds to questions and provides raw requirement descriptions [15]. This blackboard-inspired architecture allows for flexible coordination and knowledge sharing across the multi-stage requirements development process [15].

The Goal2Story framework specifically targets requirements elicitation by using a multi-agent fleet based on privately enabled small LLMs for impact mapping [44]. It utilizes agents such as Delivery Coordinator, Tactical Officer, and Format Doctor, working with data structures that encompass goals, actors, impacts, and deliverables to generate user stories [44]. These frameworks emphasise the necessity of structured collaboration, specialized roles, and persistent knowledge management via shared artifacts or pools to reliably handle complex, multi-stage RE workflows [17, 15]. The success of MARE, KGMAF, and Goal2Story demonstrates that MASs can effectively automate general RE tasks when appropriate architectural patterns and agent specializations are defined. However, these frameworks focus primarily on functional requirements and general RE processes, leaving security-specific requirements engineering largely unaddressed.

### 3.2.3 Multi-Agent Systems for Security Tasks

Beyond general software engineering, MASs architectures are increasingly being applied to specialized security and vulnerability tasks, demonstrating the potential for agent-based approaches to address complex security challenges. However, current security-focused MASs primarily operate at the code or vulnerability level rather than at the requirements level, representing a distinct gap from the SRE focus of this thesis.

The CVE-GENIE framework exemplifies this trend, designed for the automated reproduction of vulnerabilities [34]. CVE-GENIE employs collaborating agents, including the Data Processor, which extracts vulnerable version information; the Pre-Requisite Developer, which explores the codebase and defines the expected state; and the Setup Developer, which configures the vulnerable environment [34]. This division of labour is necessary because processing large codebases can exceed context limits and reduce efficiency in standalone LLMs [34]. By distributing these tasks across specialized agents, CVE-GENIE is able to gather resources, reconstruct vulnerable environments, and produce verifiable exploits for real-world vulnerabilities documented in Common Vulnerabilities and Exposures (CVE) entries [34]. The framework demonstrates that MASs can handle complex, multi-step security workflows that require coordination among agents with different specializations.

Similarly, the AutoSafeCoder framework uses multiple agents to secure LLM code generation outputs by applying static analysis and fuzz testing [27]. AutoSafeCoder divides the code generation process into iterative phases involving a Coding Agent, a Static Analyser Agent, and a Fuzzing Agent, continuously reviewing and testing code for vulnerabilities [27]. This iterative refinement through specialized security analysis agents has achieved vulnerability reduction by integrating static and dynamic analysis directly into the code generation workflow [27]. The success of AutoSafeCoder in reducing security vulnerabilities in generated code highlights the value of multi-agent collaboration for security-critical tasks, where different agents contribute complementary security checks and refinements [27].

Multi-agent frameworks are also being explored for automating threat modeling, where specialized agents collaborate with non-ML symbolic components to decompose security tasks, such as vulnerability detection, triage, and remediation [9, 41]. These emerging approaches suggest that MASs can be effective for security tasks that require coordination among agents with different security expertise. However, a critical observation is that existing security-focused MASs like CVE-GENIE and AutoSafeCoder operate primarily at the implementation level rather than at the requirements level. This distinction is significant: while these frameworks demonstrate that MASs can handle complex security workflows, they do not address the upstream challenge of translating high-level security requirements into actionable, implementation-ready security specifications.

## 3.3 Security Requirements Engineering Methodologies

RE is a pivotal phase in software development, focusing on crafting a well-defined software requirements specification from initial rough ideas [15, 5]. As modern software complexity and system interconnection increase, the process mandates alignment with standards, infrastructure specifications, and customer expectations [5]. Security, specifically, is recognized as an essential non-functional requirement that has historically proven difficult to address [35]. Many traditional software security practices rely on manual, error-prone processes that are heavily dependent on practitioner experience [5]. This section reviews traditional SRE methodologies and recent efforts to automate threat modeling, highlighting their capabilities and limitations.

A systematic mapping study identified that conventional approaches to SRE in agile contexts typically involve either modifying existing agile methods, introducing new artifacts such as extending the standard user story format to an abuser story, or providing specific guidelines [35]. However, this review, covering publications from 2005 to 2017, concluded that significant effort is still needed to empirically evaluate these approaches and mitigate limitations related to effort, resources, people, and the environment [35]. Several structured, threat-oriented methodologies have been proposed to address the security deficit in requirements gathering.

The STORE methodology is presented as a comprehensive approach for security requirements engineering [2]. STORE is a ten-step sequential process designed for effective, efficient, and systematic elicitation and documentation of security requirements for software and web-based applications from the early stages of development [2]. STORE considers the security interests of all stakeholders, identifies threat agents, and performs risk analysis, emphasising threats in the context of security requirements [2]. It aims to overcome limitations of other approaches by systematically identifying and prioritising potential threats to elicit the most appropriate security requirements [2]. This methodology makes eliciting security requirements more systematic for the requirement engineer, providing a structured workflow that guides practitioners through the SRE process [2].

The Identification of Security Threats during Requirement Engineering (ISTDRE) technique aims to efficiently identify security threats during the requirement engineering process, helping requirement engineers produce software that resists exploitation [33]. This technique was proposed to address the fact that many requirement engineers are unaware of the SRE process, leading to ambiguous information and inadequate security coverage [33]. In an examination comparing it with the Model Oriented Security Requirements Engineering (MOSRE) approach, it was found that the results obtained from ISTDRE were better, demonstrating its effectiveness in supporting requirement engineers who lack deep security

expertise [33]. The STRIDE framework, encompassing the threat categories of Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege, is commonly utilized for threat modeling and eliciting security requirements, particularly in systems engineering contexts like Cyber-Physical Systemss (CPSs) [30, 41]. The objective of STRIDE-based methodologies is to identify threats and mitigation strategies based on an input description, such as a design document of a system [41]. Threat modeling, a crucial part of SRE, is traditionally embedded in the early stages of the secure development life cycle, specifically during requirements and architectural design, to identify and mitigate potential security risks before they become costly to address [18].

In response to the manual burden of SRE, automated threat modeling tools have emerged for integration into CI/CD pipelines, enabling a DevSecOps approach [18]. Tools such as `pytm`, Threagile, and TicTaaC have been evaluated for CI/CD readiness based on external readability of artifacts and invocation capabilities [18]. The `pytm` framework utilizes Python-encoded system models and supports export to JSON, SQLite, and Markdown formats, featuring a customizable JSON threat library comprising 103 threat types [18]. Threagile performs threat analysis based on YAML input files representing models and countermeasures, supporting multiple export formats [18]. TicTaaC (Threat modeling-as-a-Code) uses YAML files for Data Flow Diagram (DFD) specification and threat libraries, and critically includes a parameter to trigger build failures when threats exceed a specified risk level, directly enforcing security checks in the pipeline [18].

These tools demonstrate the transition towards automating threat identification and mitigation analysis; however, they often rely on predefined catalogs and external system descriptions, such as DFDs, necessitating manual effort to transform high-level requirements into the required structured input format [18]. Furthermore, the ThreatModeling-LLM framework has been proposed to leverage LLMs for automating threat modeling [41]. This framework transforms banking system design documents into identified threats categorized by STRIDE and suggests mitigation strategies [41]. However, this is not without challenges, as LLMs may struggle with domain-specific knowledge and can suffer from hallucinations, requiring careful validation [10, 26, 41]. Despite these advances, a significant gap remains: while traditional methodologies like STORE and ISTDRE provide systematic processes, they are largely manual; and while automated tools like `pytm`, Threagile, and ThreatModeling-LLM offer partial automation, they require structured inputs or focus on specific phases of threat modeling rather than the end-to-end translation of high-level security requirements into detailed, actionable specifications.

## 3.4 Research Gaps and Limitations

The comprehensive review of related work reveals a clear landscape of research surrounding SRE, LLMs in RE, and MASs architectures. However, several critical intersections remain largely unexplored, revealing gaps and limitations that this thesis aims to address.

While existing research has achieved notable progress in LLM-based RE, these approaches have primarily focused on general functional requirements, with limited exploration of security-specific requirements [3, 26]. The challenges of domain ignorance, hallucinations, and transferability issues are particularly pronounced when LLMs are applied to specialized domains, such as security [10, 24]. Studies have shown that LLMs struggle with deep-seated domain nuances and lack contextual understanding, necessitating human oversight and hybrid validation approaches [10, 3]. This limitation highlights a fundamental gap: no systematic approach exists for utilizing LLMs to translate high-level security requirements into actionable, implementation-ready specifications that align with established security standards and frameworks.

MASs for software engineering, while effective for general RE automation, are designed with functional requirements in mind rather than security-centric workflows [17, 15]. Frameworks like MARE and KGMAF do not incorporate security domain knowledge, threat modeling methodologies, or mappings to security standards in their architectures [17, 15]. Security-focused MASs such as AutoSafeCoder and CVE-GENIE operate at the code or vulnerability level, addressing secure code generation and vulnerability reproduction but not the upstream challenge of requirements translation [27, 34]. These limitations reveal a critical gap: there exists no empirically validated MAS designed specifically to bridge the gap between abstract, high-level security requirements and detailed, security-compliant specifications derived from established, threat-oriented methodologies like STORE and STRIDE [2, 41, 14]. While MASs frameworks like MARE and KGMAF successfully automate the general RE workflow [17, 15], and traditional SRE methodologies like STORE and ISTDRE exist [2, 33], none of these approaches addresses the critical translation task that systematically transforms high-level security needs into actionable, implementation-ready security specifications. This missing link requires a solution that performs automated translation and integrates security knowledge bases seamlessly, leveraging both the collaborative strengths of MASs and the domain expertise encoded in SRE methodologies. No architecture currently exists that combines the collaborative strengths of MASs with the security domain expertise required for systematic SRE.

Traditional SRE methodologies like STORE and ISTDRE provide comprehensive, systematic processes but remain largely manual and time-consuming [2, 33, 35]. Automated threat modeling tools offer partial automation but require structured inputs such as DFDs or Python-encoded models, necessitating

significant manual effort to prepare these inputs from high-level requirements [18]. The ThreatModeling-LLM framework demonstrates potential for leveraging LLMs in threat modeling but still focuses on a specific phase of the SRE process and requires careful validation due to hallucination risks [41]. None of these approaches provides end-to-end automation from high-level security requirements to detailed, actionable specifications that are directly usable for implementation. This limitation underscores a further critical gap: a clear understanding of the architectural patterns and collaborative mechanisms essential for achieving robust and systematic SRE within an LLM-based MAS framework is lacking. The success of existing MAS architectures relies on explicitly defined agent roles and communication mechanisms, such as artifact pools in KGMAF [15] and shared workspaces in MARE [17]. However, the literature has not yet defined the necessary architectural patterns or collaborative mechanisms required for a MAS to execute security-centric tasks that involve threat identification, standard cross-referencing, and mitigation strategy generation [35]. Specifically, research needs to define the agent roles, their interaction protocols, and the structured knowledge base required to implement methodologies like STORE and STRIDE effectively within an end-to-end automation framework. Without clear architectural guidance, building a secure and effective MAS for SRE remains ad-hoc, and researchers and practitioners lack systematic patterns to follow.

While security challenges in LLM-based MASs have been identified and defense mechanisms proposed, the research concentrates on code generation and general LLM applications rather than SRE contexts [1, 9]. Prompt leakage, a critical threat where adversarial prompts can leak backend API calls, implementation details, and the overall system architecture, presents particular concern for SRE applications, where system prompts may encode sensitive methodologies, threat models, or proprietary security knowledge [1]. When agents interact with private information or competing objectives, they can exhibit emergent behaviors such as covert collusion or coordinated attacks [9]. Studies have shown that LLMs can covertly exchange messages via steganographic abilities, often undetectable by overseers [9]. Furthermore, adversaries can exploit combinations of ostensibly safe models to bypass security safeguards, achieving malicious goals that individual models would refuse [9]. False information, once injected through prompt infection, can persist and amplify across retrieval-augmented group chats, self-replicating as compromized agents automatically forward malicious instructions [9]. These infectious adversarial attacks demonstrate how vulnerabilities can propagate through a MAS, affecting multiple agents and potentially compromising the entire system [9].

Addressing these unique multi-agent security challenges requires sophisticated mechanisms. Query-rewriting, where an LLM is used to rewrite user prompts before they reach the main agent, filters out potential attack patterns and successfully drops attack prompts from user inputs in studies assessing prompt leakage [1]. LLM vaccination is a proactive defence strategy that involves seeding

agents' memories with examples of safely handling malicious prompts [9]. In simulated chemical research environments, LLM vaccination substantially curbed multi-hop jailbreak spread while preserving collaborative efficacy, outperforming instruction-only safeguards in maintaining both security and helpfulness [9]. Prompt encryption techniques, such as Emojicrypt, are being explored to secure LLM applications against unauthorized actions [6]. Hierarchical information management and memory protection classify information by security levels to restrict sensitive data access to authorized agents, thereby mitigating risks of unauthorized disclosure [20]. However, no study has systematically evaluated the security vulnerabilities and appropriate defences for MASs when applied specifically to the task of security requirements engineering, where sensitive threat models, security controls, and organizational security policies are processed and must be protected. This represents a significant gap: the need for rigorous, comparative evaluation of a MAS approach against traditional and contemporary AI-augmented approaches in terms of efficiency, accuracy, and scalability when performing security requirements translation, coupled with empirical assessment of security vulnerabilities inherent in using LLM-integrated MASs for sensitive SRE tasks. Practical security vulnerabilities such as prompt leakage of security methodologies or adversarial attacks targeting threat models require empirical assessment and corresponding mitigation strategies that ensure the system is not only accurate but also trustworthy and robust [16, 8]. Without such systematic evaluation, the practical deployment of LLM-based MASs for security-critical applications like SRE remains unvalidated.

The gaps identified across LLM capabilities in security domains, the absence of security-centric MAS architectures, the manual burden of traditional SRE approaches, and the limited security evaluation of MASs in SRE contexts collectively define the motivation for this thesis. By proposing, developing, and empirically evaluating a specialized MAS designed specifically for automated, security-compliant requirements translation, this research seeks to bridge these gaps, advance the state of the art in automated SRE, and provide foundational knowledge for future research in LLM-based MASs for security-critical applications.

# Chapter 4

# Research Methodology

This chapter outlines the research methodology employed to address the research questions and achieve the objectives of this thesis. The chapter begins by presenting the research questions that guide this study, followed by a detailed justification of the selected research methodology. The DSRP was chosen as the primary research approach due to its emphasis on developing and evaluating technological artifacts whilst maintaining collaboration with industry practitioners. The chapter then describes the research design, organized into three interconnected phases: problem identification and conceptualization, solution design and development, and evaluation and validation. Each phase encompasses specific activities designed to systematically address the research questions and validate the developed multi-agent system for automated security requirements engineering.

## 4.1 Research Questions

In light of the identified problem and the potential for multi-agent systems leveraging LLMs to address it, this thesis seeks to answer the following research questions:

- **RQ1: How effectively can a multi-agent system, leveraging LLMs, translate high-level software requirements into detailed, security-compliant software requirements?** This research question focuses on assessing the core capability of the proposed system to perform the primary task of automated security requirements translation. It investigates whether multi-agent architectures can successfully bridge the gap between business-level requirements and security-compliant technical specifications, and how the quality of these translations compares to manual expert-driven processes.
- **RQ2: What architectural patterns and collaborative mechanisms are essential for a multi-agent system to achieve robust and systematic security requirements engineering?** This research question explores the structural and organizational aspects of multi-agent systems that enable effective security requirements engineering. It aims to identify

the architectural patterns, agent collaboration mechanisms, and system design decisions that are critical for achieving consistency, comprehensiveness, and reliability in the automated requirements translation process.

- **RQ3: How does the proposed multi-agent system perform in terms of efficiency, accuracy, and scalability, and what is its perceived utility in realistic industrial contexts?** This research question addresses the practical performance characteristics of the multi-agent system. It evaluates processing efficiency, the accuracy of identified threats and mapped controls, and scalability across different input sizes and complexity levels. In addition, it assesses perceived utility and adoption intent through expert feedback, providing feasibility evidence for real-world deployment rather than a controlled comparative benchmark against human-led processes or alternative tools.

## 4.2 Research Method Selection

This research employs the DSRP methodology as a systematic approach for developing and evaluating technological artifacts that address real-world problems. DSRP was selected as the primary research methodology due to its strong emphasis on bridging the gap between academic research and industry practice. Given the practical nature of this thesis and its focus on addressing challenges identified through industry collaboration, DSRP provides the necessary framework for both rigorous scientific investigation and the development of practical artifacts.

DSRP operates through iterative cycles of three primary phases: problem identification, solution design, and evaluation. These phases are executed iteratively rather than strictly sequentially, allowing for continuous refinement based on empirical evidence and theoretical insights. This flexibility is particularly valuable for the development of multi-agent systems, where architectural decisions must be informed by both theoretical principles and practical performance metrics.

Alternative methodologies were considered but found less appropriate for this research context. Action Research, whilst valuable for organizational change initiatives, requires deep integration within teams and multiple operational iterations that exceed the scope and timeframe of this thesis. Moreover, Action Research focuses primarily on immediate operational changes rather than the systematic development and empirical evaluation of a technological artifact. The Technology Transfer Methodology, while promoting the application of research findings to industry, lacks mechanisms for researchers to be actively involved in the development process and does not provide the iterative refinement cycles essential for developing complex systems, such as multi-agent architectures. DSRP, by contrast, explicitly supports the creation of novel artifacts through iterative

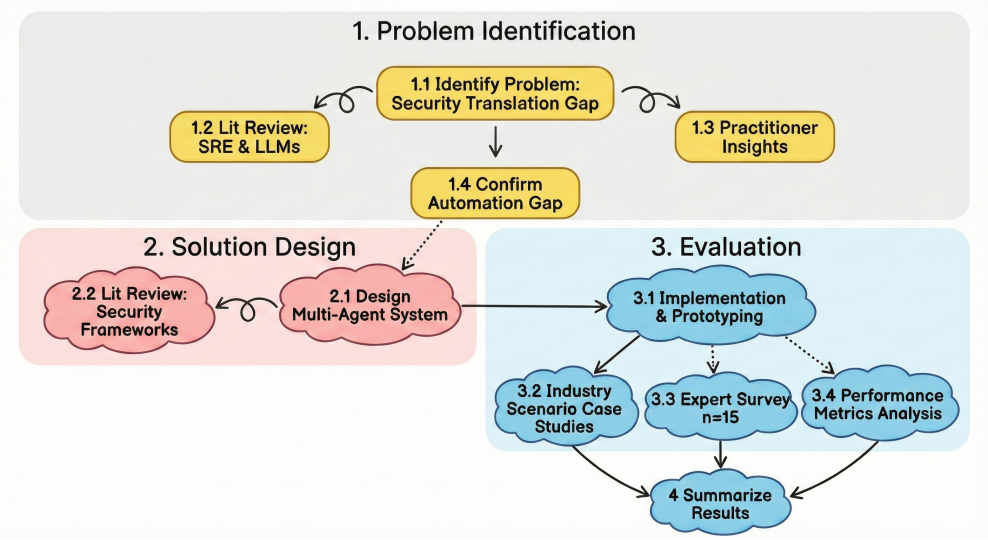


Figure 4.1: Design Science Research Process.

cycles of design and evaluation, making it the most appropriate choice for this research.

### 4.2.1 Design Science Research Process

DSRP revolves around acquiring knowledge through the development and evaluation of artifacts, which is particularly valuable in software engineering research due to its analytical examination of structured constructs. Moreover, it ensures that all developed solutions undergo thorough testing and validation before being considered as contributions to knowledge. As illustrated in Figure 4.1, the DSRP employs both qualitative and quantitative research methods throughout its iterative three-phase process.

*Problem Identification* involves identifying and confirming the importance of the problem through literature reviews and consultations with domain experts. This phase establishes the theoretical foundation and validates that the problem warrants research attention.

*Solution Design* involves developing a solution through a creative engineering process, informed by further literature research and expert knowledge. This phase transforms the identified problem into concrete technological artifacts designed to address specific requirements.

*Evaluation* entails building and empirically evaluating the designed artifacts against pre-defined criteria. This phase assesses the artifact's effectiveness in addressing the problem and contributes insights for both practical deployment and theoretical advancement.

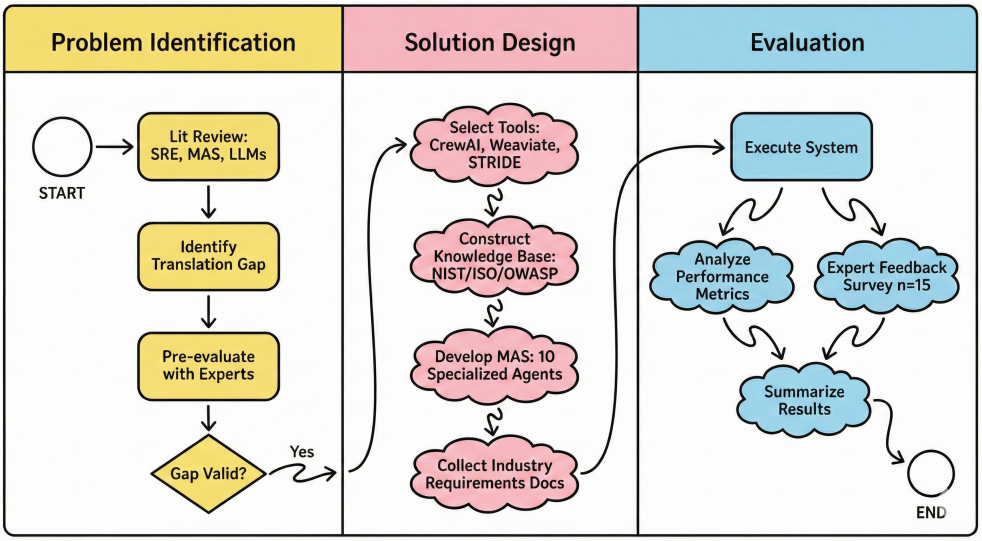


Figure 4.2: Research design process showing three phases and key activities.

## 4.3 Research Design

The research design follows the three-phase DSRP structure, with each phase encompassing specific activities designed to systematically address the research questions. As illustrated in Figure 4.2, the phases are interconnected and iterative, allowing insights from evaluation phases to inform refinements in earlier work. This structure ensures both scientific rigor and practical relevance.

### 4.3.1 Phase 1: Problem Identification and Conceptualization

The problem identification phase establishes the foundation by defining the problem space, understanding existing approaches, and identifying gaps in current knowledge and practice. This phase confirms that current security requirements engineering approaches lack adequate automation for translating high-level business requirements into security-compliant specifications, whilst addressing the challenges that organizations face in systematically capturing and documenting security requirements.

A structured literature review was conducted across multiple domains, including security requirements engineering methodologies, large language models in requirements engineering, and multi-agent systems in software engineering. Multiple academic databases were searched, including IEEE Xplore, ACM Digital Library, Springer Link, and Google Scholar. While these databases were used to establish the state of the art, it is acknowledged as a reporting limitation that the specific search strings and exact date ranges for the initial scoping review were

not formally logged. Selection criteria ensured relevance to security requirements engineering, methodological rigor, and contribution to understanding both the problem space and potential solutions. Key methodologies such as STORE and STRIDE, as well as their limitations in the context of automation, were identified and analyzed.

Beyond the literature review, practitioner insights were gathered through discussions with security professionals to validate the research problem and understand practical challenges in translating business requirements into security specifications. These insights informed the design requirements for the multi-agent system, ensuring that the developed artifact would address real-world needs and be grounded in industry practice rather than purely theoretical concerns.

### 4.3.2 Phase 2: Solution Design and Development

The solution design and development phase translates the identified problem into a working multi-agent system artifact. The design process began with identifying essential system components based on insights from Phase 1, including the selection of appropriate security frameworks (STORE, STRIDE, OWASP ASVS, NIST SP 800-53, ISO 27001) and the specification of the multi-agent system architecture.

The multi-agent system architecture follows a staged workflow approach with four logical phases: requirements analysis, parallel security analysis, synthesis and planning, and validation. Each stage is decomposed into specialized tasks assigned to dedicated agents with domain-specific expertise. A critical architectural decision involved the use of an event-driven state model where agents interact through an immutable centralized state object, ensuring deterministic behavior, facilitating debugging and auditability, and supporting parallel execution of independent tasks.

Implementation followed an iterative development approach using agile methodologies, with each agent developed, tested, and refined individually before integration into the complete system. Design patterns such as Reasoning and Acting (ReAct), Chain-of-Thought prompting, Hierarchical Decomposition, and Self-Critique and Refinement were applied to enhance agent reasoning capabilities. The CrewAI framework[1] was selected as the implementation platform due to its support for agent orchestration, state management, and tool integration. A vector database containing security controls from multiple standards was populated with ground agent recommendations in verified knowledge.

[1]CrewAI: https://docs.crewai.com/en/introduction

### 4.3.3 Phase 3: Evaluation and Validation

The evaluation is designed as a feasibility and expert-utility validation of the proposed artifact in realistic industrial contexts, rather than as a controlled experiment intended to prove superiority over human-led security requirements engineering. The evaluation and validation phase assesses the developed artifact against the research questions using a mixed-methods approach that combines quantitative performance metrics with qualitative expert assessment. The evaluation was structured around a real-world industrial scenario where software engineers provided high-level business requirements for proprietary web applications they had previously worked on.

#### Data Collection Methodology

As the primary data collection mechanism for the validation phase, 15 professional software engineers participated in the evaluation study. Each participant provided high-level requirements documentation for a proprietary web application with which they had professional experience. This yielded a diverse set of 14 unique use cases spanning domains such as e-commerce, healthcare, and financial services. Crucially, these inputs were derived directly from active industry projects rather than being synthetically generated for this study. This approach ensured that the input requirements reflected realistic, messy scenarios encountered in industry practice and were not present in publicly available training datasets. These requirements documents served as input to the development of the multi-agent system, which subsequently generated comprehensive security reports containing identified threats, mapped security controls, compliance requirements, and implementation recommendations.

The security reports generated by the multi-agent system were then distributed to each participant along with a structured questionnaire designed to capture their professional assessment of the tool's outputs and utility. This approach ensured that participants evaluated the system's performance within the context of their own requirements and domain expertise, enhancing the ecological validity of the findings.

It is important to explicitly define the baseline used for establishing correctness in this study. As outlined in the limitations (Section 1.3.1), a static "Gold Standard" baseline (such as a pre-validated answer key) was not utilized, primarily because the study employs novel, proprietary industrial requirements for which no ground truth existed. Consequently, this research adopts an Expert-Based Validation approach. The participating professionals, who possess the deepest contextual understanding of their specific systems, serve as the source of truth. Therefore, "accuracy" and "correctness" in this context are defined by the system's alignment with expert professional judgment and industrial acceptability, rather than a comparison against a synthetic or manual control.

| No. | Question and answer options |
|---|---|
| Q1 | What is your current role? |
| | *Answer:* Security Engineer, Security Engineer, Solution Architect, Tech Lead, Other (Specify) |
| Q2 | How many years of software development experience do you have? |
| | *Answer:* 0-1 year, 1-3 years, 3-5 years, 5-10 years, 10+ years |
| Q3 | I intend to use insights from this security report in my next project |
| | *Answer:* Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree |
| Q4 | I plan to incorporate this report's recommendations into our requirements process in the next quarter |
| | *Answer:* Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree |
| Q5 | I am likely to rely on this report's OWASP/NIST/ISO mappings when defining security requirements |
| | *Answer:* Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree |
| Q6 | I would recommend using reports like this for my team's projects |
| | *Answer:* Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree |
| Q7 | If available, I would request generating such security reports for new initiatives |
| | *Answer:* Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree |
| Q8 | I am likely to allocate dedicated time during requirements elicitation to review this report |
| | *Answer:* Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree |
| Q9 | I intend to continue requesting such reports after this study |
| | *Answer:* Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree |
| Q10 | What single change would most increase your likelihood of using these security reports in your projects? |
| | *Answer:* Open-ended text response |
| Q11 | Are you interested in a follow-up interview? |
| | *Answer:* Yes, No |

Table 4.1: Questionnaire items for expert evaluation

This questionnaire employed a mixed-methods approach combining Likert-scale structured questions and open-ended feedback. The structured questions used a five-point Likert scale (Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree) to assess key dimensions of system effectiveness and utility, whilst open-ended questions captured qualitative insights into participant experiences and suggestions for improvement. Table 4.1 outlines the contents and purpose of each questionnaire item.

The questionnaire design was informed by established quality assessment frameworks for requirements engineering tools and was carefully structured to elicit feedback directly aligned with the research questions. Items Q3 through Q9 collectively assessed intended adoption and utility, addressing RQ1 regarding the effectiveness of the automated translation approach. Additional metrics covered the clarity of security mappings and the practical applicability of generated requirements within real-world development contexts. Q10 captured open-ended suggestions for improvement, providing qualitative insights into tool limitations and enhancement opportunities.

### Participant Demographics

The evaluation study recruited 15 participants (2 of which reviewed the same questionnaire due to time constraints) with diverse professional backgrounds and experience levels. Figure 4.3 presents the distribution of participants by professional role. The majority of participants (46.7%) were Software Engineers (7 participants), followed by Tech Leads (13.3%, 2 participants) and Solution Architects (13.3%, 2 participants). The remaining participants represented specialized roles: Researcher (6.7%), DevOps (6.7%), Cloud Field Engineer (6.7%), and FPGA Engineer (6.7%), each with a single representative. This distribution reflects a strong engineering focus with significant technical depth, though the sample includes only one participant with formal security engineering background, highlighting the practical relevance of automated security requirements tools for general engineering practitioners.

Professional experience distribution is presented in Figure 4.4. The participant pool exhibited substantial experience depth, with 33.3% (5 participants) reporting 5-10 years of software development experience, followed by 26.7% (4 participants) with more than 10 years of experience. The remaining participants were evenly distributed between 1-3 years (20.0%, 3 participants) and 3-5 years (20.0%, 3 participants) of experience. The median experience level was 5-10 years, indicating that the majority of evaluators possessed mid-to-senior level expertise and could provide informed assessments of automated security requirements tools.

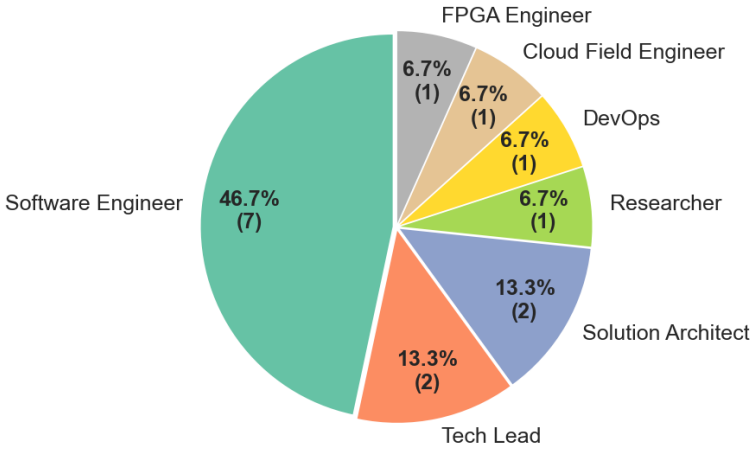


Figure 4.3: Participant Professional Roles.

### Analysis of Evaluation Data

Quantitative analysis of Likert-scale responses involved calculating descriptive statistics, including mean ratings, standard deviations, and frequency distributions for each item. These metrics provided insight into the level of professional consensus regarding the system's utility and effectiveness. Qualitative analysis of open-ended responses employed thematic coding to identify recurring themes, suggestions, and concerns raised by participants.

The evaluation metrics addressed all three research questions: Q3-Q7 and Q9 directly relate to RQ1 (effectiveness of requirements translation), with a focus on whether participants would adopt the system and rely on its mappings in practice. Demographic data (Q1-Q2) allowed for an analysis of whether feedback varied by professional role or experience level. Q10 captured specific improvement suggestions that informed our understanding of current limitations and future directions, aligning with RQ2 and RQ3.

## 4.4 Threats to Validity

This section discusses threats to the validity of the evaluation and how they were mitigated.

### Construct validity

The study evaluates "quality" and "utility" primarily through expert judgments captured via a structured questionnaire, which may not perfectly measure real-world security effectiveness or implementation outcomes. To mitigate this, the questionnaire items were designed to reflect practical adoption-oriented signals

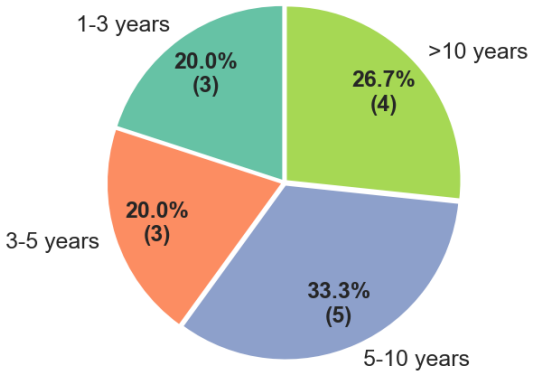


Figure 4.4: Participant Years of Software Development Experience.

(e.g., likelihood to use recommendations and mappings) and were complemented with open-ended feedback to capture nuanced concerns not represented in Likert items.

### Internal validity

Several factors could have influenced participants' assessments, including differences in individual security maturity, familiarity with standards, and varying expectations of what constitutes "implementable" security requirements. The study reduces this risk by asking each participant to evaluate reports generated from their own requirements context, where they have the strongest domain understanding, and by explicitly treating experts' professional judgment as the primary validation reference rather than assuming an external "ground truth".

### External validity

While the evaluation uses proprietary industrial requirements across multiple domains, the participant pool is relatively small and may not represent all organizational contexts (e.g., highly regulated environments or teams with dedicated security architects). The results should therefore be interpreted as evidence of feasibility and perceived utility rather than universal generalization of performance.

### Reliability and reproducibility

LLM-based systems can produce non-deterministic outputs across runs, which affects reproducibility. This is partially mitigated through configuration controls (e.g., fixed settings where possible) and by proposing operational mitigations such

as freezing model versions or using deterministic/local models in future deployments. Additionally, because inter-rater agreement was not computed, the results emphasize practitioner utility rather than statistical agreement on wording-level correctness

# Chapter 5

# Solution Design and Implementation

This chapter presents the design and implementation of a multi-agent system that automates the translation of high-level software requirements into detailed, security-compliant specifications. The system orchestrates ten specialized LLM-powered agents, each one configured for specific complexity levels. The source code of this system as well as the generated reports are hosted on the same public GitHub repository[1].

This system addresses the fundamental challenge of security requirements engineering by decomposing the complex, multi-faceted task into specialized subproblems, each handled by a dedicated agent with domain-specific expertise. The system architecture is organized into four logical stages that reflect the natural progression from requirements analysis through security specification generation, as illustrated in Figure 5.1 and detailed in the following list.

- **Stage 1** consists of the Requirements Analysis Agent, which processes high-level requirements documents and extracts security-relevant features, architectural components, and system boundaries through two sequential subtasks: requirements extraction followed by architectural modeling.
- **Stage 2** executes five specialized agents in parallel: the Stakeholder Analysis Agent identifies user roles and trust boundaries; the Threat Modeling Agent applies STRIDE methodology to generate threats; the Domain Security Agent queries the vector database to map requirements to security controls from OWASP ASVS, NIST SP 800-53, and ISO 27001; the LLM Security Specialist Agent detects AI/ML components and applies specialized controls when needed; and the Compliance Agent identifies applicable regulatory frameworks.
- **Stage 3** synthesizes all findings through three sequential agents: the Security Architecture Agent aggregates outputs into a unified security architecture design; the Implementation Roadmap Agent creates a prioritized, phased implementation plan; and the Verification and Testing Agent designs comprehensive security testing strategies.

[1]GitHub repository: https://github.com/SavvasMohito/mas-sre

- **Stage 4** performs validation and output generation: the Validation Agent assesses quality across five dimensions (completeness, consistency, correctness, implementability, alignment) using a 0.7 threshold to ensure a high-quality output; if validation passes, the system constructs a traceability matrix and generates a comprehensive security requirements report. A critical validation loop enables iterative refinement: if validation fails and fewer than three iterations have occurred, the system loops back to Stage 1 with structured feedback, enabling self-improvement without manual intervention.

This staged architecture with strategic parallelization achieves approximately a 40% reduction in execution time compared to sequential execution. It is important to distinguish this architectural efficiency from the broader business value: while the parallel design saves minutes of computational time, the system's ability to generate comprehensive security drafts in under 15 minutes represents a substantial reduction in effort relative to fully manual security requirements drafting, although human-vs-tool time savings were not measured in this study.

Agent communication follows an event-driven state model implemented through CrewAI decorators (`@start`, `@listen`) that define execution triggers and data dependencies. Agents do not directly communicate with each other; instead, all outputs flow forward through an immutable, centralized `State` object that serves as the single source of truth throughout the analysis pipeline. This uni-directional, state-mediated communication pattern provides several critical advantages: it establishes clear execution paths with explicit dependencies, prevents circular dependencies that could cause deadlocks or infinite loops, enables transparent data flow that facilitates debugging and auditability, and supports parallel execution by allowing multiple agents to read from the same state without conflicts.

The immutable state model ensures that each agent receives a consistent snapshot of all previous analyses, eliminating race conditions and ensuring deterministic behavior. When agents need to access outputs from previous phases, they read from the current state object, which contains all accumulated analysis results up to that point in the workflow. This architectural pattern is particularly well-suited for security-critical applications where traceability and auditability are essential requirements.

The ten specialized agents that constitute this system each address a specific aspect of security requirements engineering, from initial requirements parsing through threat identification, control mapping, compliance analysis, architectural synthesis, and validation. Each agent is configured with domain-specific expertise, appropriate model selection based on task complexity, and design patterns optimized for its particular responsibilities.

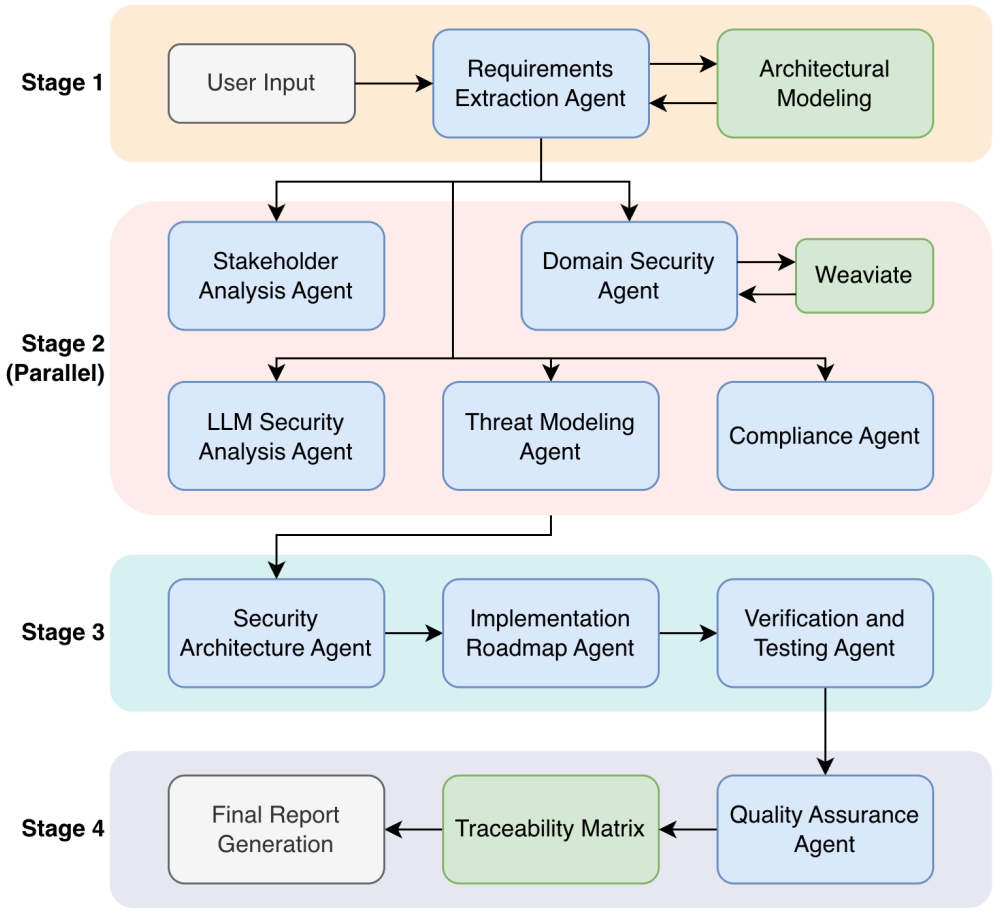


Figure 5.1: Multi-Agent System Architecture with Ten Specialized Agents

## 5.1 Agent Design and Configuration

The multi-agent architecture decomposes the complex security requirements engineering task into ten specialized sub-problems, each handled by a dedicated agent with domain-specific expertise. This decomposition addresses fundamental challenges: the breadth of knowledge spans multiple domains, different tasks require distinct reasoning patterns, and parallel execution significantly reduces processing time. By assigning each agent a focused responsibility, the system achieves both depth of analysis and operational efficiency.

Unlike general-purpose RE frameworks such as MARE [17] which utilize generic roles (e.g., 'Modeler', 'Checker'), this architecture explicitly adopts domain-specific security roles (e.g., 'Threat Modeling Agent', 'Compliance Agent'). This design choice embeds deep security domain expertise directly into the agent persona, reducing the cognitive load required to bridge the translation gap.

The system employs a tiered model selection strategy using OpenAI GPT-5 model variants to balance reasoning capability with operational cost. Different

analysis tasks exhibit varying complexity demands: some require sophisticated multi-step reasoning, tool use, and knowledge retrieval augmentation, while others involve structured operations such as pattern recognition, rule application, or evaluation against defined criteria.

GPT-5-mini was selected for high-complexity tasks (Threat Modeling, Domain Security) to leverage its superior reasoning capabilities for abstract threat extraction. Conversely, GPT-5-nano was utilized for structured tasks (Roadmap, Compliance) where the primary challenge is data formatting rather than complex deduction. This tiered strategy optimizes the cost-performance ratio without compromising the depth of security analysis.

It is important to note that the proposed architecture is model-agnostic and not inherently tied to OpenAI models. The selection of GPT-5 variants was driven by practical considerations: an existing API subscription facilitated rapid development, and the recently released GPT-5 models offer competitive performance at reduced operational costs. The system enforces structured outputs through Pydantic schemas that validate agent responses against predefined data structures, raising validation errors when outputs deviate from expected formats. This design ensures that any sufficiently capable language model from alternative providers such as Anthropic or Google could be substituted, provided it adheres to the specified instructions and output schemas. Open-source models were considered but excluded from the current implementation due to computational resource constraints required for hosting models of sufficient capability; however, the architecture remains theoretically compatible with self-hosted alternatives.

The configuration of each agent is presented below using structured agent cards[2], which systematically document their functional characteristics and technical configuration. Each agent card follows a consistent format, detailing the agent name, purpose, inputs, outputs, model, and design pattern to provide a complete understanding of the agent's responsibilities and operational parameters.

**Requirements Analysis Agent**

- **Purpose**: Parse and analyze product manager requirements, extracting security-relevant features and architectural components
- **Inputs**: High level requirements document (text)
- **Outputs**: Application summary, high-level requirements, detailed requirements with IDs and sensitivity levels, system architecture diagram (Mermaid), security context, trust boundaries, attack surface analysis
- **Model**: GPT-5-mini

[2]Agent cards: https://www.agentcard.net/

- **Design Pattern**: Sequential multi-role (Requirements Analyst → System Architect). Two specialized sub-agents execute sequentially, with the Requirements Analyst extracting and structuring requirements, then the System Architect building upon this analysis to model the system architecture. This decomposition enables focused expertise while ensuring cohesive output.

**Stakeholder Analysis Agent**

- **Purpose**: Identify system stakeholders, user personas, privilege levels, and trust boundaries for access control requirements
- **Inputs**: Requirements analysis output
- **Outputs**: User roles and personas, privilege levels, trust levels, security concerns per role, trust model description
- **Model**: GPT-5-nano
- **Design Pattern**: Single-agent analysis. The agent applies domain expertise directly, performing pattern recognition and role categorization without requiring sub-specialization.

**Threat Modeling Agent**

- **Purpose**: Perform comprehensive threat modeling using STRIDE methodology to identify and prioritize security threats
- **Inputs**: System architecture, components, data flows, trust boundaries
- **Outputs**: List of identified threats with threat ID, affected component, STRIDE category, description, likelihood (High/Medium/Low), impact (High/Medium/Low), risk level, and mitigation strategy
- **Model**: GPT-5-mini
- **Design Pattern**: Systematic enumeration. The agent applies STRIDE methodology systematically across all identified attack surfaces, ensuring comprehensive coverage of threat categories.

**Domain Security Agent**

- **Purpose**: Map requirements and threats to specific security controls from OWASP ASVS, NIST SP 800-53, and ISO 27001 using grounded, verified knowledge
- **Inputs**: Detailed requirements, identified threats

- **Outputs**: 3-5 security controls per requirement from multiple standards, each with standard name, control ID, chapter/section, requirement text, relevance explanation, integration guidance, priority, and verification method
- **Model**: GPT-5-mini
- **Design Pattern**: Retrieval-augmented generation (RAG) with tool-augmented reasoning. The agent queries a Weaviate[3] vector database containing 800+ controls from three standards via semantic search, grounding all recommendations in actual standards rather than relying solely on parametric knowledge. This prevents hallucinations and ensures factual accuracy of control recommendations.

**LLM Security Specialist Agent**

- **Purpose**: Detect AI/ML components in system description and add specialized security controls addressing LLM-specific threats
- **Inputs**: System requirements and architecture
- **Outputs**: AI/ML usage indicators, LLM-specific controls addressing prompt injection, data leakage, model inversion, adversarial inputs, model poisoning, supply chain vulnerabilities, and insecure output handling
- **Model**: GPT-5-nano
- **Design Pattern**: Conditional specialization. The agent first determines whether AI/ML components exist and then conditionally applies specialized security analysis only when necessary.

**Compliance Agent**

- **Purpose**: Identify applicable regulatory requirements and ensure security controls align with compliance obligations
- **Inputs**: System requirements, security context, data handling details
- **Outputs**: Applicable regulations (GDPR, HIPAA, PCI-DSS, SOX, CCPA), compliance triggers, mapping of controls to regulatory requirements, data protection obligations, audit and documentation needs
- **Model**: GPT-5-nano

[3]Weaviate: https://docs.weaviate.io/weaviate

- **Design Pattern**: Rule-based classification. The agent applies regulatory knowledge as classification rules (e.g., "if system handles health data, then HIPAA applies") to determine applicable frameworks, then maps controls to requirements.

**Security Architecture Agent**

- **Purpose**: Synthesize all analyses into a comprehensive security architecture with defensive patterns and data protection strategies
- **Inputs**: All previous analysis outputs (requirements, threats, controls, compliance)
- **Outputs**: Architectural principles, component-level controls, data protection strategy (classification, encryption, retention), third-party integration security requirements
- **Model**: GPT-5-nano
- **Design Pattern**: Synthesis and integration. The agent aggregates multi-source inputs into a unified architectural vision, applying security-by-design principles across all layers.

**Implementation Roadmap Agent**

- **Purpose**: Create a prioritized, phased implementation plan based on risk, compliance deadlines, and resource constraints
- **Inputs**: Security controls, threat risk levels, compliance requirements
- **Outputs**: Phased implementation plan with phases (0-3 months immediate, 3-6 months short-term, 6-12 months medium-term, 12+ months long-term), timeline estimates, prioritization rationale, dependencies, and resource requirements
- **Model**: GPT-5-nano
- **Design Pattern**: Multi-criteria prioritization. The agent applies multiple criteria simultaneously (risk, compliance, dependencies, resources) to create a practical sequence balancing security urgency with organizational constraints.

**Verification and Testing Agent**

- **Purpose**: Design comprehensive security testing and verification strategies
- **Inputs**: All security controls and implementation roadmap
- **Outputs**: Testing methods (SAST, DAST, IAST, penetration testing, compliance auditing, continuous monitoring), for each method: description, applicable controls, frequency, recommended tools, and KPIs
- **Model**: GPT-5-nano
- **Design Pattern**: Multi-method verification. The agent designs a layered strategy applying different verification methods (static, dynamic, manual) at different lifecycle stages.

**Validation Agent**

- **Purpose**: Validate completeness, consistency, correctness, implementability, and alignment of all generated security requirements
- **Inputs**: All outputs from previous agents
- **Outputs**: Overall quality score (0.0-1.0), dimension scores (completeness, consistency, correctness, implementability, alignment), validation status (passed/-failed at 0.7 threshold), structured feedback with strengths, gaps, and recommendations
- **Model**: GPT-5-nano
- **Design Pattern**: Self-evaluation and feedback loop. This agent implements quality assurance with optional iterative refinement. If validation fails (score $< 0.7$) and fewer than 3 iterations have occurred, feedback is injected into the next Requirements Analysis execution, enabling self-improvement.

### 5.1.1 Design Patterns Applied

The ten agents collectively demonstrate key multi-agent design patterns that enhance reasoning capabilities and system effectiveness. The ReAct pattern interleaves reasoning traces with task-specific actions, allowing language models to reason about what actions to take, execute those actions to gather information, and then reason about the results [42]. In our system, the Domain Security Agent implements ReAct by reasoning about which security controls are needed, acting through semantic queries to the Weaviate vector database to retrieve relevant controls from OWASP ASVS, NIST SP 800-53, and ISO 27001, and then

reasoning about the retrieved results to select and map the most appropriate controls. This iterative reasoning-acting cycle ensures that control recommendations are grounded in verified standards rather than relying solely on parametric knowledge, preventing hallucinations and ensuring factual accuracy.

**Chain-of-Thought** prompting improves complex reasoning by generating a series of intermediate reasoning steps before arriving at a final answer [40]. This technique elicits step-by-step reasoning that emerges naturally in sufficiently large language models when provided with chain-of-thought exemplars in the prompt. Our system applies Chain-of-Thought prompting in complex agents that require multi-step reasoning, particularly the Requirements Analysis Agent, Threat Modeling Agent, and Domain Security Agent. For instance, the Threat Modeling Agent uses Chain-of-Thought to systematically reason through each STRIDE category, identify attack surfaces, assess likelihood and impact, and prioritize threats, resulting in more comprehensive and accurate threat identification compared to direct answer generation.

**Hierarchical Decomposition** addresses complex tasks by breaking them into specialized sub-tasks that can be handled more effectively than attempting to solve the entire problem monolithically [36]. Our multi-agent system employs this pattern by breaking down the complex SRE task into ten specialized subtasks, each handled by a dedicated agent with domain-specific expertise. For example, the Requirements Analysis Agent decomposes its task into sequential sub-roles (Requirements Analyst $\rightarrow$ System Architect), while the overall system decomposes SRE into parallel analysis phases followed by specialized analysis and synthesis phases. This hierarchical approach enables each agent to focus on its specialized domain while contributing to the overall process of generating security requirements.

**Self-Critique and Refinement** enables models to improve the faithfulness of their explanations through an iterative critique and refinement process without external supervision [38]. In our system, the Validation Agent implements this pattern through a feedback loop that assesses the quality of all generated security requirements across five dimensions: completeness, consistency, correctness, implementability, and alignment. When validation fails (quality score below the 0.7 threshold) and fewer than three iterations have occurred, the agent generates structured feedback that identifies strengths, gaps, and recommendations. This feedback is then injected into the next Requirements Analysis execution, enabling the system to refine and improve its outputs iteratively. This validation loop was explicitly introduced as a design iteration after early testing revealed that open-loop agents occasionally hallucinated controls or omitted obvious threats. The self-critique mechanism serves as a necessary consistency check to catch these anomalies before final output generation.

## 5.2 Running Example: E-Commerce Platform

This section demonstrates the system's operation through a concrete example: an e-commerce platform with comprehensive requirements covering user management, product catalog, shopping cart and checkout functionality, secure payment processing, administrative capabilities, and AI-powered features. The input requirements document describes a platform that must support user registration and authentication with email and password as well as social login integration, product browsing and search with customer reviews, secure checkout with multiple payment methods including credit card, PayPal, and Apple Pay, guest checkout, refund and chargeback handling, and administrative functions for managing inventory, orders, and sales analytics. The platform also incorporates AI features such as product recommendations, a customer support chatbot, and automated product categorization. This example illustrates how the multi-agent system processes a realistic, moderately complex requirements document and generates comprehensive security requirements through its four-stage workflow. The complete input document is provided in Appendix A.

### 5.2.1 Stage 1: Requirements Analysis

The Requirements Analysis Agent processes the input document, performing two sequential sub-tasks: requirements extraction followed by architectural modeling. For this e-commerce example, the agent produces an application summary describing the platform's purpose and scope, extracts 32 high-level security-relevant requirements covering authentication, data protection, payment processing, and administrative functions, and generates detailed requirements with unique identifiers and security sensitivity classifications. In this case, the functional areas of the input document resulted in 32 security-relevant requirements, shown in Table 5.1.

The agent then constructs a system architecture diagram in Mermaid[4] format, identifying six major architectural layers: Frontend, Edge, Application Services, Data Layer, External Services, and Operations. The architectural decomposition reflects the specific system design inferred from the requirements; different applications would yield different architectural structures. The analysis also defines trust boundaries with associated security controls, identifies system components with their security criticality levels, and performs an attack surface analysis, identifying primary entry points and associated threats. The number and nature of trust boundaries depend on the system's architecture and integration points. Figure 5.2 shows the generated system architecture diagram for this e-commerce platform.

[4]Mermaid: https://docs.mermaidchart.com/mermaid-oss/intro/index.html

Table 5.1: High-level requirements extracted from the e-commerce platform input document.

| # | **Requirement** | **Sensitivity** |
|---|---|---|
| 1 | User registration and login with email/password and optional multi-factor authentication | High |
| 2 | Social login via Google and Facebook | Medium |
| 3 | User profiles containing shipping addresses, saved payment method tokens, and contact preferences | High |
| 4 | Order history tracking and order detail viewing | Medium |
| 5 | Guest checkout with temporary cart persistence | Medium |
| 6 | Product catalog browsing by category and hierarchical taxonomy management | Low |
| 7 | Search with faceted filters, autocomplete, relevance tuning | Low |
| 8 | Product detail pages with images, technical specifications, inventory status, and related products | Low |
| 9 | Customer reviews and star ratings with moderation workflow | Medium |
| 10 | Shopping cart management: add, update quantity, remove items, persist carts across sessions | Medium |
| 11 | Promotions and discount code application with validation rules | Medium |
| 12 | Checkout flow supporting multiple payment methods, and saved payment methods using tokenization | High |
| 13 | Secure payment processing and PCI-DSS handling of card transactions | High |
| 14 | Refund and chargeback handling with reconciliation and audit trails | High |
| 15 | Order confirmation and transactional email notifications | Medium |
| 16 | Admin dashboard for product inventory management | High |
| 17 | Order management: view, update fulfillment status, shipment tracking, and returns processing | High |
| 18 | Customer service tools: order lookup, issue tickets, refunds, and communication templates | High |
| 19 | Sales analytics and reporting (revenue, conversion, inventory levels) with export capabilities | Medium |
| 20 | Product recommendation engine based on browsing and purchase history | Medium |
| 21 | Conversational AI chatbot for customer support; fallback to human agents | High |
| 22 | Automated product categorization and tagging using ML | Low |
| 23 | Mobile-responsive UI and progressive enhancement for mobile web | Low |
| 24 | Multi-currency pricing and display with currency conversion and locale-aware formatting | Medium |
| 25 | Integration with shipping carriers for rate calculation and label generation | Medium |
| 26 | Email notification system for marketing and transactional communications with opt-in management | Medium |
| 27 | Customer data capture and segmentation for marketing campaigns with consent management | High |
| 28 | Customer behavior analytics tracking with privacy controls and opt-out | Medium |
| 29 | Logging, monitoring, and alerting for system health and security events | High |
| 30 | Role-based access control for administrative functions and audit logging | High |
| 31 | Data export, retention and deletion workflows to satisfy GDPR/CCPA subject requests | High |
| 32 | Third-party integrations management and secure secret handling | High |

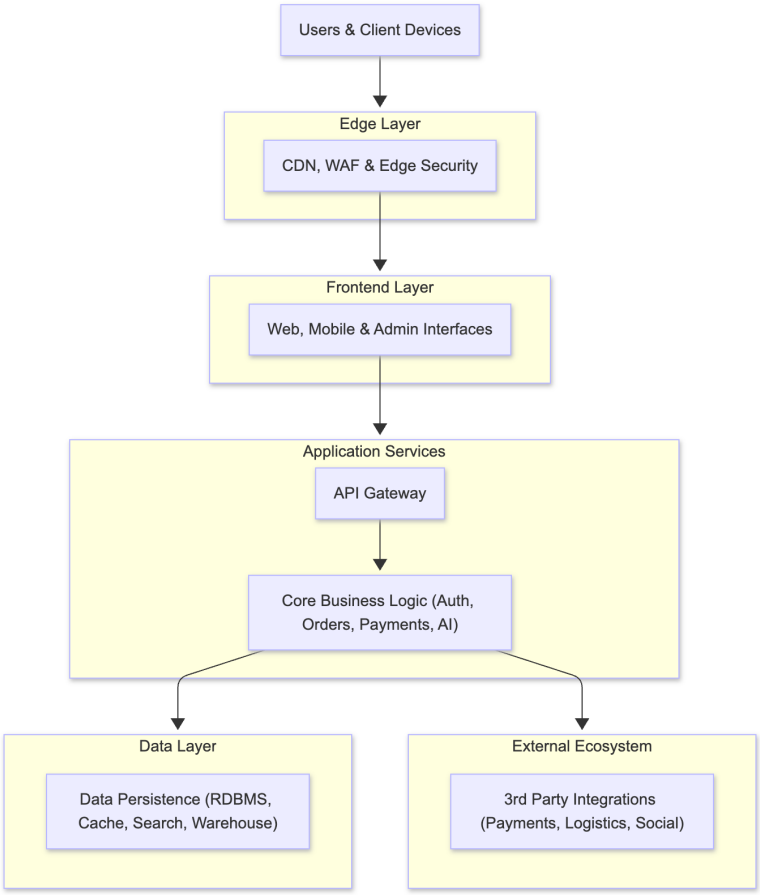


Figure 5.2: Generated Mermaid system architecture diagram.

### 5.2.2 Stage 2: Parallel Analysis

Five specialized agents execute concurrently, each analyzing different aspects of the system. The Stakeholder Analysis Agent identifies user roles and personas based on the requirements. For this e-commerce platform, the agent identifies nine distinct stakeholders with varying trust levels: Guest Users (untrusted), Customers (partially trusted), and Admin Users (trusted) represent the human actors, while service accounts include the Payment Processor and Shipping Carrier (both partially trusted), Marketing Analytics Vendor and Chatbot System (both untrusted), and the API Gateway and System Monitoring Service (both trusted). Each stakeholder is associated with specific security concerns; for instance, Customers face risks of identity theft and payment fraud, Admin Users present insider threat and privilege escalation risks, while untrusted service accounts such as the Marketing Analytics Vendor raise compliance concerns related to GDPR and data processing regulations. The specific roles identified depend on the application domain and requirements; a different system might yield different

stakeholder categories.

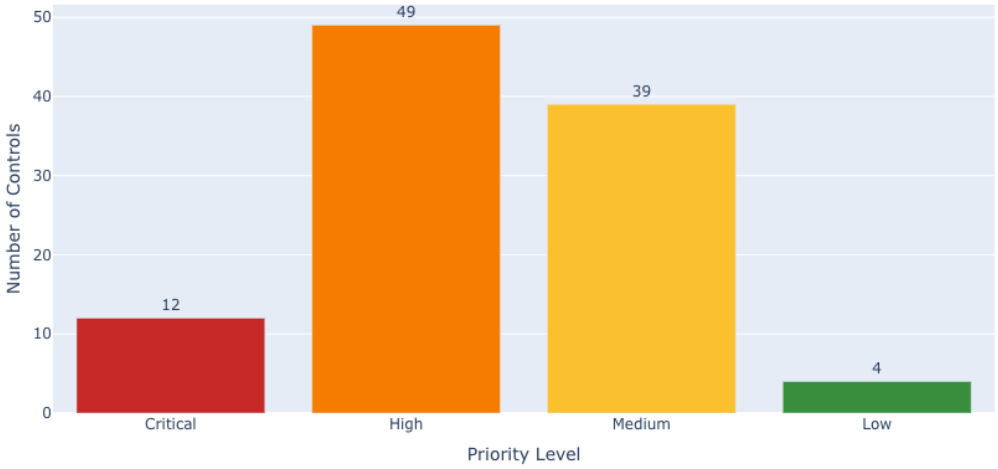


Figure 5.3: Security control priority distribution.

The Threat Modeling Agent applies the STRIDE methodology systematically across all identified components and data flows. For this example, the agent generates 28 distinct threats, as shown in Table 5.2. The number of threats varies significantly based on system complexity, attack surface, and component interactions; simpler systems may generate fewer threats, while more complex distributed systems may generate substantially more. Each threat includes a unique identifier, affected component, likelihood and impact assessments, risk level calculation, and specific mitigation strategies. Among the critical threats identified, THR-001 addresses credential stuffing and authentication bypass attacks, where attackers exploit password reuse or phished credentials to access user accounts, saved payment methods, and order history. The recommended mitigations include enforcing strong password policies with bcrypt or argon2 hashing, mandatory multi-factor authentication for sensitive actions, rate limiting with IP throttling, and integration with breach detection services. Another critical threat, THR-016, concerns third-party JavaScript vulnerabilities in frontend components, where compromised analytics scripts, widgets, or chatbots can exfiltrate cookies, credentials, and behavioral data through supply-chain attacks. Mitigations for this threat involve minimizing third-party scripts, hosting critical assets locally, applying strict Content Security Policy headers with Subresource Integrity verification, and implementing vendor risk management processes.

The Domain Security Agent queries the Weaviate vector database using semantic search to retrieve relevant security controls from OWASP ASVS, NIST SP 800-53, and ISO 27001 standards. The agent maps 3-5 specific controls from multiple standards to each of the 32 high-level requirements, providing control identifiers, requirement text, relevance explanations, integration guidance, priority levels, and verification methods.

Table 5.2: Threats identified in the e-commerce platform.

| # | **Threat Description** |
|---|---|
| 1 | Credential stuffing, password reuse, or credential theft (phished credentials) allows attackers to authenticate as legitimate users and access accounts, saved payment methods, order history, and perform fraudulent purchases. |
| 2 | Compromise or misuse of social OAuth tokens or flawed OAuth integration allows attacker impersonation or account takeover. |
| 3 | Secrets, API keys, or credentials embedded in frontend code, public repositories, or third-party bundles can be used to access internal APIs or third-party services. |
| 4 | Stored XSS through product reviews or user-generated content allows attackers to execute scripts in other users' browsers, steal session tokens, manipulate UI, or perform actions on behalf of users. |
| 5 | CSRF or forged checkout requests cause unauthorized orders, coupon manipulation, or changes to cart state leading to financial loss or inventory issues. |
| 6 | SQL injection or other injection attacks against APIs or administration interfaces allow attackers to read, modify or delete database records like Personally Identifiable Information (PII) and order data. |
| 7 | Broken access control allows attackers or low-privileged users to access or modify other users' orders, payment info, or administrative functions. |
| 8 | Admin account takeover via weak credentials, lack of MFA, or stolen administrative tokens leads to full control over product/catalog, orders, refunds, and PII. |
| 9 | Sensitive customer data (PII, addresses, partially masked payment info) ends up in logs, debug dumps, or backups are accessible/leaked to unauthorized personnel. |
| 10 | Publicly accessible object store buckets (product images or backups) expose product assets or PII stored in objects. |
| 11 | Manipulated or replayed payment tokens, or weak validation of payment webhooks, could allow fraudulent charges, refunds, or mismatched order/payment state. |
| 12 | Spoofed or forged webhooks/notifications from shipping carriers or third-party integrations result in incorrect order status updates, triggering refunds or revealing tracking/addresses. |
| 13 | DDoS or application-layer floods overwhelm the CDN or origin, causing outages and degraded user experience for customers in EU/US markets. |
| 14 | API abuse via brute-force, credential stuffing, or excessive automated requests to search, checkout or recommendation APIs leads to performance degradation or resource exhaustion. |
| 15 | Cache or index poisoning (e.g., tainted search results or malicious cached content) can serve malicious or misleading content to users. |
| 16 | Third-party JavaScript (analytics, widgets, chatbots) compromises can exfiltrate cookies, credentials or capture PII and behavioral data; supply-chain compromises of third-party vendors lead to widespread data leakage. |
| 17 | Poisoning of training data or manipulation of analytics inputs for recommendation models causes biased or malicious product recommendations, potentially leading to fraud, poor UX, or reputational damage. |
| 18 | Chatbot or AI integrations inadvertently send or reveal PII (order numbers, addresses, payment metadata) to external model providers or third-parties, violating GDPR/PCI and exposing customer data. |
| 19 | Fraudulent refunds, chargeback fraud, or disputes engineered by attackers (or colluding insiders) cause financial loss and operational overhead. |
| 20 | Improper session handling (session fixation, missing rotation at login, insecure cookie flags) allows attackers to hijack sessions and impersonate users. |
| 21 | CSRF or forged requests targeting administrative actions (e.g., price changes, inventory changes, refunds) executed without proper anti-CSRF protections. |
| 22 | Insider threats: privileged DB users or operations personnel intentionally or accidentally access or exfiltrate PII, order or payment data. |
| 23 | Misconfigured cloud IAM policies or overly permissive roles allow attackers or compromised service accounts to access databases, storage, or change configuration. |
| 24 | Open redirect or unvalidated redirect URIs in OAuth flows enable phishing or token-stealing redirects, allowing attackers to capture authorization codes or tokens. |
| 25 | Weak or misconfigured encryption (at rest or in transit) or poor key management leads to unauthorized data access or decryption of backups and stored data. |
| 26 | Gaps in logging, monitoring and alerting enable attackers to operate for long periods without detection, increasing scope and impact of breaches. |
| 27 | Search index or analytics stores inadvertently index PII or sensitive fields and expose them via search APIs or dashboards. |
| 28 | Guest checkout correlates device/user behavior causing privacy leakage or unintended linking of user profiles across sessions (tracking), or allows attackers to exploit guest flows for fraud. |

Similar to the number of threats, the number of controls mapped per requirement varies based on each use case. For example, user authentication requirements in this case map to OWASP ASVS V2.2.1 (MFA support), V2.1.7 (password hashing), NIST IA-2(1) (multi-factor authentication), and ISO 27001 A.9.4.2 (secure log-on procedures). The agent recommends OWASP ASVS Level 2 as the appropriate assurance level for this platform, though the recommended level would differ for systems with different risk profiles or compliance requirements. Figure 5.3 shows the distribution of security control priorities across the requirements while Figure 5.4 shows the distribution of threats by likelihood and impact.

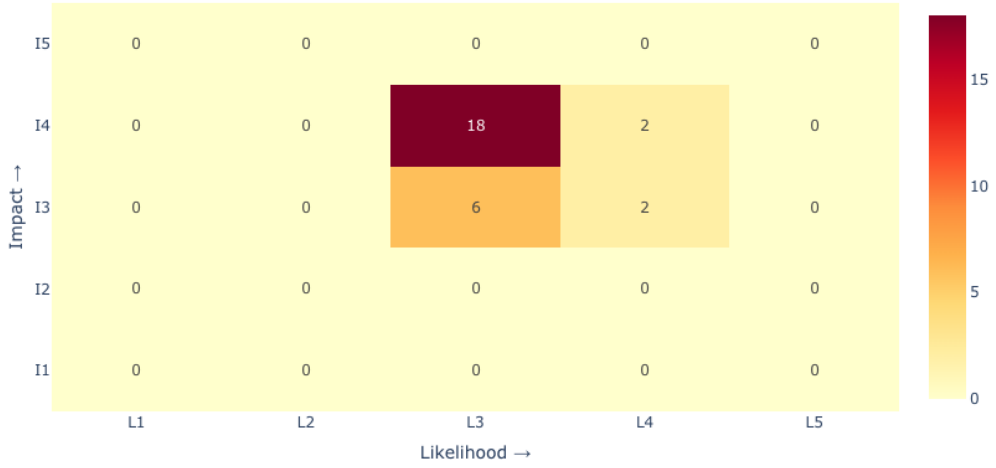


Figure 5.4: Threat distribution by likelihood and impact.

The LLM Security Specialist Agent analyzes the system description and architecture to detect AI/ML components requiring specialized security controls. For this e-commerce platform, the agent identifies three AI/ML components: a Chatbot Interface providing customer support through natural language processing, a Product Recommendation Engine suggesting products based on browsing and purchase history, and an Automated Product Categorization system that tags products using machine learning algorithms. Each component is analyzed for specific threats. The Chatbot Interface faces prompt injection and data leakage risks, the Recommendation Engine is vulnerable to model inversion attacks and Personally Identifiable Information (PII) exposure in training data, while the Categorization system presents model poisoning and bias considerations. The agent generates targeted security controls for each component, including input sanitization and output filtering for the chatbot, strict model access controls and adversarial input monitoring for the recommendation engine, and model versioning with supply chain security measures for the categorization system. These AI/ML security controls integrate with standard security practices through role-based access control, comprehensive logging of AI component interactions, and

encryption for sensitive data processed by the models.

The Compliance Agent examines the system requirements, data handling practices, and business context to identify applicable regulatory frameworks. For this e-commerce platform targeting EU and US markets, the agent identifies five applicable regulations, each with its own specific compliance controls:

- **GDPR** (processing personal data of EU residents):
  - User data access mechanisms
  - Data minimization principles
  - Encryption requirements
  - 72-hour breach notification processes
  - Mapping and enforcement of data subject rights (access, rectification, deletion, and data portability)
  - Consent management and comprehensive privacy notices
- **CCPA** (California Consumer Privacy Act, for California residents):
  - User rights to disclosure, deletion, and opt-out of sale of personal information
  - "Do Not Sell My Personal Information" workflows
  - Updates to privacy notices specific to California requirements
  - Mechanisms for consumer inquiries and response logging
- **PCI-DSS** (processing of payment card information):
  - Strong encryption of payment data (at rest and in transit)
  - Role-based access controls for payment systems
  - Regular penetration testing and security assessments
  - Maintenance of secure network infrastructure and vulnerability management programs
  - Restriction and monitoring of access to cardholder data
- **SOX** (Sarbanes-Oxley Act, for financial data management):
  - Comprehensive logging and accounting audit trails for financial transactions
  - Tamper-proof audit trails and retention policies
  - Segregation of duties and change management controls
  - Continuous monitoring for unauthorized modifications of financial records

- **COPPA** (Children's Online Privacy Protection Act, if data from users under 13 is collected):
  - Verifiable parental consent mechanisms
  - Clear parental notification and access to children's information
  - Procedures for deletion of children's data upon request
  - Age screening and data minimization for underage users

Additionally, the output includes audit and monitoring requirements across regulations, such as comprehensive logging, continuous monitoring for unauthorized access, and tamper-proof audit trails for financial transactions. The system also enforces data handling rules governing retention periods, secure deletion workflows, and data minimization practices as required by the applicable laws and standards.

### 5.2.3 Stage 3: Synthesis and Planning

Three agents work sequentially to synthesize findings and create implementation guidance. The Security Architecture Agent aggregates outputs from all previous analyses to produce a unified security architecture design. For this e-commerce platform, the agent specifies eleven architectural security principles including Zero Trust Architecture, Defense in Depth, Principle of Least Privilege, Secure by Default, Separation of Duties, and Privacy by Design. Component-level security controls are defined across six layers: Frontend, Edge, Application Services, Data Layer, External Services, and Operations. The data protection strategy establishes four classification levels, encryption requirements for data in transit and at rest, retention policies aligned with regulatory requirements, and handling procedures for access control, transmission, storage, and deletion. Third-party integration security requirements are specified for payment processors, shipping carriers, email providers, social login providers, analytics vendors, and infrastructure services.

The Implementation Roadmap Agent then generates a prioritized, phased approach organized into five phases:

1. **Immediate (0–1 months):** Addressing critical vulnerabilities such as password hashing, Multi-Factor Authentication (MFA) enforcement, and Transport Layer Security (TLS) deployment.

2. **Short-term (1–3 months):** Focusing on role-based access controls and API security.

3. **Medium-term (3–6 months):** Implementing advanced threat detection and security testing automation.

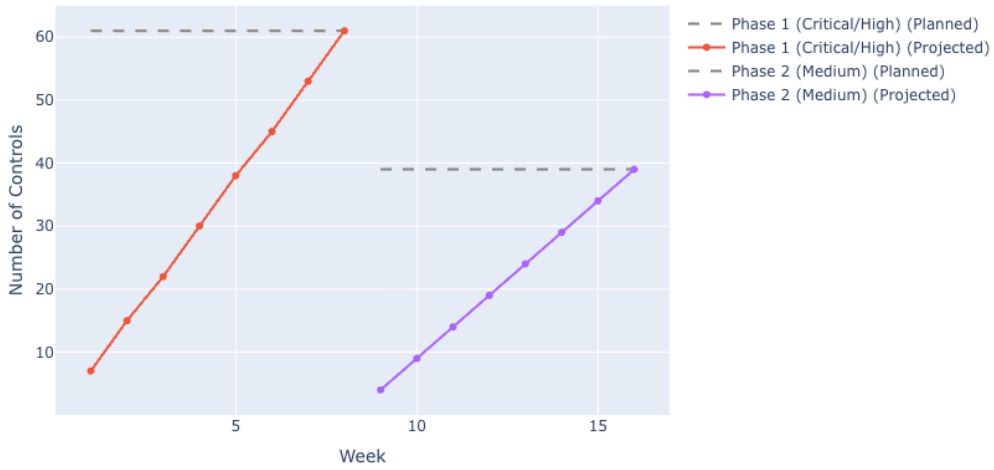


Figure 5.5: Projected implementation timeline by phase and week.

4. **Long-term (6–12 months):** Covering AI/ML security controls and comprehensive penetration testing.

5. **Ongoing:** Activities for continuous monitoring, patch management, and compliance audits.

Prioritization criteria include risk level, compliance deadlines, technical dependencies, and resource availability, with estimated resource requirements spanning security engineers, architects, and compliance specialists using Security Information and Event Management (SIEM) solutions, vulnerability scanners, and encryption libraries over an initial 3–6 month implementation period.

The Verification and Testing Agent designs a comprehensive security testing strategy that integrates security testing throughout the software development lifecycle with an emphasis on shift-left security principles. The strategy specifies eight testing methods with recommended frequencies: Static Application Security Testing (SAST) on every commit, Dynamic Application Security Testing (DAST) on a nightly or weekly basis, dependency scanning on every build, secrets scanning on every commit, container and infrastructure scanning on every deployment, penetration testing quarterly or before major releases, security code reviews for critical features, and continuous compliance scanning. Compliance verification covers multi-standard complience (OWASP ASVS, NIST SP 800-53, ISO 27001) verified against regulatory requirements (GDPR, CCPA, PCI-DSS) through automated tools and manual checks. The agent also specifies continuous monitoring requirements using SIEM for real-time monitoring with intrusion detection and prevention systems, along with key performance indicators such as mean time to detect, mean time to remediate, percentage of critical vulnerabilities patched within Service Level Agreement (SLA), and security test coverage percentage.

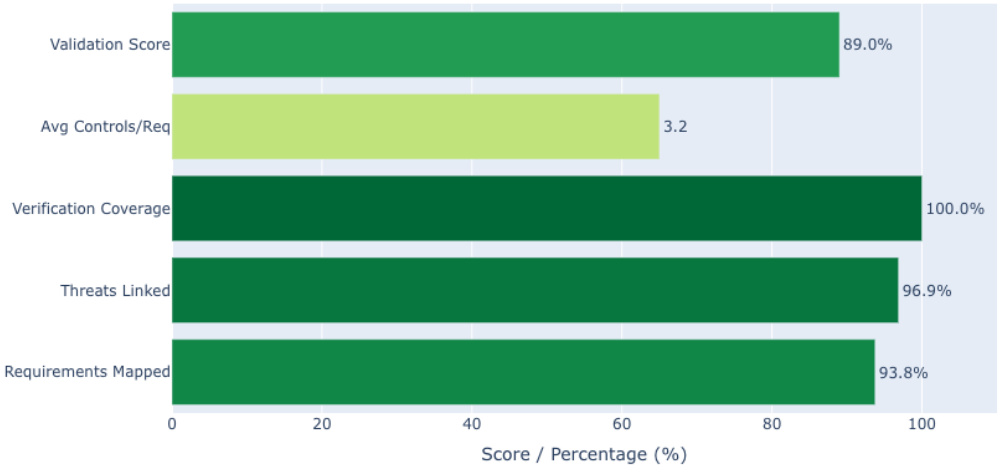


Figure 5.6: Coverage metrics.

### 5.2.4 Stage 4: Validation and Output

The Validation Agent performs comprehensive quality assurance across five dimensions: completeness, consistency, correctness, implementability, and alignment with business objectives. For this e-commerce example, the agent calculated scores of 0.85 for completeness, 0.95 for consistency, 0.90 for correctness, 0.85 for implementability, and 0.90 for alignment, resulting in an overall quality score of 0.89, which exceeds the quality threshold of 0.8. The validation highlights several key strengths, such as comprehensive security mapping across authentication, authorization, data protection, and payments, with relevant standards coverage from OWASP ASVS, NIST SP 800-53, and ISO 27001. Additionally, the agent generates prioritized and actionable improvements, including enhanced vulnerability and patch management, incident response playbooks, infrastructure resilience controls, PCI-DSS scoping guidance, credential stuffing protections, supply chain and CI/CD security measures, and AI/ML-specific enhancements for training data governance and model audit logging. The system also constructs a traceability matrix that links each requirement to associated threats, security controls, and verification methods, enabling complete end-to-end tracking from business requirements through security implementation. The full traceability matrix mapping all 32 requirements to their associated threats, security controls, and priorities is provided in Appendix B. Figure 5.6 shows the coverage metrics for this system.

Finally, the system generates a comprehensive Quarto[5] markdown report that includes executive summaries, detailed analysis sections, interactive dashboards with risk heatmaps and control distributions, implementation timelines, and complete traceability matrices. This markdown file is rendered to HTML format, pro-

[5]Quarto: https://quarto.org/

ducing an interactive security requirements report suitable for stakeholder review and implementation guidance. The report's structure and content are tailored to the specific analysis performed, with dashboards visualizing the threats, controls, and priorities identified for this e-commerce platform. The full generated report can be found in the GitHub repository hosting the code for the multi-agent system developed in this thesis[6].

## 5.3 Implementation and Design Decisions

The system was developed using Python 3.12 and leverages CrewAI for multi-agent orchestration rather than alternative approaches for several strategic reasons. CrewAI provides native support for agent specialization (roles, goals, backstories) aligned with domain-expert approaches needed for security engineering. CrewAI Flow's event-driven orchestration with decorators (`@start`, `@listen`) enables clear execution paths and dependency management, while its immutable state management through centralized state objects prevents circular dependencies and ensures transparent data flow. The framework's built-in LLM integration with per-agent model specification supports our tiered model strategy, and its native support for structured outputs through Pydantic models enforces type safety and reduces parsing errors. Additionally, CrewAI's support for both parallel and sequential execution patterns enables the system's phased workflow with concurrent analysis phases, resulting in approximately a 40% reduction in execution time compared to sequential execution.

Other protocols were also considered. Model Context Protocol (MCP) was evaluated but not chosen. MCP is designed for single-agent scenarios requiring flexible, on-the-fly tool use and is most suitable for generalist AI assistants accessing diverse enterprise data sources. The security requirements system is domain-specialized with predetermined analysis phases and explicit tool use (Weaviate queries in the Domain Security Agent), making MCP's flexibility unnecessary. Additionally, MCP's tool discovery mechanism would add overhead in a tightly orchestrated multi-agent pipeline. Instead, the Domain Security Agent uses direct tool invocation (WeaviateQueryTool), which is more deterministic and cost-effective.

Agent2Agent Protocol (A2A) was similarly evaluated. A2A enables agent-to-agent communication across heterogeneous systems via standardized HTTP/JSON-RPC messaging, which is particularly useful for loosely coupled agent networks where agents are developed independently by different vendors. The security requirements system operates in a tightly coupled, single-developer context with shared state management through CrewAI Flow, making A2A's standardization unnecessary. The current state-mediated communication pattern provides sufficient transparency and auditability for security-critical applications. A2A would

[6]GitHub repository: https://github.com/SavvasMohito/mas-sre

introduce unnecessary protocol complexity without addressing a present need.

### 5.3.1 Knowledge Base: Vector Database Design

The Domain Security Agent requires grounding in verified, factual security knowledge to prevent LLM hallucinations and ensure that all recommended controls correspond to actual, verifiable standards rather than fabricated control identifiers. The three security standards, OWASP ASVS, NIST SP 800-53, and ISO 27001, collectively contain over 800 distinct security controls, each with detailed requirement text, control identifiers, and contextual information. To optimize token usage and retrieval precision, the parsing process indexed only the normative control requirements, excluding non-normative annexes and auxiliary guidance text. Encoding all these controls directly in prompts would consume excessive context tokens, making LLM interactions prohibitively expensive and limiting the model's ability to focus on the specific requirements being analyzed. Furthermore, the agent needs to perform a semantic search to find relevant controls based on natural language queries rather than exact keyword matching, as requirements may be expressed in various ways that do not directly match standard control text.

To address these challenges, the system implements an Retrieval-Augmented Generation (RAG) approach using a Weaviate vector database for semantic search. The knowledge base construction process began by downloading the raw specification files for each standard from their official sources. Then, it parsed and split each standard's content into individual control entries, preserving control identifiers, requirement text, chapter references, and metadata. These parsed controls were organized according to the standard and imported into a Weaviate collection with a schema designed to support semantic search across control text, identifiers, and metadata fields. Each control was embedded using OpenAI's `text-embedding-3-small` model, generating 1536-dimensional vector representations that capture semantic meaning, and the embeddings were indexed using Hierarchical Navigable Small World (HNSW)[7] for efficient similarity search across large vector collections. When the Domain Security Agent needs to find relevant controls for a requirement, it formulates a semantic query that is embedded using the same model. Weaviate then performs a similarity search, returning the top 5 most relevant controls across all three standards.

This RAG approach ensures that every recommended control cites actual, verifiable standards with correct identifiers and requirement text, providing auditable traceability crucial for regulatory compliance and preventing the LLM from hallucinating non-existent control references. Similar to the model selection rationale, the choice of Weaviate as the vector database was driven by practical considerations: prior experience with its capabilities and familiarity with local deployment

[7]HNSW: https://www.pinecone.io/learn/series/faiss/hnsw/

procedures facilitated rapid implementation. However, the architecture is not inherently dependent on Weaviate. The operations performed (chunking documents, generating embeddings, storing vectors, and executing semantic similarity searches) are standard capabilities provided by virtually any vector database, whether proprietary solutions such as Pinecone or open-source alternatives such as Milvus, Qdrant, or ChromaDB. Consequently, the vector storage component can be substituted without architectural modifications.

# Chapter 6

# Results and Analysis

This chapter presents the empirical findings from the evaluation of the proposed multi-agent system for automating the translation of security requirements. The evaluation employs a mixed-methods approach combining quantitative system performance metrics with qualitative expert feedback to comprehensively assess the system's capabilities and practical utility. The chapter is organized around the three research questions established in Chapter 4, with each major section dedicated to addressing one research question through systematic presentation of empirical data. The evaluation was conducted using 14 distinct security report generations across diverse application domains, including e-commerce platforms, healthcare systems, financial services, and enterprise collaboration tools. Each generation involved translating high-level business requirements into comprehensive security reports containing threat analyses, security requirements, and standards-compliant control mappings.

## 6.1 RQ1 - System Effectiveness Analysis

This section addresses Research Question 1: *How effectively can a multi-agent system, leveraging LLMs, translate high-level software requirements into detailed, security-compliant software requirements?* The analysis examines the system's performance across 14 security report generations, evaluating the quality of automated translation, the comprehensiveness of security coverage, and the accuracy of standards compliance mappings. Three core dimensions are investigated: (1) the quality and completeness of generated security requirements, (2) the accuracy and coverage of security control mappings to established standards, and (3) the comprehensiveness of threat identification and security aspect coverage.

### 6.1.1 Translation Quality Assessment

The overall translation quality was assessed through five dimensions: completeness, consistency, correctness, implementability, and alignment with security best practices. Figure 6.1 presents the distribution of quality scores across these dimensions for all 14 generations. Consistency emerged as the highest-performing

dimension with a mean score of 0.91 ($SD = 0.055$), indicating that the system reliably produced coherent requirements that aligned with the established security frameworks. Alignment with security standards achieved a mean score of 0.89 ($SD = 0.054$), demonstrating strong integration of domain knowledge from the vector database. The correctness of threat identification and requirement specifications achieved a mean score of 0.88 ($SD = 0.026$), indicating the system's ability to accurately map business contexts to relevant security concerns.

Completeness scores exhibited greater variability, with a mean of 0.81 ($SD = 0.066$), suggesting that while the system generally provided comprehensive coverage, certain generations produced fewer requirements relative to the scope of input specifications. Implementability scored lowest among the quality dimensions with a mean of 0.76 ($SD = 0.061$), indicating that while requirements were technically sound, some lacked the specificity needed for immediate implementation without further elaboration.

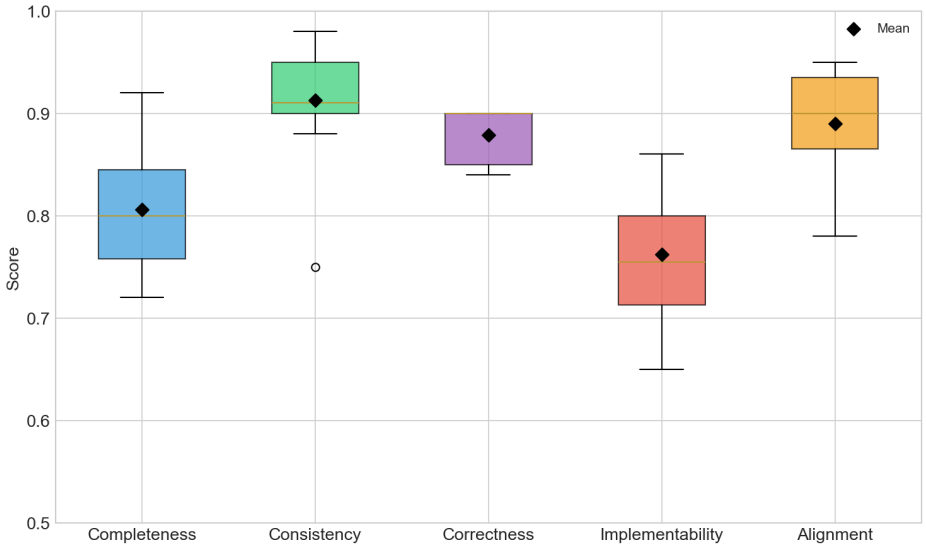


Figure 6.1: Quality Dimension Scores Distribution.

The overall quality score, computed as the weighted average of all five dimensions, is presented in Figure 6.2. Across all 14 generations, the mean quality score was 0.85 with a standard deviation of 0.038, indicating consistent performance with relatively low variability. The majority of generations (7 out of 14, 50.0%) achieved scores above 0.85, classified as "high quality" while 6 generations (42.9%) fell within the "acceptable" range (0.80-0.85). Only one generation scored below 0.80 at 0.79. The highest-performing generation achieved a score of 0.92, demonstrating the system's capability to produce comprehensive, accurate, and standards-aligned security requirements when provided with well-structured input.

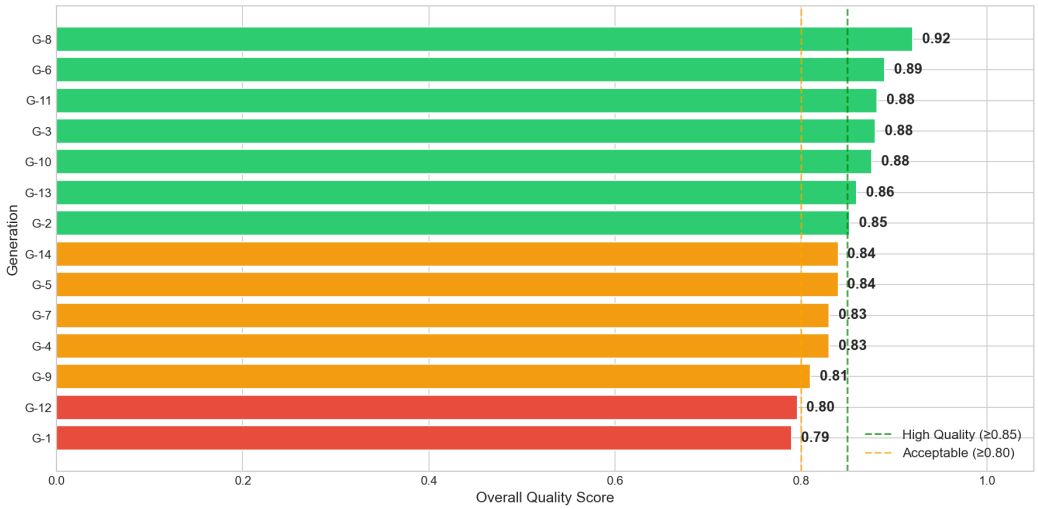


Figure 6.2: Translation Quality - Overall Validation Scores.

### 6.1.2 Requirements Coverage Analysis

The breadth and depth of requirements generation were analyzed to understand the trade-off between comprehensive coverage of security aspects versus detailed specification of individual requirements. Each generation (G-1 through G-14) corresponds to a security report produced from a participant's proprietary requirements document, ensuring the input data was not present in publicly available training datasets. Figure 6.3 illustrates this relationship through a scatter plot where the x-axis represents the number of unique security requirements generated (breadth) and the y-axis represents the average number of security controls mapped per requirement (depth). The color intensity indicates overall quality scores. Across all 14 generations, the mean number of requirements generated was 24.1, ranging from 16 to 35, while the mean depth was 3.18 controls per requirement.

Results reveal distinct generation patterns that illustrate the trade-off between breadth and depth. The high-depth cluster includes G-7 (22 requirements, 4.00 controls per requirement), G-12 (28 requirements, 4.00 controls per requirement), G-3 (20 requirements, 3.95 controls per requirement), and G-2 (20 requirements, 3.70 controls per requirement), demonstrating that these generations prioritized detailed control mapping per requirement. Notably, G-12 achieved both high breadth and the highest depth, representing an outlier that excelled in both dimensions. The high-breadth, low-depth cluster includes G-6 (35 requirements, 2.14 controls per requirement), G-10 (30 requirements, 2.53 controls per requirement), and G-8 (28 requirements, 2.64 controls per requirement), indicating a strategy that prioritized broader security coverage over detailed control specification. The balanced cluster includes generations such as G-1 (23 requirements, 3.39

controls per requirement), G-9 (25 requirements, 3.20 controls per requirement), and G-4 (27 requirements, 3.26 controls per requirement), which maintained moderate values in both dimensions. The lowest breadth was observed in G-13 (16 requirements) and G-14 (18 requirements), though both maintained depth values close to the mean at 3.19 and 3.00 controls per requirement, respectively.

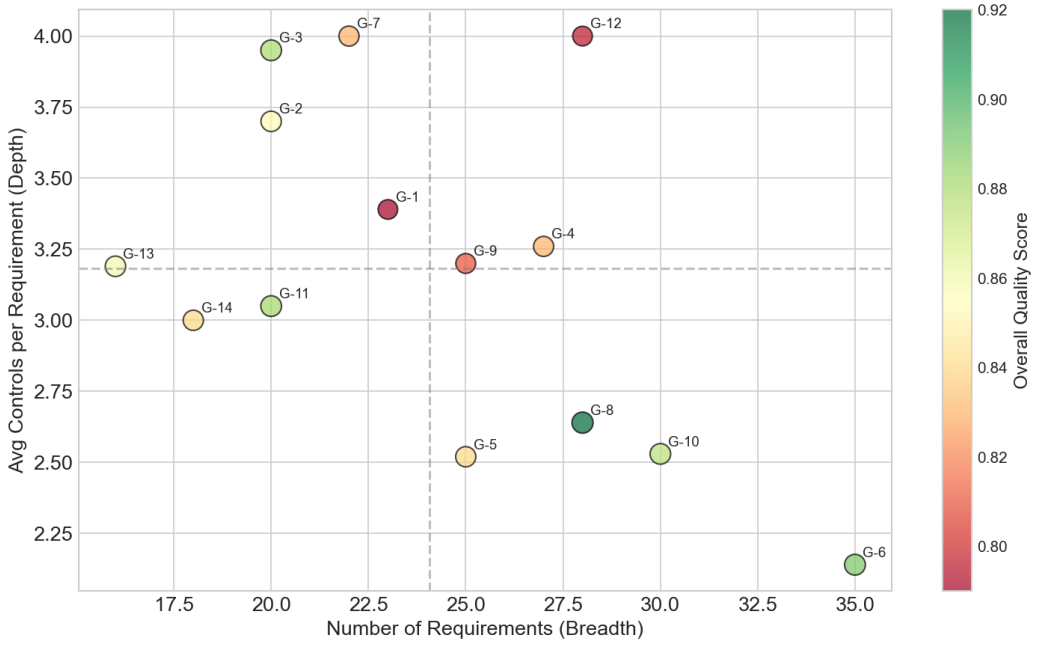


Figure 6.3: Breadth vs Depth - Requirements Coverage Trade-off.

Analysis of traceability metrics demonstrates the system's ability to maintain end-to-end mappings from input requirements through threats to security controls and verification tests. The results are summarized in Table 6.1. Verification test coverage achieved a perfect 100% across all 14 generations, indicating that every generated requirement included testable validation criteria. Threat mapping coverage, which measures the percentage of generated security requirements that explicitly reference identified threats, achieved a mean of 98.6%, with 11 out of 14 generations reaching 100%. Only three generations fell below perfect threat coverage: G-7 (90.9%), G-14 (94.4%), and G-11 (95.0%). Control mapping coverage, representing the proportion of requirements linked to specific standards-based controls, exhibited the most variability among the three metrics, ranging from 71.4% (G-6) to 100% (G-7, G-11, and G-13), with a mean of 87.6%.

### 6.1.3 Standards Compliance Validation

The system's ability to accurately map security requirements to established standards was assessed across three frameworks: OWASP ASVS, NIST SP 800-53,

Table 6.1: Traceability Metrics Requirements Coverage.

| **Gen.** | **Threat Map (%)** | **Control Map (%)** | **Verific. Tests (%)** |
|---|---|---|---|
| G-1 | 100.0 | 82.6 | 100.0 |
| G-2 | 100.0 | 90.0 | 100.0 |
| G-3 | 100.0 | 85.0 | 100.0 |
| G-4 | 100.0 | 92.6 | 100.0 |
| G-5 | 100.0 | 72.0 | 100.0 |
| G-6 | 100.0 | 71.4 | 100.0 |
| G-7 | 90.9 | 100.0 | 100.0 |
| G-8 | 100.0 | 89.3 | 100.0 |
| G-9 | 100.0 | 76.0 | 100.0 |
| G-10 | 100.0 | 86.7 | 100.0 |
| G-11 | 95.0 | 100.0 | 100.0 |
| G-12 | 100.0 | 85.7 | 100.0 |
| G-13 | 100.0 | 100.0 | 100.0 |
| G-14 | 94.4 | 94.4 | 100.0 |
| *Mean* | *98.6* | *87.6* | *100.0* |

and ISO 27001. The distribution of mapped controls by standard for each generation is summarized in Table 6.2. On average, each generation produced 75.2 total control mappings across the three standards. NIST SP 800-53 emerged as the most frequently referenced standard with an average of 30.5 controls per generation (40.6% of total mappings), followed by OWASP ASVS with 24.7 controls (32.9%), and ISO 27001 with 19.9 controls (26.4%). The highest number of control mappings was observed in G-12 with 112 total controls, while G-13 produced the fewest with 51 controls. Notably, G-6 achieved a perfectly balanced distribution across standards with 25 controls mapped to each framework.

Control coverage, defined as the percentage of generated security requirements that received at least one standards-based control mapping, is presented in Figure 6.4. The mean control coverage across all generations was 87.5%. Six generations achieved excellent coverage ($\geq 90\%$): G-7, G-11, and G-13 reached perfect 100% coverage, followed by G-14 (94.4%), G-4 (92.6%), and G-2 (90.0%). Five generations demonstrated good coverage (80-89%): G-8 (89.3%), G-10 (86.7%), G-12 (85.7%), G-3 (85.0%), and G-1 (82.6%). The remaining three generations exhibited coverage below 80%: G-9 (76.0%), G-5 (72.0%), and G-6 (71.4%), suggesting opportunities for improved control identification in these cases.

Cross-standard consistency, measuring the distribution of controls across the three standards for each generation, reveals that NIST SP 800-53 controls dominated with a mean of 40.6% of total mappings ($SD = 5.8\%$), followed by OWASP ASVS at 32.9% ($SD = 3.9\%$), and ISO 27001 at 26.3% ($SD = 4.6\%$). OWASP

Table 6.2: Security Control Mappings by Standard.

| **Generation** | **OWASP ASVS** | **NIST SP 800-53** | **ISO 27001** | **Total** |
|---|---|---|---|---|
| G-1 | 26 (33.3%) | 29 (37.2%) | 23 (29.5%) | 78 |
| G-2 | 22 (29.7%) | 30 (40.5%) | 20 (27.0%) | 74 |
| G-3 | 35 (44.3%) | 25 (31.6%) | 19 (24.1%) | 79 |
| G-4 | 27 (30.7%) | 34 (38.6%) | 27 (30.7%) | 88 |
| G-5 | 20 (31.7%) | 23 (36.5%) | 20 (31.7%) | 63 |
| G-6 | 25 (33.3%) | 25 (33.3%) | 25 (33.3%) | 75 |
| G-7 | 22 (25.0%) | 44 (50.0%) | 22 (25.0%) | 88 |
| G-8 | 26 (35.1%) | 28 (37.8%) | 20 (27.0%) | 74 |
| G-9 | 26 (32.5%) | 34 (42.5%) | 20 (25.0%) | 80 |
| G-10 | 25 (32.9%) | 37 (48.7%) | 14 (18.4%) | 76 |
| G-11 | 20 (32.8%) | 29 (47.5%) | 12 (19.7%) | 61 |
| G-12 | 38 (33.9%) | 45 (40.2%) | 29 (25.9%) | 112 |
| G-13 | 16 (31.4%) | 25 (49.0%) | 10 (19.6%) | 51 |
| G-14 | 18 (33.3%) | 19 (35.2%) | 17 (31.5%) | 54 |
| *Mean* | *24.7 (32.9%)* | *30.5 (40.6%)* | *19.9 (26.3%)* | *75.2* |

ASVS exhibited the most consistent distribution across generations with the lowest standard deviation, while NIST SP 800-53 showed the highest variability. Notable outliers include G-6, which achieved a perfectly balanced distribution (33.3% across all three standards), G-7, which was most NIST-heavy (50.0% NIST), and G-3, which was most OWASP-heavy (44.3% OWASP). Despite this variability, the RAG-based retrieval mechanism successfully identified relevant controls from all three standards rather than disproportionately favoring a single framework.

The verification level distribution for OWASP ASVS controls reveals that the system predominantly identified Level 2 (Standard) controls, which accounted for an average of 17.5 controls per generation (76% of total ASVS mappings), compared to 5 Level 1 (Basic) controls (21%) and 0.6 Level 3 (Advanced) controls (3%). This distribution aligns with the practical reality that most applications require standard-level security verification, while basic controls are often insufficient and advanced controls may be unnecessarily stringent for many contexts.

### 6.1.4 Threat Analysis and Security Aspects Coverage

Threat identification analysis, based on the STRIDE methodology, is presented in Figure 6.5. Across all 14 generations, an average of 30.0 threats were identified per generation, with individual generations ranging from 28 to 36 threats. Notably, 12 out of 14 generations identified exactly 30 threats, demonstrating remarkable

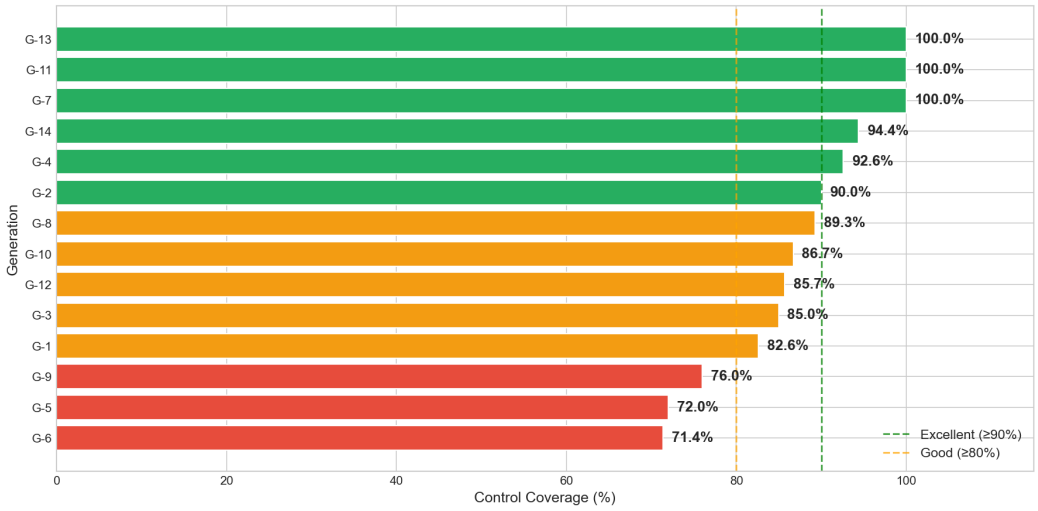


Figure 6.4: Compliance Coverage - Requirements with Security Control Mapping.

consistency in the threat modeling process. The risk level distribution shows that High-risk threats dominated, accounting for 63.8% of all identified threats (mean of 19.1 per generation), followed by Medium-risk threats at 28.1% (mean of 8.4), and Critical-risk threats at 7.9% (mean of 2.4). Low-risk threats were nearly absent, comprising only 0.2% of total threats, with just a single Low-risk threat identified across all generations (in G-14). The generation with the highest number of threats was G-7 with 36 threats, while G-2 and G-6 identified the most Critical-risk threats (4 each). This distribution reflects the system's focus on identifying serious security concerns that require immediate attention rather than exhaustively cataloging low-priority risks.

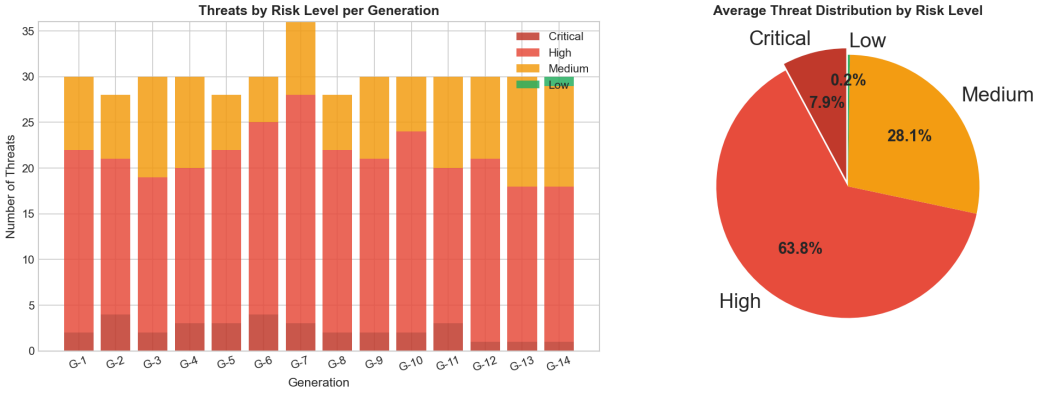


Figure 6.5: Threats by Risk Level per Generation.

All 14 generations identified threats in each of the six STRIDE categories (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service,

and Elevation of Privilege), achieving 100% coverage. This universal coverage across all threat categories indicates that the Threat Modeling Agent systematically applies the STRIDE framework to analyze security from multiple perspectives, ensuring no major threat category is overlooked regardless of the specific application context. The consistent identification of threats across all STRIDE categories demonstrates the robustness of the multi-agent approach in comprehensive threat analysis.

## 6.2 RQ2 - Architectural Patterns and Multi-Agent Collaboration

This section addresses Research Question 2: *What architectural patterns and collaborative mechanisms are essential for a multi-agent system to achieve robust and systematic security requirements engineering?* While Chapter 5 presented the design rationale for the multi-agent architecture, this section examines the empirical outcomes of those architectural decisions across the 14 security report generations. The analysis focuses on validating whether the chosen patterns achieved their intended objectives and identifying the mechanisms that proved essential for reliable security requirements generation.

### 6.2.1 Workflow Architecture and Execution Patterns

The four-stage workflow architecture was designed to balance sequential dependencies with parallel execution opportunities.

- **Stage 1** executes the Requirements Analysis Agent sequentially because all subsequent agents depend on its outputs: the application summary, extracted requirements, and system architecture diagram serve as foundational inputs for all downstream analyses.
- **Stage 2** exploits the independence of five specialized analyses (Stakeholder, Threat Modeling, Domain Security, LLM Security, and Compliance) by executing them in parallel, as none of these agents require outputs from the others.
- **Stage 3** synthesizes all Stage 2 outputs through three sequential agents (Security Architecture, Implementation Roadmap, Verification Strategy) that must execute in order because each builds upon the previous agent's synthesis.
- **Stage 4** performs validation and output generation, implementing a self-evaluation loop that enables iterative refinement when quality thresholds are not met.

Empirical evidence from the 14 generations validates this architectural design. All generations completed the four-stage workflow without deadlocks or circular dependencies, confirming that the explicit dependency graph defined through event-driven coordination (`@listen` decorators) correctly enforces execution order. The parallel execution in Stage 2 achieved approximately 40% reduction in total execution time compared to fully sequential execution, reducing typical analysis time from an estimated 20-25 minutes to 10-15 minutes for medium-sized requirements documents. This performance improvement directly resulted from the architectural decision to identify and exploit parallelizable analysis phases.

The state management architecture employed an immutable, centralized state object as the sole coordination mechanism between agents. Rather than implementing direct agent-to-agent communication, all outputs flow forward through the state object, with each agent reading from the current state and writing its outputs back to the state upon completion. This unidirectional, state-mediated communication pattern proved robust across all 14 generations, with zero instances of state corruption, race conditions, or data inconsistencies observed. The immutable state model ensured that parallel agents in Stage 2 received consistent snapshots of Stage 1 outputs without conflicts, while the centralized nature of the state object provided complete auditability of the data flow through the pipeline.

The decision to decompose security requirements engineering into ten specialized agents rather than employing a single monolithic LLM was validated by the distinct contributions each agent made to the final output. As presented in Section 6.1, the Threat Modeling Agent generated an average of 30.0 threats per generation with complete STRIDE category coverage, the Domain Security Agent mapped an average of 75.2 security controls per generation across three standards, and the Validation Agent assessed quality across five distinct dimensions. These outputs represent specialized analyses that would be difficult to achieve with equivalent depth and consistency through a single prompt to a general-purpose LLM. Preliminary manual testing confirmed that a single-agent architecture struggled to process the full context of 800+ security controls, resulting in frequent 'loss in the middle' and hallucinations that necessitated this multi-agent decomposition. The specialization enabled each agent to apply focused domain expertise, systematic methodologies (such as STRIDE for threat modeling), and dedicated tool access (such as vector database queries for the Domain Security Agent). Furthermore, the parallel execution of specialized agents directly contributed to the observed performance improvements, as their independent nature allowed for concurrent processing.

### 6.2.2 Knowledge Base Integration and Output Reliability

A critical architectural decision was the implementation of an RAG approach using a Weaviate vector database to ground security control recommendations in verified standards. This decision addressed a fundamental challenge with LLM-

based security requirements generation: the tendency of language models to hallucinate plausible but non-existent security control identifiers and requirements. By pre-loading 803 controls from OWASP ASVS (345 controls), NIST SP 800-53 (323 controls), and ISO 27001 (135 controls) into a vector database with semantic search capabilities, the Domain Security Agent could retrieve and cite actual standards rather than generating fabricated control references.

The empirical results demonstrate the effectiveness of this grounding mechanism. Across all 14 generations, the security control mappings consistently referenced real control identifiers from the three integrated standards. The distribution of mapped controls across standards (NIST SP 800-53 at 40.6%, OWASP ASVS at 32.9%, and ISO 27001 at 26.3%) reflects balanced retrieval across the knowledge base rather than over-reliance on any single standard. The mean of 75.2 total control mappings per generation, with an average of 3.18 controls per requirement, indicates that the semantic search effectively identified multiple relevant controls for each security requirement. The control coverage metric of 87.6% (percentage of requirements receiving at least one standards-based control mapping) further validates that the RAG approach successfully matched most requirements to appropriate controls.

Structured output enforcement through Pydantic data models provided an additional layer of reliability. All agent outputs were constrained to predefined schemas that specified required fields, data types, and valid values. This enforcement prevented free-form generation that could introduce inconsistencies or unparseable outputs. Across all 14 generations, every agent output successfully validated against its schema, with zero JSON parsing failures or schema violations observed. The structured outputs also enabled the reliable construction of the traceability matrix, which requires precise linking between requirements, threats, controls, and verification methods.

The validation loop architecture implemented a quality gate with a 0.7 threshold score and a maximum of three refinement iterations. The Validation Agent assessed each generation across five dimensions (completeness, consistency, correctness, implementability, and alignment), providing structured feedback when scores fell below the threshold. Analysis of the 14 generations reveals that 13 achieved overall quality scores of 0.80 or higher, with only one generation scoring 0.79. The mean quality score of 0.85 ($SD = 0.038$) exceeded the threshold by a substantial margin, indicating that the combination of specialized agents, grounded knowledge retrieval, and structured outputs consistently produced high-quality results. The low variability in quality scores across generations demonstrates the reliability of the architectural approach, as the multi-agent pipeline consistently produces high-quality output regardless of the specific input requirements or application domain.

## 6.3 RQ3 - System Performance Evaluation

This section addresses Research Question 3: *How does the proposed multi-agent system perform in terms of efficiency, accuracy, and scalability when compared to traditional and other AI-augmented approaches for security requirements engineering?* The analysis evaluates the system's practical utility, perceived effectiveness, and likelihood of adoption through expert assessment. Rather than direct comparison with alternative approaches (which is beyond the scope of this thesis), the evaluation focuses on gathering professional feedback from practitioners regarding the system's usefulness, reliability, and potential for real-world deployment. Fifteen security professionals and software engineers participated in the evaluation, providing both quantitative ratings and qualitative feedback based on their review of security reports generated by the multi-agent system.

### 6.3.1 Quantitative Assessment of System Utility

This subsection analyzes the results of the developer survey, which assessed the system's practical utility and likelihood of adoption through seven structured Likert-scale questions (Q1-Q7). Each question measured a specific dimension of behavioral intention on a 1-5 scale, where 1 represents "Strongly Disagree" and 5 represents "Strongly Agree." The following paragraphs present the response distributions for each question, followed by summary statistics in Table 6.3.

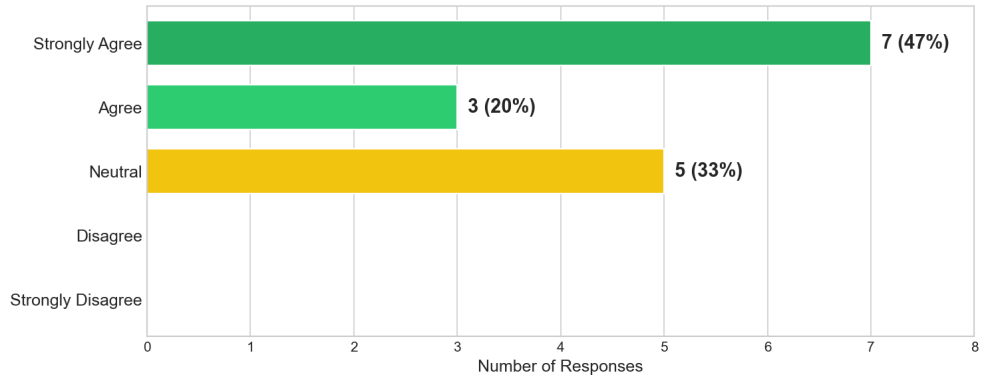


Figure 6.6: Q1: Intent to Use Insights in Next Project.

Q1 asked participants whether they intend to use insights from the security report in their next project. As shown in Figure 6.6, 7 participants (46.7%) selected "Strongly Agree," 3 participants (20.0%) selected "Agree," and 5 participants (33.3%) selected "Neutral." No participants expressed disagreement, yielding a mean score of 4.13 ($SD = 0.92$).

Q2 assessed whether participants plan to incorporate the report's recommendations into their requirements process in the next quarter. Figure 6.7 shows that

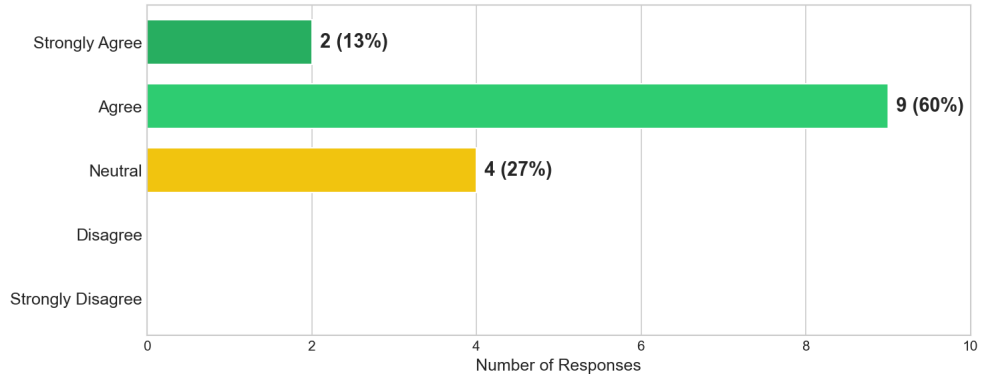


Figure 6.7: Q2: Plan to Incorporate into Requirements Process.

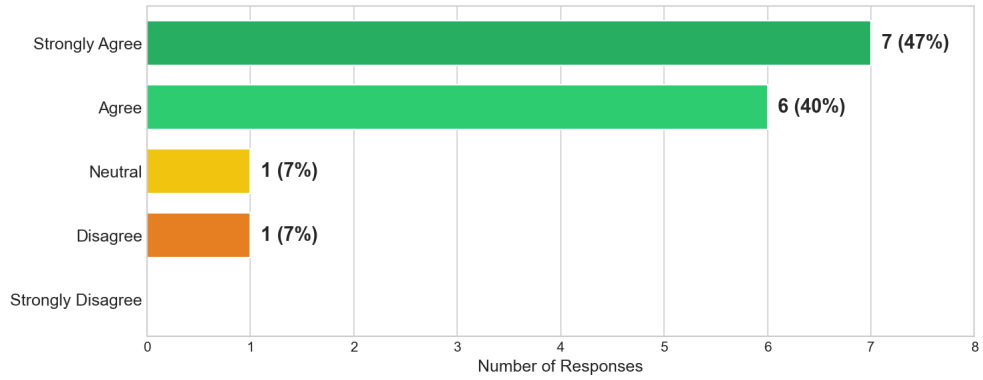


Figure 6.8: Q3: Rely on OWASP/NIST/ISO Mappings.

2 participants (13.3%) selected "Strongly Agree," 9 participants (60.0%) selected "Agree," and 4 participants (26.7%) selected "Neutral." This question received the lowest mean score at 3.87 ($SD = 0.64$), suggesting moderate concerns about practical integration into existing organizational workflows.

Q3 measured the likelihood of participants relying on the report's OWASP/NIST/ISO mappings when defining security requirements. As presented in Figure 6.8, 7 participants (46.7%) selected "Strongly Agree," 6 participants (40.0%) selected "Agree," 1 participant (6.7%) selected "Neutral," and 1 participant (6.7%) selected "Disagree." Despite the single disagreement, this question achieved one of the highest mean scores at 4.27 ($SD = 0.88$).

Q4 asked whether participants would recommend using reports like this for their team's projects. Figure 6.9 indicates that 5 participants (33.3%) selected "Strongly Agree," 9 participants (60.0%) selected "Agree," and 1 participant (6.7%) selected "Neutral." No disagreement was recorded, resulting in a mean of 4.27 ($SD = 0.59$).

Q5 evaluated whether participants would request generating such security

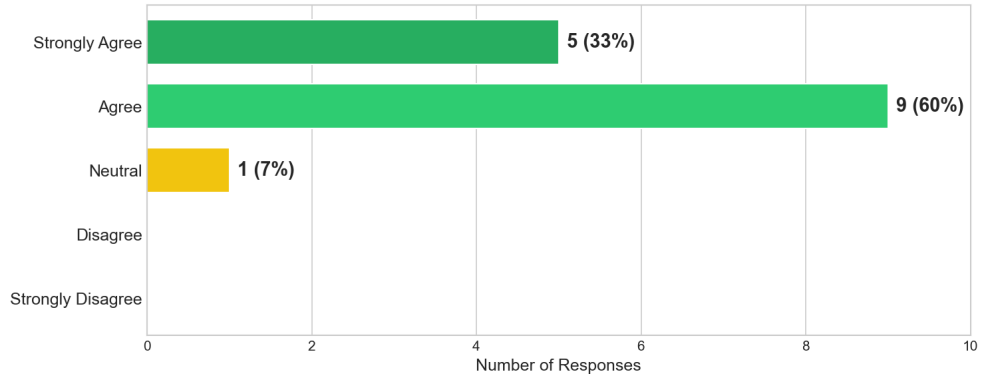


Figure 6.9: Q4: Recommend for Team Projects.

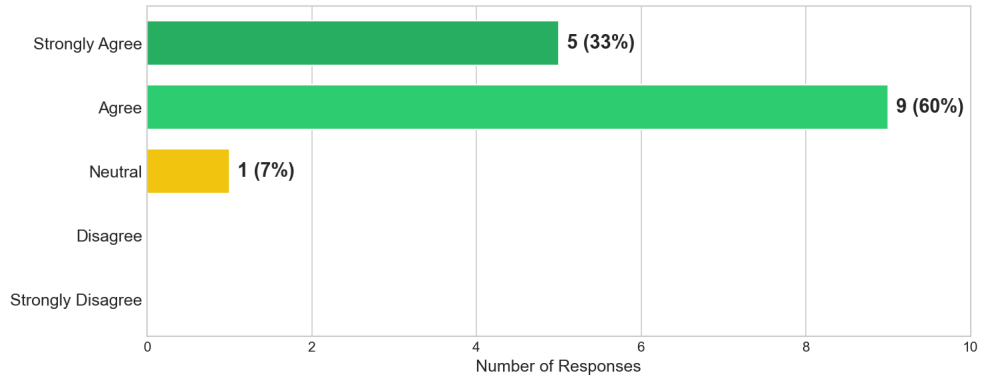


Figure 6.10: Q5: Request for New Initiatives.

reports for new initiatives if available. As shown in Figure 6.10, 5 participants (33.3%) selected "Strongly Agree," 9 participants (60.0%) selected "Agree," and 1 participant (6.7%) selected "Neutral." This question also achieved a mean of 4.27 ($SD = 0.59$), tied for the highest among all questions.

Q6 assessed whether participants are likely to allocate dedicated time during requirements elicitation to review the report. Figure 6.11 shows that 5 participants (33.3%) selected "Strongly Agree," 6 participants (40.0%) selected "Agree," and 4 participants (26.7%) selected "Neutral." The mean score was 4.07 ($SD = 0.80$), indicating willingness to incorporate report review into existing workflows.

Q7 measured participants' intention to continue requesting such reports after the study concludes. As presented in Figure 6.12, 4 participants (26.7%) selected "Strongly Agree," 8 participants (53.3%) selected "Agree," and 3 participants (20.0%) selected "Neutral." The mean score was 4.07 ($SD = 0.70$), reflecting sustained interest in the system beyond the evaluation period.

All items achieved mean scores above 3.87, indicating strong overall agreement that the multi-agent system provides practical value. The three highest-rated items were Q3, Q4, and Q5, each with a mean of 4.27, suggesting that partici-

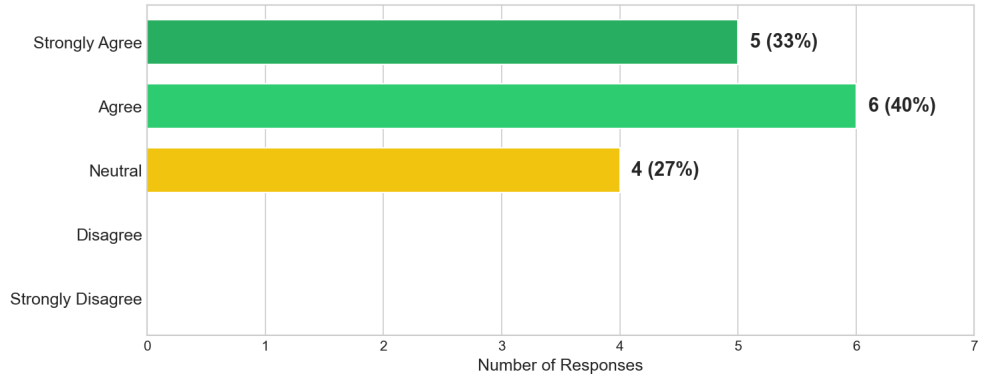


Figure 6.11: Q6: Allocate Time During Elicitation.

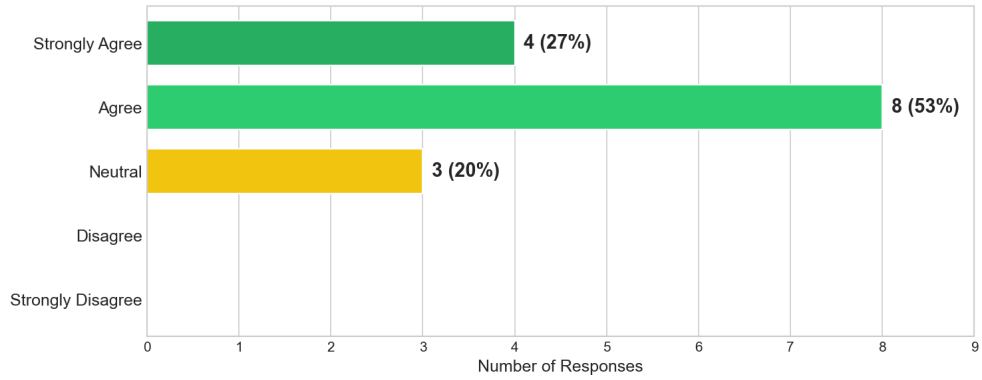


Figure 6.12: Q7: Continue Requesting After Study.

pants particularly valued the system's ability to map requirements to established security standards and its potential for team-based adoption. Summary statistics for all questions are provided in Table 6.3.

### 6.3.2 Qualitative Feedback and Improvement Suggestions

Participants provided open-ended responses to the question: "What single change would most increase your likelihood of using these security reports in your projects?" Fourteen distinct improvement suggestions were provided. Table 6.4 presents these suggestions, categorized by enhancement type, with condensed versions for ease of table presentation.

The qualitative feedback revealed several recurring themes across participants' suggestions for improvement. Process integration and output format concerns were most prominent, with participants requesting better alignment with organizational workflows, multiple output modalities (e.g., Markdown for LLM agents, formats compatible with code analysis tools), and executive-level summaries prioritizing critical actions. Documentation and customization suggestions empha-

Table 6.3: Questionnaire Summary Statistics.

| **Question** | **Mean** | **SD** | **Min-Max** | **Median** |
|---|---|---|---|---|
| Q1: Use insights in next project | 4.13 | 0.92 | 3-5 | 4.0 |
| Q2: Incorporate into requirements process | 3.87 | 0.64 | 3-5 | 4.0 |
| Q3: Rely on OWASP/NIST/ISO mappings | 4.27 | 0.88 | 2-5 | 4.0 |
| Q4: Recommend for team projects | 4.27 | 0.59 | 3-5 | 4.0 |
| Q5: Request for new initiatives | 4.27 | 0.59 | 3-5 | 4.0 |
| Q6: Allocate time during elicitation | 4.07 | 0.80 | 3-5 | 4.0 |
| Q7: Continue requesting after study | 4.07 | 0.70 | 3-5 | 4.0 |
| **Overall Mean** | **4.13** | **0.73** | **2-5** | **4.0** |

sized the need for clearer explanations and technology-specific recommendations. Positive feedback validated the system's core value proposition, with participants noting the reports were "very useful for teams without security experts." A detailed analysis of these qualitative findings and their implications for future development is presented in Chapter 7.

Table 6.4: Q10: Participant Suggestions for Improvement.

| **Category** | **Improvement Suggestion** |
|---|---|
| **Process Integration** | Prioritized remediation tasks with assigned owners and effort estimates for immediate actionability<br>Integration with internal tools to automatically pull project planning data into report generation |
| **Output Format** | Output as Markdown file for integration with LLM agents and iterative refinement<br>Format suitable for code analysis tools (Claude Code, Cursor) to connect findings with codebase<br>Executive summary with highlighted concerns for quick stakeholder review |
| **Documentation** | Better descriptions of Security Overview Dashboard axes and figures with overview explanations<br>References to external sources for standards compliance guidance |
| **Customization** | Request programming language and framework information for technology-specific recommendations<br>Explicit questions about software architecture approaches to avoid misleading mainstream assumptions |
| **Presentation** | Unified table ranking countermeasures by urgency and linking to threats (combining sections 9.2, 10.2, 12.3)<br>Case study examples demonstrating successful report usage for confidence building |
| **Multi-Agent Specific** | Concrete patterns for production agentic systems: governance, sandboxing, provenance tracking, CI/CD<br>Operational runbooks for detecting and responding to agent misbehavior and adversarial attacks |
| **General Utility** | Reports demonstrate clear value for teams without dedicated security experts |

# Chapter 7

# Discussion

This chapter interprets the research findings presented in Chapter 6, analyzing them in relation to the research objectives and questions defined in Chapter 1. The discussion is grounded in the context of the literature review (Chapters 2 and 3), specifically highlighting how the developed multi-agent system advances the state of the art in SRE. By synthesizing quantitative performance metrics with qualitative expert feedback, this chapter evaluates the system's effectiveness, architectural robustness, and practical utility. The discussion is structured to address each research question sequentially, followed by an analysis of the study's contributions to the research community and industry practice.

## 7.1 Effectiveness of Automated Security Requirements Translation (RQ1)

The first research objective was to assess the effectiveness of using a multi-agent system to translate high-level business requirements into detailed, security-compliant specifications. The empirical results from the 14 security report generations indicate that the proposed system achieves a high degree of effectiveness, particularly in terms of consistency and alignment with security standards.

The overall quality score of 0.85 across all generations demonstrates that the system reliably bridges the "translation gap" identified by [5] and [14]. A critical finding is the high consistency score (mean = 0.91), which suggests that the multi-agent architecture successfully mitigates the variability and incoherence often associated with stochastic LLMs. This contrasts with the challenges highlighted by [26], who noted that generic LLMs often struggle with maintaining context and logical consistency in complex RE tasks. By decomposing the translation process into specialized sub-tasks (Requirements Analysis, Threat Modeling, Control Mapping), the system maintains a coherent narrative from high-level inputs to technical specifications, effectively acting as a deterministic filter over a probabilistic model.

However, the analysis of quality dimensions revealed a distinct divergence between "Correctness" (0.90) and "Implementability" (0.76). While the agents suc-

cessfully identified valid and necessary security controls, the lower implementability score suggests that automated agents currently lack the deep, tacit organizational context required to prescribe highly specific configuration details (e.g., specific firewall vendor commands or IP subnet ranges). This finding aligns with the insights of [10], who argue that while AI can augment the requirements engineering process, human oversight remains essential for contextualizing requirements to specific infrastructure constraints. The system effectively generates the *what* (the requirement) and the *why* (the threat), evidenced by the 100% verification test coverage, but the specific *how* of implementation often requires human refinement to bridge the "last mile" of deployment.

Furthermore, the system demonstrated a capability to balance breadth and depth, identifying an average of 30 threats per generation with complete STRIDE category coverage. This systematic comprehensiveness addresses the limitations of manual SRE methodologies described by [2] and [33], where human analysts often overlook specific threat categories (such as repudiation or information disclosure) due to cognitive fatigue or lack of specific domain expertise. The result that 12 out of 14 generations identified exactly 30 threats suggests a highly structured application of the STRIDE methodology by the agents. While a human expert might find fewer, more nuanced threats, the automated system provides a rigorous baseline assurance that no major category is neglected, serving as a robust "safety net" for the SRE process.

## 7.2 Architectural Patterns for Robust SRE (RQ2)

The second research question sought to identify the architectural patterns essential for robust automated SRE. The findings strongly validate the efficacy of a hierarchical, multi-agent architecture combined with RAG and state-mediated communication.

The success of the Domain Security Agent in mapping requirements to an average of 75.2 controls across NIST SP 800-53, ISO 27001, and OWASP ASVS standards validates the absolute necessity of RAG in security-critical contexts. Previous studies, such as [41], identified hallucinations as a critical barrier to adopting LLMs for threat modeling. In this study, the integration of a vector database containing over 800 verified security controls ensured that the agents referenced actual, auditable controls rather than hallucinating plausible-sounding but non-existent standards.

This architectural decision offers a significant advantage over model fine-tuning. As security standards evolve (e.g., the transition from ISO 27001:2013 to 2022), a RAG-based system requires only an update to the vector store indices, whereas a fine-tuned model would require expensive retraining. This flexibility addresses the “domain ignorance” challenge raised by [10] and ensures the system remains compliant with the latest regulatory frameworks without architectural

overhaul.

The comparison of this system's architecture with general RE multi-agent frameworks like MARE [17] and KGMAF [15] reveals the importance of domain-specific agent roles. While frameworks like MARE utilize generic roles (e.g., Modeler, Checker), the architecture developed in this thesis introduced strictly defined security roles (e.g., Threat Modeling Agent, Compliance Agent, LLM Security Specialist). The empirical success of these specialized agents supports the theory by [11] that task specialization in MASs leads to improved accuracy by reducing the "cognitive load" on any single agent context window.

Specifically, the uni-directional, state-mediated communication pattern proved essential. By maintaining an immutable state object that agents read from and write to, the system prevented the circular dependencies and agent confusion often reported in unstructured collaborative agent systems [9]. This pattern ensures fully auditable data flows; a critical requirement for security auditing where the provenance of a requirement must be traceable back to its originating threat and business need.

Additionally, the implementation of a self-critique and refinement loop (via the Validation Agent) proved to be a critical architectural component. The ability of the system to self-evaluate outputs against a threshold (0.7) and trigger refinement cycles is an advancement over the static generation pipelines often seen in related work. This aligns with the prompt patterns suggested by [28] but automates the feedback loop entirely within the agent fleet, reducing the need for human-in-the-loop intervention during the initial drafting phase.

## 7.3 Performance, Utility, and Adoption (RQ3)

The third research question evaluated the system's performance and practical utility. Given the lack of a manual control group, RQ3 focuses on the "Perceived Utility" and architectural efficiency of the system rather than a strict A/B performance benchmark. The quantitative results indicate a significant efficiency gain. The parallel execution of analysis agents reduced processing time by approximately 40% (from an estimated 20–25 minutes for sequential execution to 10–15 minutes).

However, the true efficiency gain is most apparent when comparing the automated system to manual processes. A comprehensive security review involving threat modeling (STRIDE), compliance mapping (GDPR, PCI-DSS), and control selection typically consumes days of effort for human security architects [2]. The ability of this system to produce a comprehensive, verified draft in under 15 minutes represents a transformative scalability advantage. It allows security reviews to be conducted per-sprint or even per-feature, rather than only at major release milestones, thereby enabling true "Shift Left" security practices.

The expert evaluation (N=15) provided strong evidence for the system's prac-

tical utility. The high mean scores for "Intent to use" (4.13/5) and "Reliance on Mappings" (4.27/5) indicate that practitioners trust the system's output enough to integrate it into their workflows. This high level of trust in the automated mappings contrasts with the skepticism often found regarding general-purpose LLMs outputs in critical domains [26]. It implies that the system's transparency, achieved through the RAG-backed citation of specific standards, successfully builds user confidence.

Nevertheless, the evaluation also highlighted significant barriers to adoption. The lower score for "incorporation into requirements process" (3.87) and the qualitative feedback suggest that while the *content* is valuable, the *delivery format* requires better integration with existing development tools (e.g., Jira, Azure DevOps, CI/CD pipelines). As noted in the improvement suggestions (Table 6.4), practitioners desire outputs that are directly actionable within their existing ecosystems, rather than static HTML reports. This finding aligns with [18], who emphasize that threat modeling tools must integrate seamlessly into the DevOps pipeline to be effective. The feedback suggests that while the multi-agent system solves the *generation* problem, the *integration* problem remains a valid concern for future development to reduce friction in adoption.

## 7.4 Contributions to the Research Community

This research makes several distinct contributions to the fields of RE and software security:

1. **Bridging the Gap between MAS and SRE:** While existing work has applied MAS to general code generation [27] or vulnerability reproduction [34], this study is among the first to successfully apply a multi-agent architecture specifically to the upstream translation of security requirements. It demonstrates that the collaborative agent patterns described by [22] are highly effective in the security domain when constrained by structured methodologies like STRIDE.

2. **Prototype for Reproducible Research:** The development and release of the open-source prototype hosted on GitHub contributes a tangible artifact to the community. This responds to the call by [35] for more empirical tools in SRE. By providing the source code for the agent orchestration and vector database integration, this thesis offers a platform for other researchers to build upon, test different LLMs, or integrate additional security standards.

3. **Empirical Evidence of RAG in SRE:** The study provides empirical data quantifying the benefits of RAG in requirements engineering. The high accuracy of control mappings (98.6% threat coverage) serves as a benchmark for future studies aiming to reduce hallucinations in automated compliance tasks.

4. **Validation of LLM-based Threat Modeling:** The results confirm that LLM-based agents can effectively perform systematic threat modeling (STRIDE) without human intervention, identifying critical threats (e.g., THR-001, THR-016 in Table 5.2) comparable to those identified by human experts. This extends the work of [41] by integrating the threat modeling phase into a complete end-to-end requirements generation pipeline.

This research confirms that multi-agent systems, when architected with domain-specific roles and grounded knowledge bases, offer a viable solution to the manual bottlenecks of SRE. The system moves beyond theoretical applicability to demonstrate practical utility in industrial scenarios, providing a foundation for the next generation of AI-augmented security tools that empower engineers to build secure systems by design.

## 7.5 Ethical and Societal Considerations

Automating security requirements translation can provide societal benefit by helping teams incorporate security earlier ("shift-left"), potentially reducing vulnerabilities that impact users and organizations. At the same time, the approach introduces ethical and organizational risks that must be managed in practice.

First, requirements documents may contain sensitive or proprietary information; sending such inputs to external model providers can create confidentiality and compliance concerns. This motivates deployment patterns that minimize data exposure (e.g., redaction, strict access controls, and preference for on-premise or private inference where required).

Second, there is a risk of automation bias: engineers may over-trust well-written recommendations, even when context-specific constraints make them incorrect or incomplete. For this reason, the system should be used as decision support rather than an autonomous authority, with explicit human review and accountability for final security requirements and control selection.

Finally, the system could be misused to generate attacker-relevant threat analyses if applied in adversarial contexts. Practical mitigations include authentication/authorization for the tool, careful logging/auditing, and limiting outputs in untrusted settings.

# Chapter 8

# Conclusions and Future Work

This research set out to address the critical "translation gap" in software engineering: the disconnect between high-level business requirements and actionable, security-compliant technical specifications. By designing, implementing, and evaluating a MAS powered by LLMs and grounded via RAG, this thesis has demonstrated that automated, systematic SRE is not only feasible but offers significant efficiency and consistency advantages over manual methods. The primary aim of this study was to automate the translation of requirements using a novel multi-agent architecture. Through the development of a prototype employing ten specialized agents and its evaluation against 14 industrial use cases, we have successfully met the research objectives outlined in Chapter 1.

Regarding **RQ1 (System Effectiveness)**, the results demonstrate that the system translates requirements with a high degree of quality, achieving a mean overall quality score of 0.85 across verified dimensions. The system proved particularly effective in maintaining consistency (0.91) and alignment with security standards (0.89). A crucial finding was that the decomposition of the SRE process into specialized agent roles, such as the Threat Modeling Agent applying STRIDE and the Compliance Agent mapping GDPR, allowed for a depth of analysis that generic LLMs prompting cannot achieve. The system consistently identified critical threats and maintained 100% verification test coverage, effectively creating a robust "safety net" for security architects.

Regarding **RQ2 (Architectural Patterns)**, the empirical evidence validates the necessity of the RAG architecture. By grounding the Domain Security Agent in a vector database containing over 800 verified controls from NIST SP 800-53, ISO 27001, and OWASP ASVS, the system successfully mitigated the hallucination risks typically associated with generative AIs. Furthermore, the unidirectional, state-mediated communication pattern proved essential for stability, ensuring auditability and preventing circular dependencies between agents.

Regarding **RQ3 (Performance and Utility)**, the system demonstrated a transformative impact on efficiency, reducing the time required for comprehensive security analysis compared to manual processes. The expert evaluation confirmed the system's practical utility, with participants reporting a high intent to use the generated insights (4.13/5). However, while the system excelled at defining

*what* needs to be secured and *why*, the lower score in "implementability" (0.76) indicates that bridging the final gap to specific configuration commands remains a challenge.

Overall, the results support the feasibility of a security-focused multi-agent architecture and indicate strong perceived practitioner utility; stronger causal claims (e.g., outperforming human processes) require controlled comparative studies.

## 8.1 Future Work

While this thesis establishes a strong foundation for automated SRE, the evaluation and expert feedback highlighted several avenues for future research and development. A recurring theme in the expert feedback was the need for seamless workflow integration. Future work should focus on moving beyond static report generation to direct integration with project management and CI/CD tools.

- **Issue Tracker Integration:** Developing agents capable of pushing identified requirements directly to Jira or Azure DevOps as user stories with acceptance criteria.
- **IDE Plugins:** Creating lightweight versions of the agents that can run within an IDE (e.g., VS Code) to provide real-time security feedback as developers draft high-level feature descriptions.
- **Comparative Controlled Experiment:** While this thesis established the feasibility of the MAS architecture, future work must establish its superiority through a formal controlled experiment (A/B testing). This would involve a side-by-side comparison of MAS-generated outputs against those produced by human security teams to statistically quantify improvements in completeness and accuracy.

Beyond workflow integration, the evaluation revealed opportunities to address the lower scores in implementability. Future iterations of the system should transition from generating textual recommendations to generating configuration artifacts that can be directly deployed.

- **IaC Generation:** Extending the *Implementation Roadmap Agent* to generate preliminary Terraform, Ansible, or Kubernetes configuration files that enforce the identified security controls.
- **Context-Aware Configuration:** Fine-tuning the agents on specific organizational tech stacks (e.g., AWS-specific vs. Azure-specific controls) to provide copy-paste executable commands rather than generic architectural advice.

While the fully automated pipeline demonstrates efficiency, security engineering requires accountability and human oversight. Future research should investigate "Human-in-the-Loop" architectural patterns. Specifically, a mandatory human review gate should be implemented after Stage 2 (Threat Analysis) and before Stage 3 (Planning). This would allow security architects to calibrate the system's risk appetite and validate identified threats before the system generates the implementation roadmap. This would combine the exhaustive speed of the MAS with the nuanced judgment of human security architects, potentially increasing the adoption rate in highly regulated industries.

Another important consideration concerns the deployment model of the underlying language models. The current implementation relies on external model providers (OpenAI), which raises concerns for organizations handling sensitive security requirements and proprietary business logic. Future work should evaluate the performance of the architecture using open-source, locally hosted LLMs (e.g., Llama 3 or Mistral). Benchmarking the trade-off between the privacy of a local deployment and the reasoning capabilities of larger, cloud-hosted models would be valuable for organizations handling classified or highly sensitive data.

As software systems grow in complexity, the traditional, manual approach to security requirements engineering is becoming a bottleneck. This thesis has demonstrated that a specialized MAS can effectively untangle this complexity, ensuring that security is not an afterthought but a foundational element of the requirements phase. By combining the reasoning power of LLMs with the rigor of structured security frameworks, we move one step closer to a future where secure-by-design is the default standard for software development.

# References


[1] Divyansh Agarwal, Alexander R. Fabbri, Ben Risher, Philippe Laban, Shafiq Joty, and Chien-Sheng Wu. Prompt leakage effect and defense strategies for multi-turn llm interactions, 2024.

[2] Md Tarique Jamal Ansari, Dhirendra Pandey, and Mamdouh Alenezi. Store: Security threat oriented requirements engineering methodology. *Journal of King Saud University - Computer and Information Sciences*, 34(2):191–203, 2022.

[3] Chetan Arora, John Grundy, and Mohamed Abdelrazek. Advancing requirements engineering through generative ai: Assessing the role of llms, 2023.

[4] Prashik Buddhaghosh Bansod. Distinguishing autonomous ai agents from collaborative agentic systems: A comprehensive framework for understanding modern intelligent architectures, 2025.

[5] Sarmad Bashir. Towards ai-centric requirements engineering for industrial systems. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 242–246, 2024.

[6] Shih-Han Chan. Encrypted prompt: Securing llm applications against unauthorized actions, 2025.

[7] Huaben Chen, Wenkang Ji, Lufeng Xu, and Shiyu Zhao. Multi-agent consensus seeking via large language models, 2025.

[8] Badhan Chandra Das, M. Hadi Amini, and Yanzhao Wu. Security and privacy challenges of large language models: A survey, 2024.

[9] Christian Schroeder de Witt. Open challenges in multi-agent security: Towards secure systems of interacting ai agents, 2025.

[10] Moemen Ebrahim, Shawkat Guirguis, and Christine Basta. Enhancing software requirements engineering with language models and prompting techniques: Insights from the current research and future directions. In *Proceedings of the 63rd Annual Meeting of the Association for Computational*

*Linguistics (Volume 4: Student Research Workshop)*, pages 486–496, Vienna, Austria, 07 2025. Association for Computational Linguistics.

[11] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges, 2024.

[12] Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, and Zhaozhuo Xu. Llm multi-agent systems: Challenges and open problems, 2025.

[13] Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead, 2025.

[14] Arshia Hemmat, Mohammadreza Sharbaf, Shekoufeh Kolahdouz-Rahimi, Kevin Lano, and Sobhan Y. Tehrani. Research directions for using llm in software requirement engineering: a systematic review. *Frontiers in Computer Science*, Volume 7 - 2025, 2025.

[15] Jiangping Huang, Dongming Jin, Weisong Sun, Yang Liu, and Zhi Jin. Knowledge-guided multi-agent framework for automated requirements development: A vision, 2025.

[16] Fengqing Jiang, Zhangchen Xu, Luyao Niu, Boxin Wang, Jinyuan Jia, Bo Li, and Radha Poovendran. Identifying and mitigating vulnerabilities in llm-integrated applications, 2023.

[17] Dongming Jin, Zhi Jin, Xiaohong Chen, and Chunhui Wang. Mare: Multi-agents collaboration framework for requirements engineering, 2024.

[18] Dimitri Van Landuyt, Laurens Sion, Walewein Philips, and Wouter Joosen. From automation to ci/cd: a comparative evaluation of threat modeling tools. In *2024 IEEE Secure Development Conference (SecDev)*, pages 35–45, 2024.

[19] Dawei Li, Zhen Tan, Peijia Qian, Yifan Li, Kumar Satvik Chaudhary, Lijie Hu, and Jiayi Shen. Smoa: Improving multi-agent large language models with sparse mixture-of-agents, 2024.

[20] Junyuan Mao, Fanci Meng, Yifan Duan, Miao Yu, Xiaojun Jia, Junfeng Fang, Yuxuan Liang, Kun Wang, and Qingsong Wen. Agentsafe: Safeguarding large language model-based multi-agent systems via hierarchical data management, 2025.

[21] Mariana Martins, Taciana Kudo, and Renato Neto. A qualitative study on requirements engineering practices in an artificial intelligence unit of the brazilian industrial research and innovation company. pages 46–60, 05 2024.

[22] Viviana Mascardi, Danny Weyns, and Alessandro Ricci. Engineering multi-agent systems: State of affairs and the road ahead. *SIGSOFT Softw. Eng. Notes*, 44(1):18–28, March 2019.

[23] Giovane Mendonça, Iderli Pereira De Filho, and Gilleanes Guedes. A systematic review about requirements engineering processes for multi-agent systems. pages 69–79, 01 2021.

[24] Polina Minina, Andrey Sadovykh, and Vladimir Ivanov. Detecting security requirements in github issues -novel dataset and smallbert-based model. In *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 315–318, 2023.

[25] Thu-Trang Nguyen, Thanh Trong Vu, Hieu Dinh Vo, and Son Nguyen. An empirical study on capability of large language models in understanding code semantics, 2024.

[26] Johannes J. Norheim, Eric Rebentisch, Dekai Xiao, Lorenz Draeger, Alain Kerbrat, and Olivier L. de Weck. Challenges in applying large language models to requirements engineering tasks. *Design Science*, 10:e16, 2024.

[27] Ana Nunez, Nafis Tanveer Islam, Sumit Kumar Jha, and Peyman Najafirad. Autosafecoder: A multi-agent framework for securing llm code generation through static analysis and fuzz testing, 2024.

[28] Krishna Ronanki, Simon Arvidsson, and Johan Axell. Prompt engineering guidelines for using large language models in requirements engineering, 2025.

[29] Beata Smela, Mondher Toumi, Karolina Świerk, Clement Francois, Małgorzata Biernikiewicz, Emilie Clay, and Laurent Boyer. Rapid literature review: Definition and methodology. *Journal of Market Access & Health Policy*, 11(1), 2023.

[30] Martin Trae Span, Gabe Salinger, Mars Rayno, and Jeremy Daily. Security requirements engineering: A survey for the systems engineer. In *2024 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–8, 2024.

[31] Olga Springer and Jakub Miler. A comprehensive overview of software product management challenges. *Empirical Software Engineering*, 27, 09 2022.

[32] Karthik Sreedhar and Lydia Chilton. Simulating human strategic behavior: Comparing single and multi-agent llms, 2024.

[33] R. Subha and Anandakumar Haldorai. An efficient identification of security threats in requirement engineering methodology. *Computational Intelligence and Neuroscience*, 2022:1–14, 08 2022.

[34] Saad Ullah, Praneeth Balasubramanian, Wenbo Guo, Amanda Burnett, Hammond Pearce, Christopher Kruegel, Giovanni Vigna, and Gianluca Stringhini. From cve entries to verifiable exploits: An automated multi-agent framework for reproducing cves. 2025.

[35] Hugo Villamizar, Marcos Kalinowski, Marx Viana, and Daniel Méndez Fernández. A systematic mapping study on security in agile requirements engineering. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 454–461, 2018.

[36] Liyuan Wang, Jingyi Xie, Xingxing Zhang, Mingyi Huang, Hang Su, and Jun Zhu. Hierarchical decomposition of prompt-based continual learning: Rethinking obscured sub-optimality, 2023.

[37] Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. G-safeguard: A topology-guided security lens and treatment on llm-based multi-agent systems, 2025.

[38] Yingming Wang and Pepa Atanasova. Self-critique and refinement for faithful natural language explanations, 2025.

[39] Bruce Warrens. The challenge of mapping security requirements to standards, May 2025.

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[41] Tingmin Wu, Shuiqiao Yang, Shigang Liu, David Nguyen, Seung Jang, and Alsharif Abuadbba. Threatmodeling-llm: Automating threat modeling using large language models for banking system, 2025.

[42] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

[43] Xian Yeow, Shunichi Lee, Lasitha Akatsuka, Vidyaratne Aman, Ahmed Kumar, Chetan Farahat, and Gupta. Reliable decision-making for multi-agent llm systems.

[44] Xinkai Zou, Yan Liu, Xiongbo Shi, and Chen Yang. Goal2story: A multi-agent fleet based on privately enabled sllms for impacting mapping on requirements elicitation, 2025.

[45] Muhammad Ihsan Zul, Suhaila Mohd Yasin, and Dadang Syarif Sihabudin Sahid. Exploring requirement engineering challenges in software development: Insights from global and indonesian landscape. In *2024 4th International Conference on Electrical Engineering and Informatics (ICon EEI)*, pages 136–141, 2024.

# Appendix A

# Example Input Document

```
E-Commerce Platform Requirements

We need to build a modern e-commerce platform for selling consumer
electronics online.

Key Features:

1. User Management
   - User registration and login with email/password
   - Social login (Google, Facebook)
   - User profiles with shipping addresses and payment methods
   - Order history tracking

2. Product Catalog
   - Browse products by category
   - Search functionality with filters
   - Product details with images and specifications
   - Customer reviews and ratings

3. Shopping Cart & Checkout
   - Add/remove items from cart
   - Apply discount codes and promotions
   - Multiple payment methods (credit card, PayPal, Apple Pay)
   - Guest checkout option
   - Order confirmation emails

4. Payment Processing
   - Secure payment processing with Stripe integration
   - Save payment methods for future use
   - Handle refunds and chargebacks

5. Admin Dashboard
```

```
   - Product inventory management
   - Order management and tracking
   - Customer service tools
   - Sales analytics and reporting

6. AI Features
   - Product recommendations based on browsing history
   - Chatbot for customer support
   - Automated product categorization

7. Additional Requirements
   - Mobile-responsive design
   - Support for multiple currencies
   - Integration with shipping carriers (FedEx, UPS)
   - Email notifications for order updates
   - Customer data for marketing campaigns
   - Store customer behavior analytics

The platform will primarily serve customers in the EU and US markets.
```

# Appendix B

# Traceability Matrix

Table B.1: Complete traceability matrix mapping requirements to threats and security controls.

| Req ID | Category | Sensitivity | Threat IDs | Security Controls | Priority |
|---|---|---|---|---|---|
| REQ-001 | Authentication | High | 1, 2, 4, 7, 8, 9, 10, 11, 20, 22, 23, 28 | [OWASP] 2.2.1, [OWASP] 2.6.1, [NIST] IA-2, [ISO27001] A.9.4.2 | Critical |
| REQ-002 | Authentication | Medium | 2, 12, 18, 20, 24, 25 | [OWASP] 2.7.1, [OWASP] 2.7.3, [ISO27001] A.9.4.2, [NIST] IA-2 | Critical |
| REQ-003 | Data Management | High | 1, 2, 4, 7, 9, 10, 22, 25, 26, 27 | [OWASP] 1.2.1, [NIST] SC-28, [ISO27001] A.8.2.1, [OWASP] 4.2.1 | Critical |
| REQ-004 | Order Management | Medium | 1, 5, 6, 7, 9, 10, 11, 14, 17, 22 | [OWASP] 4.2.1, [OWASP] 9.1.1, [NIST] AU-2 | Critical |
| REQ-005 | Checkout | Medium | 5, 14, 16, 20, 28 | [OWASP] 3.1.1, [OWASP] 3.4.1, [NIST] SC-23 | High |
| REQ-006 | Catalog Management | Low | 4, 7, 8, 10, 21, 22 | [ISO27001] A.12.1.2, [NIST] CM-3, [OWASP] 4.1.2 | High |
| REQ-007 | Search | Low | 14, 15, 27 | [OWASP] 5.3.2, [OWASP] 7.1.1, [NIST] SI-10 | High |
| REQ-008 | Catalog Display | Low | 4, 8, 10, 16 | [OWASP] 5.1.3, [OWASP] 10.1.2, [NIST] SI-7 | High |
| REQ-009 | Content Management | Medium | 4, 9, 10, 14, 17, 22 | [OWASP] 5.3.3, [OWASP] 7.3.1, [NIST] AU-6 | High |
| REQ-010 | Cart | Medium | 1, 4, 5, 7, 9, 10, 11, 14, 20, 22 | [OWASP] 3.5.1, [OWASP] 4.3.1, [ISO27001] A.14.1.3 | High |
| REQ-011 | Promotions | Medium | 5, 19 | [OWASP] 5.3.2, [OWASP] 7.1.1, [NIST] AU-12 | High |
| REQ-012 | Payment Processing | High | 1, 5, 6, 7, 9, 10, 11, 14, 20, 22, 25 | [OWASP] 9.1.2, [OWASP] 9.3.1, [NIST] SC-12 | Critical |
| REQ-013 | Payment Processing | High | 1, 2, 4, 7, 9, 10, 11, 20, 22, 25 | None | Medium |

Continued on next page

**Table B.1 – continued from previous page**

| Req ID | Category | Sensitivity | Threat IDs | Security Controls | Priority |
|---|---|---|---|---|---|
| REQ-014 | Payment Processing | High | 1, 8, 11, 14, 19, 21, 22, 26 | [NIST] AU-3, [OWASP] 7.3.2, [ISO27001] A.12.4.1 | High |
| REQ-015 | Notifications | Medium | 1, 5, 6, 9, 12, 20, 22 | [NIST] SC-8, [ISO27001] A.13.2.1, [OWASP] 7.2.1 | Medium |
| REQ-016 | Administration | High | 1, 4, 5, 7, 8, 9, 10, 11, 14, 21, 22, 23 | [OWASP] 4.1.2, [OWASP] 4.2.3, [NIST] AC-6 | Critical |
| REQ-017 | Order Management | High | 1, 3, 4, 5, 7, 9, 10, 11, 14, 22 | [OWASP] 4.3.3, [NIST] AU-2, [ISO27001] A.12.4.3 | Critical |
| REQ-018 | Customer Service | High | 1, 2, 3, 4, 7, 9, 10, 11, 22 | [OWASP] 9.1.1, [OWASP] 4.2.1, [NIST] AU-12 | High |
| REQ-019 | Analytics | Medium | 1, 3, 7, 9, 10, 14, 17, 22, 27 | [ISO27001] A.9.4.1, [OWASP] 9.2.1, [NIST] MP-5 | High |
| REQ-020 | AI/ML | Medium | 1, 4, 8, 17, 22 | [OWASP] 1.2.2, [NIST] AR-2, [OWASP] 9.1.3 | High |
| REQ-021 | AI/ML | High | 4, 9, 10, 14, 16, 18, 22, 26 | [OWASP] 7.3.1, [OWASP] 5.3.2, [ISO27001] A.14.2.1 | High |
| REQ-022 | AI/ML | Low | 4, 6, 7, 9, 10, 14, 17, 22 | [OWASP] 10.2.1, [ISO27001] A.12.1.2, [OWASP] 1.3.1 | High |
| REQ-023 | User Experience | Low | 28 | [OWASP] 9.2.1, [OWASP] 10.1.2, [NIST] SC-18 | High |
| REQ-024 | Localization & Finance | Medium | None | None | Medium |
| REQ-025 | Integrations | Medium | 2, 4, 10, 12, 14, 16, 22, 25 | [OWASP] 10.3.1, [NIST] SA-9, [ISO27001] A.15.2.1 | High |
| REQ-026 | Notifications & Marketing | Medium | 1, 12, 20, 26 | [ISO27001] A.18.1.4, [OWASP] 1.2.2, [NIST] AP-1 | High |
| REQ-027 | Marketing Data | High | 6, 9, 10, 14, 17, 22, 25, 27 | [OWASP] 1.2.1, [NIST] IP-1, [ISO27001] A.18.1.3, [OWASP] 4.2.1 | High |
| REQ-028 | Analytics | Medium | 1, 4, 6, 9, 10, 14, 17, 22, 27 | [NIST] PT-2, [ISO27001] A.18.1.4, [OWASP] 1.2.2 | High |
| REQ-029 | Operations & Security | High | 1, 2, 3, 7, 9, 10, 14, 22, 23, 26 | [NIST] AU-6, [ISO27001] A.12.4.1, [OWASP] 7.3.3, [NIST] SI-4 | Critical |
| REQ-030 | Authorization | High | 1, 3, 4, 7, 8, 9, 10, 11, 14, 22, 23 | [NIST] AU-3, [OWASP] 7.3.2, [ISO27001] A.12.4.1 | High |
| REQ-031 | Compliance | High | 5, 6, 9, 10, 14, 17, 22, 25, 27 | [OWASP] 1.3.2, [NIST] DM-2, [ISO27001] A.18.1.4, [OWASP] 4.2.1 | Critical |
| REQ-032 | Integrations & Governance | High | 1, 2, 3, 7, 9, 10, 11, 14, 22, 23, 25 | [OWASP] 10.3.2, [NIST] SC-12, [ISO27001] A.9.2.4, [OWASP] 7.3.2 | Critical |