SYNTACTIC AMBIGUITY DETECTION FRAMEWORK FOR SOFTWARE REQUIREMENTS SPECIFICATION

BY

SITI SYARA AIMAN BINTI SEH WALI

A thesis submitted in fulfillment of the requirement for the degree of Master of Computing (Computer Science and Information Technology).

Kulliyyah of Information and Communication Technology
International Islamic University Malaysia

July 2025

ABSTRACT

In software development, clear and precise requirement specifications are essential for project success. However, ambiguities in software requirements often cause misunderstandings that lead to costly errors and delays. Among the various types of ambiguity, syntactic ambiguity arising from sentence structure poses a significant challenge in requirement specifications.

This research introduces the Syntactic Ambiguity Detection Framework (SADF), specifically designed to identify and resolve syntactic ambiguities in software requirement documents. The framework adopts a multi-layered approach that combines linguistic analysis with practical heuristics, developed through a comprehensive review of existing ambiguity detection techniques and related ambiguity knowledge. This review examined various aspects of syntactic ambiguity explored in prior studies, highlighting both the strengths and limitations of current methods. These insights guided the design and development of SADF.

To evaluate the framework's effectiveness, a questionnaire survey was conducted involving experienced Requirement Engineering experts. Their valuable feedback contributed to refining the framework's components and confirmed its practical utility in guiding engineers to detect syntactic ambiguities more effectively in requirement documents. This collaborative validation process ensured that SADF not only performs theoretically but also meets the practical needs of professionals in the field.

Therefore, the main objective of this study is to develop a framework that can help Requirement Engineers with a systematic and user-friendly guideline which facilitates early detection and resolution of syntactic ambiguities, thereby improving the quality of Software Requirement Specification (SRS). Implementing SADF is expected to reduce misunderstandings, minimize costly rework, and streamline the software development life cycle ultimately contributing to more successful project outcomes.

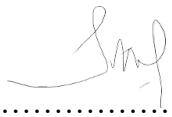
In conclusion, the SADF addresses a critical issue in requirements engineering by emphasizing syntactic clarity. The framework developed in this study offers structured guidelines and rules for identifying syntactic ambiguities in Software Requirements Specifications (SRS) and its adoption may help to enhance the quality of requirements specifications and support the overall success and efficiency of software projects.

ملخص البحث

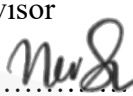
في تطوير البرمجيات، تُعد مواصفات المتطلبات الواضحة والدقيقة من العوامل الأساسية لنجاح المشروع. ومع ذلك، فإن الغموض في صياغة المتطلبات كثيراً ما يؤدي إلى سوء فهم مما يُسفر عن أخطاء مكلفة وتأخيرات في إنجاز المشاريع. من بين أنواع الغموض المختلفة، يُعد الغموض التركيبي الناتج عن بنية الجملة من أكثر التحديات تعقيدًا في وثائق متطلبات البرمجيات. يقدم هذا البحث إطار عمل يُعرف باسم إطار الكشف عن الغموض التركيبي (SADF)، وهو مصمم خصيصًا للكشف عن الغموض التركيبي والتعامل معه في وثائق متطلبات البرمجيات. يعتمد الإطار المقترح على نهج متعدد الطبقات يجمع بين التحليل اللغوي والاستدلال العملي، وقد تم تطويره بناءً على مراجعة شاملة للأبحاث السابقة والتقنيات الحالية في مجال الكشف عن الغموض. لتقييم فعالية إطار SADF، تم إجراء دراسة استقصائية باستخدام استبيان شارك فيه عدد من خبراء هندسة المتطلبات ذوي الخبرة. وقد ساعدت ملاحظاتهم وتعليقاتهم القيّمة في تحسين تصميم الإطار وتأكيد جدواه العملية، مما يضمن توافقه مع احتياجات الممارسين في المجال الواقعي. أثبتت نتائج التقييم أن SADF يمكن أن يساهم بشكل ملموس في دعم مهندسي المتطلبات على اكتشاف الغموض التركيبي بشكل منهجي وفعّال منذ المراحل المبكرة من تطوير البرمجيات. الهدف الرئيس من هذا الإطار هو توفير أداة إرشادية سهلة الاستخدام لمهندسي المتطلبات، تساعدهم على رفع جودة وثائق متطلبات البرمجيات من خلال تقليل الغموض، وبالتالي تقليل الحاجة لإعادة العمل المكلف، وتسريع دورة حياة تطوير البرمجيات. ختامًا، يعالج إطار SADF مشكلة حرجة في هندسة المتطلبات من خلال تعزيز الوضوح التركيبي، مما يساهم في دعم كفاءة المشاريع البرمجية ورفع احتمالية نجاحها.

APPROVAL PAGE

I certify that I have supervised and read this study and that in my opinion, it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Computing (Computer Science and Information Technology).



…….
Azlin Binti Nordin
Supervisor



…….
Norsaremah Binti Salleh
Co-Supervisor

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Computing (Computer Science and Information Technology).

…….
Norzariyah Binti Yahya
Examiner

…….
Haslina Binti Mohd
External Examiner

This thesis was submitted to the Department of Computer Science and is accepted as a fulfillment of the requirement for the degree of Master of Computing (Computer Science and Information Technology).

…….
Azlin Binti Nordin
Head, Department of Computer Science

This thesis was submitted to the Kulliyyah of Information and Communication Technology and is accepted as a fulfillment of the requirement for the degree of Master of Computing (Computer Science and Information Technology).

…….
Tengku Mohd Bin Tengku Sembok

Dean, Kulliyyah of Information and Communication Technology (KICT)

DECLARATION

I hereby declare that this thesis is the result of my own investigations, except where otherwise stated. I also declare that it has not been previously or concurrently submitted as a whole for any other degrees at IIUM or other institutions.

Siti Syara Aiman Binti Seh Wali

Signature…… Date: 25 July 2025

COPYRIGHT

INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA

DECLARATION OF COPYRIGHT AND AFFIRMATION OF FAIR USE OF UNPUBLISHED RESEARCH

SYNTACTIC AMBIGUITY DETECTION FRAMEWORK FOR SOFTWARE REQUIREMENTS SPECIFICATION

I declare that the copyright holder of this thesis/thesis are jointly owned by the student and IIUM.

No part of this unpublished research may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior written permission of the copyright holder except as provided below

1. Any material contained in or derived from this unpublished research may only be used by others in their writing with due acknowledgement.

2. IIUM or its library will have the right to make and transmit copies (print or electronic) for institutional and academic purpose.

3. The IIUM library will have the right to make, store in a retrieval system and supply copies of this unpublished research if requested by other universities and research libraries.

By signing this form, I acknowledged that I have read and understand the IIUM Intellectual Property Right and Commercialization policy.

Affirmed by Siti Syara Aiman Binti Seh Wali

.....

Signature Date: 25 July 2025

This thesis is dedicated to the people who have supported me throughout my education
Thanks for making me see this adventure through the end.

ACKNOWLEDGEMENTS

All praise and gratitude are due to Allah, the Almighty, whose boundless grace and mercy have sustained me throughout the duration of my study. Though the journey was demanding, His blessings made the completion of this thesis possible.

I am profoundly grateful to my supervisor, Asst. Prof. Dr. Azlin Binti Nordin, for her unwavering support, patience, and guidance. Her insightful comments, constructive suggestions, and thoughtful queries have greatly enriched this work. Despite her busy schedule, she generously dedicated time to mentor me, providing both academic and moral support that was invaluable throughout this research. I also extend my sincere thanks to my co-supervisor, Prof. Dr. Norsaremah Salleh, whose encouragement and cooperation have been essential to the successful completion of this study.

My deepest appreciation goes to my beloved husband for his prayers, understanding, and constant encouragement throughout this journey. I am also sincerely thankful to my parents and parent-in-law for their unconditional love, support, and sacrifices, which have been a source of strength and motivation.

Once again, I praise Allah for His infinite mercy, which has enabled me to accomplish this milestone. Alhamdulillah.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 OVERVIEW

This chapter is divided into six sections which are research background, problem statement, research question, research objective, research scope, and research significance.

1.2 RESEARCH BACKGROUND

The aim of this research is to ease requirements engineers to detect and reduce syntactic ambiguity in Software Requirement Specification (SRS). According to Pandey, Suman, and Ramani (2010), requirement engineers play a key role in documenting system requirements during the early stages of software development, ensuring that stakeholder needs are accurately captured and formally recorded. Before the software is developed, it should undergo a process called Requirement Engineering (RE). As stated by Shah, Tejas & Patel (2014) in their recent studies, RE is the process of discovering the needs of stakeholders, understanding the context of requirements, modeling, negotiating, validating, documenting, and managing these requirements.

As highlighted by Pandey et al., (2010), the RE process consists of four stages which are known as follows:

1) requirement elicitation and development
2) documentation of requirements
3) validation and verification of requirements
4) requirement management and planning

RE is regarded as one of the major stages in the rest of whole software development because an ineffective RE will lead to a risk for software project failure. A study by Hussain, Ahmad, Khan, and Akbar (2023) highlight the importance of RE in determining the success of failure of software projects. Authors conducted a systematic literature review and found that poor requirements gathering is a leading cause of software project failure, often due to unclear objectives and scopes, and insufficient stakeholder communication. Their review also reported that requirements related issues such as ambiguity, incompleteness, and uncontrolled changes account for a significant proportion of project delays and budget overruns. These issues typically lead to projects that fail to deliver expected value or incur higher costs when addressed late in the development lifecycle. The authors emphasize that inadequate documentation, lack of effective stakeholder engagement, and weak requirements validation processes remain primary contributors to project failure.

In the initial stage of RE, which is requirement elicitation, the requirements and features of the system are gathered before they are documented in a technical document recognized as SRS. It is crucial to document the SRS in a correct way because it will affect the overall process of the system development. The SRS should contain a complete description of the software behavior, and it must satisfy the needs of users of a system. However, there is a common problem where requirement engineers may be facing during the phase of documenting the requirements. In most software projects, RE will encounter ambiguity primarily because the requirements are typically expressed in natural language, chosen for its ease of comprehension. However, Gavran, Štemberger, and Orehek, (2021) that the natural language, while widely used for requirements specifications, remains inherently ambiguous and prone to misinterpretation, which can lead to inconsistencies and errors in software development. A study by Damian, Zowghi, and Neill (2020) report that most requirements specifications are still expressed in common natural language, with structured and formalized languages representing only a small fraction. Specifically, their survey indicates that approximately 75-80% of requirements documents use unrestricted

natural language, while structured natural language and formal specification languages account for much lower percentages. Nevertheless, though the natural language is easy to understand by the layman, ambiguous expressions, which may interpret the meaning of the sentence in multiple ways will exist, thus, will negatively affect the entire quality of software projects.

A study by Sabriye and Zainon (2018), emphasized that among the various forms of ambiguity, syntactic ambiguity is especially prevalent. Syntactic ambiguity arises from the grammatical structure or sentence construction, where a single requirement can be interpreted in more than one way due to unclear phrasing, misplaced modifiers, or ambiguous conjunctions. For example, a sentence like "The system shall generate reports for users with high priority" could imply either that the system prioritizes reports or that the users themselves have high priority. Such structural ambiguities often go unnoticed but can lead to serious misunderstandings during development and testing. As a result, they reduce the clarity, consistency, and overall quality of the SRS, which in turn can negatively impact the success of the software project.

Therefore, it is essential to address syntactic ambiguity early in the RE process to help requirements engineers detect and resolve these issues before they propagate into later stages of development.

1.3 PROBLEM STATEMENT

According to Hayman et al., (2018), ambiguity is one of the major problems in SRS. A recent study confirms that ambiguity remains one of the major problems in SRS. Hussain, Ahmed, Khamaj and Nawi (2021) emphasized that ambiguous requirements can lead to inconsistencies and misinterpretations, which negatively impact software project outcomes such as failure in testing. Similarly, Ribeiro and Berry (2020) found that persistent ambiguity in SRS is widespread and often overlooked, contributing significantly to project

risks and failures. These findings align with the view that ambiguity in SRS continues to be a critical challenge affecting the quality and success of software development projects. In software development, the presence of ambiguity within SRS poses a significant challenge to developers and stakeholders alike. This is because natural language (NL), which is commonly used to specify SRS, is generally ambiguous. Ambiguities arising from imprecise or unclear language within SRS documents can lead to misinterpretation, resulting in flawed software designs or software failures, increased development time, and costly rework.

Despite numerous approaches proposed to detect and resolve syntactic ambiguity, requirement engineers often face confusion in selecting the most suitable method, which hinders their ability to document high-quality, unambiguous requirements. This confusion arises from the lack of comprehensive guidelines and empirical validation of existing frameworks. Besides, there are also a few existing models and frameworks in detecting ambiguity in SRS. Notable frameworks like the Software Requirement Ambiguity Avoidance Framework (SRAAF) by Gupta and Deraman (2019) and the DARA framework by Osama and Aref (2018) have attempted to address ambiguity. However, these frameworks present several gaps. SRAAF lacks empirical validation and detailed, structured processes for technique selection, while DARA is limited to specific types of ambiguities and does not provide extensive comparative analysis or user experience insights. Additionally, frameworks such as the one proposed by Sabriye and Wan Zainon (2017) and the unifying framework by Gervasi, Ferrari, Zowghi and Spoletini (2019) offer valuable perspectives but fail to detail the technical implementation, empirical validation, and practical integration into existing workflows.

Therefore, there is a critical need for a novel, comprehensive framework that not only encompasses a broader range of syntactic ambiguity types but also provides clear guidelines, empirical validation, and practical integration strategies. This research proposes to develop such a framework, aiming to enhance the detection and resolution of syntactic

ambiguities in SRS. The proposed framework will be validated by RE experts and evaluated through empirical studies to ensure its effectiveness and applicability in real-world settings.

By addressing these gaps, this research aims to assist requirement engineers in avoiding and reducing ambiguous requirements, thereby improving the overall quality of software development processes and outcomes.

1.4 RESEARCH QUESTION

The research questions of this work are as follows:

1. What are the components that contribute to detecting syntactic ambiguities in SRS?
2. How to develop a framework that can be used to detect syntactic ambiguity?
3. How to evaluate the framework?

1.5 RESEARCH OBJECTIVE

The main objective of this study is to develop a framework that can help Requirement Engineers with a systematic and user-friendly guideline which facilitates early detection and resolution of syntactic ambiguities, thereby improving the quality of Software Requirement Specification (SRS). Syntactic ambiguity, which results from grammatical constructions or structural ambiguities, can cause stakeholders to understand things differently, which can eventually impact the quality of software and project outputs.

The study proposes developing a structured framework that offers thorough guidance for identifying such ambiguities throughout the requirements engineering stage to address this problem.

The objectives of this study include:

1. to identify main components to detect syntactic ambiguity in SRS.
2. to develop a comprehensive and practical framework for detecting syntactic ambiguity in SRS.
3. to validate the propose framework using expert review approach.

Table 1: Research Problem, Question, Objective

Research Problem	Research Question	Research Objective
Previous studies have focused solely on a single approach within the framework and have not provided clear or comprehensive guidelines.	What are the components that contribute to detecting syntactic ambiguities in SRS?	To propose a framework that can guide in detecting syntactic ambiguity in SRS. to identify main components to detect syntactic ambiguity in SRS.
Lack of a structured method or comprehensive framework that integrates various components necessary for effective syntactic ambiguity detection.	How to develop a framework that can be used to detect syntactic ambiguity?	To develop a comprehensive and practical framework for detecting syntactic ambiguity in SRS.
No validation on the effectiveness of the proposed	How to evaluate the framework?	To validate the proposed framework.

framework has been made by human experts.		

This study addresses several critical gaps identified in existing research on syntactic ambiguity detection in Software Requirements Specifications (SRS). First, previous studies have predominantly focused on isolated approaches within the framework and have not provided clear, comprehensive guidelines for ambiguity detection. Consequently, this raises the research question: What are the components that contribute to detecting syntactic ambiguities in SRS? To address this, the objective is to propose a well-structured framework that guides the identification of syntactic ambiguities effectively.

Second, there exists a lack of an integrated, systematic framework that combines the essential components necessary for comprehensive and practical syntactic ambiguity detection. This limitation prompts the research question: How can a framework be developed to detect syntactic ambiguity effectively? Accordingly, the objective is to design and develop a comprehensive framework that can be practically applied to detect syntactic ambiguities within SRS.

Finally, the effectiveness of existing frameworks has not been validated through expert evaluation, which is essential to establish their practical utility. This leads to the research question: How can the proposed framework be evaluated to ensure its validity and effectiveness? The objective, therefore, is to conduct a rigorous validation of the developed framework through expert assessment via survey to confirm its applicability and reliability.

1.6 RESEARCH SCOPE

This research primarily focuses on the development and evaluation of a framework designed as a guideline to detect syntactic ambiguity in Software Requirements Specification (SRS). The scope is limited to identifying ambiguities that arise from

syntactic structures, such as vague modifiers, misplaced punctuation, parallelism errors, or ambiguous sentence constructions, rather than addressing lexical, semantic, or pragmatic ambiguities. The framework targets English-language SRS and assumes that these documents are written using natural language, not formal specification languages.

The study includes a review of existing ambiguity detection techniques, formulation of a set of syntactic rules and patterns, and the integration of these into a structured detection framework. Validation of the framework is conducted through expert evaluation involving practitioners and academics in requirements engineering. However, the implementation of the framework in a real software development environment or its integration with automated requirement management tools is beyond the scope of this research. A summary of the research scope is provided in the table below.

Table 2: Summary of Research Scope

Aspect	Scope Description	Explanation
Primary Focus	Development and evaluation of a framework for detecting syntactic ambiguity.	The research centers on creating a tool or method to help identify confusing sentence structures in SRS.

Type of Ambiguity Covered	Focus is limited to syntactic ambiguity, such as: 1. Vague modifiers 2. Misplaced punctuation 3. Parallelism errors 4. Ambiguous sentence structures	Only grammatical or sentence-structure-related ambiguities are addressed, not those based on word meaning or interpretation.
Excluded Ambiguity Types	Does not cover lexical, semantic, or pragmatic ambiguities.	Other types of ambiguities—like unclear word choices (lexical) or contextual confusion (semantic/pragmatic) are intentionally left out.
Language Scope	Targets English-language SRS documents written in natural language (not formal specification languages).	The framework is designed for typical requirement documents written in plain English, not for formal, mathematical-style notations.
Framework Components	1. Review of existing ambiguity detection methods. 2. Formulation of syntactic rules and patterns. 3. Integration into a structured detection framework	The research involves studying existing approaches, creating new syntactic rules, and organizing them into a usable framework.

Validation Method	Conducted via expert evaluation, involving practitioners and academics in the RE field.	The effectiveness of the framework will be tested by collecting feedback from professionals with expertise in software requirements.
Out of Scope	The implementation of the framework in real software development environments. Integration with automated requirement management tools	The research stops short of deploying the framework in real software projects or linking it with tools like Jira, DOORS, or IBM Rational.

1.7 RESEARCH SIGNIFICANCE

Zafar et al. (2021) discuss challenges in requirements engineering in agile projects, including ambiguity causing communication breakdown and rework throughout development stages. Syntactic ambiguity can lead to serious confusion among stakeholders, particularly developers and testers, but is frequently overlooked during early requirement analysis. Early resolution of this issue during the requirements phase improves communication between development teams and clients while also increasing clarity.

This study makes a significant contribution to RE techniques in both research and industry by putting out a framework to detect syntactic ambiguity. The following are the significant consequences of this study:

1. To assist requirements engineers or practitioners to reduce ambiguities in documenting SRS by providing guidelines to detect ambiguities.
2. Benefiting the developer to produce a good quality software application that meets the client's expectation.

3. To reduce the cost of correcting errors in the software.

Detecting ambiguities early in the requirements phase helps prevent misunderstandings that could lead to defects later in the development cycle. Fixing errors at later stages (e.g., during testing or after deployment) is much more costly and time-consuming. This study helps identify ambiguities early, thus lowering the risk of expensive rework.

The framework developed in this study offers structured guidelines and rules for identifying syntactic ambiguities in Software Requirements Specifications (SRS). This directly supports requirements engineers and practitioners in producing clearer, more precise documentation, which improves communication among stakeholders. When the requirements are clear and unambiguous, developers have a better understanding of what the client wants. This reduces the chances of misinterpretation and helps ensure that the final software product aligns with user needs and expectations, ultimately improving software quality and client satisfaction.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter offers a depth of insight of the previous works related to the approaches to detect ambiguities in SRS. This chapter is divided into 5 sections, which include type of requirement ambiguity and type of syntactic ambiguity, existing techniques in reducing or solving ambiguities, existing frameworks to detect ambiguities, ambiguity detector, and knowledge on ambiguity.

2.2 AMBIGUITY IN SOFTWARE REQUIREMENTS SPECIFICATION

Ambiguity in SRS remains a persistent challenge for both researchers and practitioners in software engineering. IEEE 830 (1998) and ISO/IEC/IEEE 29148 (2018) define ambiguity in a Software Requirements Specification (SRS) as a requirement that can be interpreted in more than one way, leading to inconsistent understanding among stakeholders. Both standards emphasize that requirements must be unambiguous, ensuring each statement has only one possible interpretation to support clarity, consistency, and verifiability. One of the primary reasons is that requirements are typically expressed in natural language, which is inherently prone to vagueness and multiple interpretations. As Ashfaq et al. (2021) highlight, unclear or ambiguous requirements can lead to varying interpretations among stakeholders, resulting in inconsistencies during development and potentially costly rework. For example, a requirement such as “the system shall respond quickly” is ambiguous as what qualifies as “quick” may vary from one stakeholder to another.

Kwizera (2025), describes ambiguity as the occurrence of multiple interpretations of a statement due to natural language's inherent flexibility and contextual gaps, leading to uncertainty and misunderstanding among stakeholders. This echoes the earlier formalization by Ceccato, Kiyavitskaya, Zeni, Mich, and Berry (2004), ambiguity is defined as the capability of being understood in two or more possible senses or ways.

These definitions emphasize that ambiguity arises from multiple plausible meanings embedded within a single requirement, posing significant challenges to ensuring shared understanding and preventing defects during software development. The common thread is that ambiguity is not simply about multiple meanings, but also about the contextual factors and stakeholder backgrounds that contribute to diverse interpretations. This definition remains highly relevant, as recent research by Bashir et al. (2025) also emphasizes that ambiguity arises from vague or imprecise language in requirements, which inherently allows multiple interpretations. Authors highlight that ambiguity is fundamentally about multiplicity of meaning, posing significant challenges in requirements engineering where clarity and shared understanding are crucial. In SRS, the requirement is said to be ambiguous when it can be interpreted in more than one way.

Scholars have long emphasized that ambiguity often stems from vague phrasing or insufficient detail during the requirements gathering phase. Several studies highlight that ambiguity often originates during the requirements elicitation phase, particularly when stakeholders provide vague, incomplete, or inconsistent input. Although natural language is widely used in software requirements because it is easily understood by stakeholders, it remains highly susceptible to ambiguity unless strict controls and structured guidelines are applied. Recent industrial studies emphasize that ambiguous natural language requirements lead to misunderstandings and costly errors throughout the development lifecycle, highlighting the importance of early detection and automated resolution techniques using advanced tools such as Large Language Models (Bashir et al., 2025; Jia et al., 2025). For

instance, terms such as “user-friendly,” “efficient,” or “secure” are commonly used in SRS but are rarely quantified, leading to differing interpretations across development teams.

To address this, some organization adopt formal specification techniques that translate requirements into mathematical notations, significantly reducing the risk of ambiguity. For example, the vague requirement “the system shall process data efficiently” can be rewritten formally as “the system shall complete processing of 500 records in under 2 seconds.” Huisman et al. (2020) highlight that despite advances in tooling, the industrial adoption of formal methods continues to be constrained by their inherent complexity and the steep learning curves, as well as poor integration with existing development environments. These findings echo the earlier observation of Cofer et al. (2013), who noted the adoption of formal methods in industry is limited primarily due to the complexity of applying formal techniques and the need for specialized skills and education. The study emphasizes that many engineers lack adequate training to correctly use formal methods and interpret their output, while usability and tool integration issues further hinder widespread acceptance. Similarly, Davis et al. (1993), states that such precision improves clarity, the adoption of formal methods is limited due to their complexity and the specialized skills required.

Recent research by Bashir et al. (2025) and Kwizera (2025) emphasizes the critical importance of early and systematic detection and clarification of ambiguous requirements to prevent misunderstandings and costly errors during software development. Bashir et al. (2025) highlights the effectiveness of large language models in industrial settings for identifying and explaining ambiguities, while Kwizera (2025) advocates for automated AI-assisted techniques to manage ambiguity early in the requirements engineering process. These findings build upon earlier work by Zowghi and Coulin (2005), who also stressed that early detection and clarification of ambiguous requirements are essential to avoid misinterpretations and ensure successful project outcomes.

Involving multiple stakeholders through structured elicitation techniques such as interviews, use case analysis, and joint application design sessions can help expose potential ambiguities. Activities like requirements reviews, walkthroughs, and prototyping are also effective in clarifying intentions. For instance, a review might uncover that the phrase “users can easily reset their password” lacks detail, prompting further specification such as “users can reset their password via email link within 3 steps.” Furthermore, automated ambiguity detection tools have come up to support requirements engineers in identifying vague expressions. These tools analyze textual requirements for linguistic patterns that commonly lead to ambiguity, such as modal verbs (“shall,” “should”), vague adjectives (“fast,” “easy”), and unspecified quantities. According to Bashir et al. (2025), tools leveraging large language models significantly enhance the automated detection and explanation of ambiguities in natural language requirements, which is particularly beneficial in large-scale projects where manual reviews become impractical. As Hughes et al. (2007) also highlight, such tools are especially useful in these complex contexts because they help manage the volume and complexity of requirements, improving early ambiguity identification and reducing costly errors.

By highlighting ambiguous terms, these tools assist teams in refining documentation before it enters the development phase, thus helping to prevent misunderstandings and rework downstream. This is supported by empirical findings from Bashir et al. (2025), whose industrial study demonstrated that large language models achieve a notable improvement around 20% performance increase in classifying ambiguous requirements compared to baseline approaches and provide useful natural language explanations that facilitate stakeholder understanding and clarification. These advancements mark a significant step forward in handling ambiguity in real-world, large-scale software development projects.

The impact of ambiguity in software requirements can lead to big issues later on even when misunderstandings appear minor at first. Ambiguous requirements are a major

cause of defects in software systems and often resulting in scope creep, project delays, and customer dissatisfaction. When requirements are misunderstood, the resulting functionality may misalign with business goals or fail to meet user needs. Therefore, managing and clarifying ambiguity early in the requirements process is essential to ensure software quality and stakeholder satisfaction. In summary, ambiguity in software requirements remains a significant obstacle to successful software development. To fix this problem, a multifaceted approach involving precise language use, active stakeholder engagement, systematic validation techniques, and technological support through automated tools would assist with the process. As the field evolves, developing user-friendly ambiguity detection frameworks and promoting ambiguity awareness among stakeholders will be vital for enhancing the quality and reliability of software requirements.

2.3 TYPE OF REQUIREMENT AMBIGUITY AND TYPE OF SYNTACTIC AMBIGUITY

According to Kiyavitskaya, Zeni, Mich, and Berry (2008), ambiguity in software requirements can be categorized into four primary types: lexical, syntactic, semantic, and pragmatic ambiguity. These categories provide a foundational framework for understanding how ambiguity arises from language and context in requirements specifications. This classification remains highly relevant in contemporary research. For example, Mohamed, Din, and Baharom (2022) emphasize these same four types of ambiguity in their work on detecting pragmatic ambiguity with automated tools. Their study highlights pragmatic ambiguity as particularly challenging due to its dependence on context and stakeholder perspective, which often requires advanced analysis beyond surface-level linguistic patterns.

Further recent investigations, like those by Gervasi et al. (2022) and Kwizera (2025), continue to utilize this typology while exploring practical approaches for early

detection and resolution of ambiguities in natural language requirements. These studies underline that effective management of all four ambiguity types is crucial for producing clear, unambiguous specifications that reduce costly errors down the development lifecycle. A summary of the type of ambiguity and its details in SRS is shown in Table 3.

Table 3: Type of Requirement Ambiguity in SRS

Ambiguity Type	Definition	Example in requirement	Explanation	Focus
Syntactic	Syntactic or structural ambiguity is a type of ambiguity that arises when a sentence can be parsed in multiple ways, each resulting in a different meaning. This ambiguity type often leads to confusion and misinterpretation of software requirements, complicating their analysis and implementation (Sabriye & Wan Zainon, 2018; Frattini et al., 2017; Kwizera, 2025).	"I saw a woman with a bottle."	The statement can be parsed into two different meaning: 1) The woman has a bottle. 2) I used a bottle to see the woman. This ambiguity arises because the phrase "with a bottle" can grammatically attach either to the noun "woman", indicating	Sentence structure and Grammar

			possession or accompaniment, or to the verb “saw”, implying the bottle was used as a tool (such as binoculars) to see the woman. This type of ambiguity is purely syntactic, meaning it stems from the sentence structure rather than the meanings of the individual words. Since the sentence structure allows multiple valid parses, the parser cannot determine which attachment is intended, resulting in multiple plausible interpretations.	

Lexical	Lexical ambiguity refers to a situation where a single word can have several meanings, which can lead to multiple interpretations in understanding requirements (Sabriye & Wan Zainon, 2018; Gervasi, Ferrari, Zowghi, & Spoletini, 2019).	"The users of the system are teachers and students. They login to the system."	The word "They" in the second sentence is lexically ambiguous because its reference is unclear. It is not explicit whether "They" refers to just teachers, just students, or both groups. This ambiguity arises from uncertain meaning or reference of the word, causing confusion in interpreting who should log in to the system.	Meaning of word
Semantic	Semantic ambiguity occurs when a sentence or requirement statement has multiple interpretations within its context, without involving lexical,	"All residents should have a house security pin code."	The sentence can be interpreted into two ways: 1) Every resident has an individual house security pin code.	Meaning of word

	structural, or syntactic ambiguity (Sabriye & Wan Zainon, 2018). More broadly, it arises when multiple meanings exist within the semantic space for the same text, independent of other types of ambiguity (Sabriye & Wan Zainon, 2018; Gervasi, Ferrari, Zowghi, & Spoletini, 2019). This type of ambiguity often results from unclear or multiple meanings in requirement statements, which can negatively affect the elicitation process and the overall quality of the specifications (Dar et al., 2022).		2) All residents have the same house security pin code. This ambiguity arises from the unclear semantics of the phrase "a house security pin code," which does not specify whether the pin code is personal or shared among all residents.	
Pragmatic	Pragmatic ambiguity focuses on the relationship between a sentence's meaning and its context, depending heavily on the requirement's context	"Would you like to have a glass of water?"	The sentence can be interpreted in two ways: (1) a question about preference for cold or warm water, or (2) an	Meaning of word

	and the knowledge of its reader. For example, two readers with different backgrounds may interpret the same requirement differently (Sabriye & Wan Zainon, 2018). Similarly, Shareef et al. (2022) highlight that pragmatic ambiguity arises from differences in interpretation caused by the context and the reader's knowledge. Kwizera (2025) further discusses pragmatic ambiguity as being influenced by the reader's background and other contextual factors in requirements documents.		offer to prepare a glass of water if desired. The ambiguity arises due to the reader's knowledge and situational context, not from the words themselves or their grammatical structure. The meaning depends on pragmatic considerations, such as tone, setting, or prior knowledge.	

Many researchers focus primarily on syntactic ambiguity in requirements because analyzing sentence structure and grammar provides clearer and more immediate indicators of ambiguity compared to semantic or pragmatic ambiguities, which require deeper contextual understanding and domain knowledge (Sabriye & Wan Zainon, 2018; Mohamed, Din & Baharom, 2022). They often prioritize detecting syntactic ambiguity first

because it provides a solid foundation by resolving issues with sentence structure and parsing; they reduce a significant source of misunderstanding early in the requirements validation process.

Scholars have stated that it is critical to first resolve syntactic ambiguities before considering the semantics. For example, Spivey-Knowlton, Trueswell, and Tanenhaus (1993) argue that during sentence comprehension, the initial syntactic structure building happens before semantic and discourse context effects take place, emphasizing that syntactic alternatives are considered first and semantic interpretation follows structural parsing is completed. This study discusses how the initial syntactic structure building is a prerequisite step during sentence comprehension before semantic and discourse context can affect ambiguity resolution. Their work emphasizes that syntactic alternatives must be considered first, and only after this structural parsing does semantic interpretation proceed. Likewise, Van Gompel et al. (2005) note that the processor constructs initial syntactic analyses without initial input from semantic plausibility, which comes only after the syntactic structure has been established. These support that semantics depends on syntactic disambiguation for correct meaning.

Syntactic ambiguity is a critical concern in SRS because it directly affects the clarity and precision of documented requirements. The syntactic structure forms the foundation of clear communication even when individual terms are well-defined, flawed grammar or sentence construction can render the meaning ambiguous (Funk et al., 2017). This type of ambiguity often leads to misinterpretation, which can result in incorrect implementation, communication breakdowns, or even contractual disputes (Kamsties, 2001). Compared to other forms such as semantic or pragmatic ambiguity, syntactic ambiguity is relatively easier to detect systematically through rule-based methods, pattern matching, or the use of syntactic parsers (Friedrich et al., 2013). This makes it a practical and effective starting point for ambiguity detection efforts. Moreover, addressing syntactic ambiguity early helps reduce the propagation of errors, thereby minimizing the chances of lexical, semantic, or

pragmatic misunderstandings further down the line (Goldin & Berry, 1994). This approach has been supported by studies from Sabriye and Wan Zainon (2018) and Mohamed, Din, and Baharom (2022).

By focusing on the structure and grammar of the sentence, readers can more accurately determine the intended interpretation, which then allows for more precise comprehension of the overall meaning. Recent research confirms that participants resolve syntactic ambiguities more quickly and accurately when they are directed to focus on sentence structure rather than infer meaning from context (Wu, Liu, & Lu, 2020). This aligns with earlier psycholinguistic findings and highlights the benefits of promoting syntactic awareness to improve comprehension and disambiguation. Authors conclude that training individuals to prioritize syntactic analysis over semantic processing can significantly improve their ability to navigate and understand complex sentences.

Syntactic ambiguity can be divided into four types which are analytical, attachment, elliptical, and coordination. A summary of the type of syntactic ambiguity is listed in the table 4:

Table 4: Type of Syntactic Ambiguity

Type	Definition	Example in requirement	Explanation
Analytical	Analytical ambiguity occurs when the role of the constituents within a phrase or sentence is ambiguous (Kiyavitskaya, Zeni, Mich and Berry, 2008).	A complex noun group with an identifier. "The Italian office director. . ."	"Italian" can be referred to the office or to the director.
Attachment	Attachment ambiguity occurs when a particular syntactic constituent of a sentence, such as a prepositional phrase or a relative clause, can be legally attached to two parts of a sentence (Kiyavitskaya, Zeni, Mich and Berry, 2008).	"The officer edits a resume with a template for a final assessment."	Here "for" can be referred to the "template", or to the "resume "or can specify a deadline (i.e., before the final assessment)
Elliptical	Elliptical ambiguity occurs when there is uncertainty about whether a sentence contains an ellipsis (an omission of words) or when the omitted elements themselves are unclear. This	"The employee met the council, and the head of office and secretary assessed his presence."	Here the sentence can have several parses. It is unclear whether both the head of office and the secretary assessed the

	ambiguity arises because the listener or reader must infer the missing parts from context, and different interpretations are possible depending on how the ellipsis is resolved. (Kiyavitskaya, Zeni, Mich and Berry, 2008).		presence of the employee or just the secretary.
Coordination	Coordination ambiguity occurs: (1) when more than one conjunction, i.e. 'and' and 'or', is used in a sentence creating uncertainty about how the elements are grouped; and (2) when one conjunction is used with a modifier, making the scope and association of the modifier unclear (Kiyavitskaya, Zeni, Mich and Berry, 2008).	"The successful candidate receives the letter on Sept. 12, and the unsuccessful doesn't."	Here, the ambiguity is whether the unsuccessful candidate receives a notification on another date or does not receive any notification.

2.4 EXISTING TECHNIQUES IN DETECTING AMBIGUITIES IN SRS

SRS are typically written in natural language to ensure they are understandable by both technical and non-technical stakeholders. Nonetheless, natural language introduces various ambiguities such as lexical, syntactic, semantic, and pragmatic that can result in

misunderstandings, implementation errors, and costly rework. Over the years, several techniques have been developed to detect and mitigate these ambiguities in software requirements. While a lot of research has been conducted on techniques for reducing or solving ambiguities in software requirements, the practical implementation of these methods remains a significant challenge for many organizations.

Manual inspection techniques involve a thorough review of SRS by human experts using structured aids like checklists, reading strategies, and style guidelines. This approach relies heavily on the expertise and judgment of reviewers who manually identify ambiguous statements, such as vague or overly general terms. For example, Kamsties et al. (2001) demonstrated the use of checklist-based reading, where requirements like "The system shall support all standard protocols" were flagged due to the unclear term "support." While this method allows for deep contextual understanding and analyzes details carefully, it is very time-consuming, subjective, can be biased, error-prone and does not work well for larger projects (Riaz, Butt, & Rehman, 2019).

Semi-automatic approaches using Natural Language Processing (NLP) apply linguistic techniques such as part-of-speech (POS) tagging and rule-based heuristics to automatically detect ambiguity by analyzing the syntactic and lexical structure of sentences as exemplified by frameworks that compare favorably to manual inspections (Sabriye & Zainon, 2017). For instance, tools such as QuARS utilize POS tagging and pattern recognition to flag problematic constructs, such as modal verbs, vague adjectives, and passive voice. For instance, a requirement like "The user may access the system remotely" would be marked ambiguous due to the uncertainty introduced by the word "may." These methods offer automation and can find many issues, but they often produce many false alarms and depend on how advanced the NLP technology is.

Alternatively, machine learning (ML) models, particularly supervised classifiers, have recently demonstrated strong performance in ambiguity detection. For instance, tools

like TAPHSIR utilize ML to detect and resolve anaphoric ambiguity in requirements to achieve high precision and recall rates that outperform both fine-tuned language models and rule-based methods (Ezzini et al., 2022). ML based semi-automatic techniques employ trained models to classify ambiguity in SRS texts. Using supervised learning algorithms such as Naïve Bayes, these approaches analyze features like sentence length, POS tags, and the frequency of vague terms to detect ambiguity. Osman and Zaharin (2018), for example, used such a model to identify ambiguous phrases like "The system must be efficient," highlighting how the model can generalize from training data to new inputs. While this method can grow and get better over time, it still requires regular training to keep working well.

Lastly, knowledge dictionary-based methods use lists of words that are often unclear or only used in a certain area. Tools scan SRS and flag any phrases that match entries in the dictionary, such as "user-friendly" or "optimal performance." When encountering a sentence such as "The database shall maintain optimal performance," the tool prompts a clear explanation of what "optimal" means. These techniques are simple and easy to implement, but they need to be adjusted for each specific field and may miss unclear meanings that depend on the context and are not in the dictionary (Kato, Tsuda, & Masuda, 2022).

2.4.1 Techniques Specifically Targeting Syntactic Ambiguity

While many techniques address general ambiguity in SRS, only some of them target syntactic ambiguity. In "Techniques for Mitigating Syntactic Ambiguities in Software Requirements Specification", a paper by Sharma et al., (2020), the authors outline several existing approaches to address this challenge:

1. Controlled Natural Language (CNL): Kuhn (2014) provides a comprehensive survey of CNL, including their applications in reducing ambiguities in technical

and software requirements. This work extensively discusses restricting grammar and vocabulary. The core idea was first introduced by Fuchs and Schwitter (1996), who demonstrated that using a restricted subset of natural language with well-defined grammar and vocabulary rules can significantly minimize syntactic ambiguities in requirements. They explored different CNL types, their precision, expressiveness, and practical industrial use, supporting the argument that CNL can improve clarity and reduce misunderstandings in SRS.

2. Template-based Requirements: Sharma et al., (2020) mention a study by Berry and Kamsties (2004), which show that using standard templates with predefined syntactic structures helps make requirements clear and consistent.
3. Formal Specification Languages: The paper highlights the early foundational work by Hall (1990), who concluded that translating requirements into formal notations like Z or VDM can eliminate syntactic ambiguities due to the precise, mathematical semantics of these languages. Recent research continues to support this perspective, highlighting advancements in the use of formal methods for requirements formalization. For example, Beg, O'Donoghue, and Monahan (2025) survey current techniques whereby natural language requirements are transformed into formal specifications using languages like Z, VDM, or Dafny, significantly reducing and improving ambiguity.
4. Dependency Parsing: The authors cite the work of Yadav et al. (2018), which showed how applying natural language processing techniques, such as dependency parsing, can automatically analyze the syntactic structure of requirements and identify potential ambiguities. Additionally, Li, Zhang, and Chen (2014) introduced an ambiguity-aware ensemble training framework for semi-supervised dependency parsing, which effectively captures ambiguous syntactic structures by leveraging parse forests generated from multiple parsers. This approach supports the identification of various ambiguity types, including lexical, syntactic, and semantic ambiguities in natural language requirements.

5. Requirement Reviews and Inspections: The paper highlights the research of Fabbrini et al. (2001), who found that involving multiple stakeholders in thorough reviews of requirements can help identify and resolve syntactic ambiguities through discussion and consensus building. More recent research confirms the ongoing importance of such collaborative efforts. Zafar et al., (2021) examine challenges in agile software development, emphasizing that ambiguity in requirements continues to impact quality significantly and that effective communication and review among agile teams are critical to mitigating these issues.

Controlled Natural Language (CNL), template-based methods, and formal specification languages are most effective at resolving syntactic issues because they use strict and clear grammatical structure rules. Dependency parsing, as an NLP-based technique, also helps identify ambiguous sentence constructions such as unclear clause boundaries or multiple possible parses. Additionally, rule-based ambiguity detection tools and lexicon checkers assist in recognizing syntactic patterns associated with ambiguity, such as unclear modifiers or ambiguous references. These methods are crucial for identifying grammatical constructs that can lead to misinterpretation during requirements analysis or system design.

Sharma et al., (2020) conclude that a combination of these techniques is essential to systematically address syntactic ambiguities and ensure that software requirements are expressed in a clear, unambiguous manner.

2.4.2 Limitations of Existing Techniques

Despite the growing number of studies and tools developed to detect ambiguities in software requirements, there are still important problems that make them less useful and harder to use in real projects. Automated RE tools frequently face significant limitations

due to domain dependency, requiring extensive customization for specific application areas such as healthcare, finance, or manufacturing. These tools rely on specialized terminologies, ontologies, and domain knowledge that work effectively only within their intended domains. When applied to different fields without substantial adaptation, their accuracy and usefulness often degrade, making cross-domain reuse difficult and costly. As a result, companies working across multiple industries face increased expenses, longer deployment times, and reduced likelihood of widespread tool adoption; Domain knowledge is essential for improving requirements quality and reducing ambiguities, but scalable, domain-independent tools remain largely undeveloped. Recent systematic studies and empirical research consistently highlight this challenge, emphasizing that despite technological advances, domain dependency continues to be a key barrier to broader adoption and effectiveness of automated requirements engineering tools (Arora et al., 2018; Araújo et al., 2025; Kücherer, 2018; Fischer & Bauer, 2010; Umar & Lano, 2024).

Automated tools based on Natural Language Processing (NLP) or dictionary-based techniques often generate false alarms by flagging terms as ambiguous solely because they appear on predefined ambiguity lists, regardless of whether the meaning is clear from context. Ferrari, Gnesi, and Lami (2014) demonstrated that this over-sensitivity causes unnecessary warnings that frustrate users and reduce trust in the tool's recommendations. Recent research corroborates these findings. According to Bajceta et al. (2022), a systematic evaluation of ambiguity detection tools revealed that pattern-based methods tend to achieve high recall, successfully identifying many ambiguous instances. However, this comes at the cost of generating numerous false positives, which can overwhelm users with irrelevant alerts. Pragmatic ambiguity, where meaning depends heavily on context, continues to challenge traditional NLP models. This limitation often results in a high number of false positives, especially when contextual cues are missing or misinterpreted. Prior studies (Hembree, 2021; Osama, 2020) have highlighted this issue, and recent models such as the Pragmatic Ambiguity Detection Model (Shi et al., 2025) attempt to address it through uncertainty calibration. However, even state-of-the-art language models still

struggle to capture subtle ambiguity, as shown by Liu et al. (2023), which can lead to incorrect flags in requirement analysis. To improve contextual sensitivity, Mehrparvar and Pezzelle (2024) propose multilingual translation-based methods, though they acknowledge that semantic depth is still needed to eliminate false alarms. These findings reinforce the need for hybrid approaches that combine automated detection with human insight, especially in domains where precision is critical. This research builds on these foundations by proposing a context-aware framework that reduces false positives through semantic refinement and domain-specific adaptation.

On the other hand, checking requirements manually or manual inspection can work well but takes a lot of time and poses a challenge. Mavin et al., (2017) explain that while manual techniques such as checklist-based reviews are often effective at identifying ambiguities, but they do not work as well in big or fast changing projects especially in agile environments. This is because manual reviews take a lot of time and effort. When a project grows or requirements keep changing, it is hard to keep up. Agile development moves quickly and needs constant updates, while manual checks cannot match the speed. Thus, while these methods are useful for small projects, they are not effective in fast moving projects. This shows the need for tools that are partly or fully automated to handle more work quickly and accurately.

Furthermore, formal methods such as mathematical specifications offer high precision resulting in detecting ambiguities. Unlike natural language approaches, formal methods use rigorously defined syntax and semantics to represent system properties unambiguously, enabling systematic verification and validation (Clarke, Grumberg, & Peled, 1999). However, as Zowghi & Gervasi, (2003) point out, these methods are often avoided due to their complexity and steep learning curves. This complexity makes it difficult for many stakeholders especially those without a strong technical background to fully understand or engage with the specifications. As a result, formal methods tend to be underutilized in practice, limiting their impact despite their accuracy.

Lastly, ML-based techniques require large amounts of labeled data to train their models effectively, as Zhao and Kamalrudin (2018) highlighted. Producing these high-quality datasets involves significant effort, time and cost, which many organizations find prohibitive. Without sufficient training data, ML models may perform poorly or fail to generalize well, reducing their reliability and usefulness in real-world projects. This data dependency creates a practical barrier to adopting ML based ambiguity detection widely. These limitations highlight the need for a more accessible and balanced framework such as the one proposed in this study, which focuses on syntactic ambiguity detection while considering practical adoption in software engineering processes.

2.5 EXISTING FRAMEWORK TO DETECT AMBIGUITIES

2.5.1 Software Requirement Ambiguity Avoidance Framework (SRAAF)

Gupta and Deraman (2019) proposed a framework called the Software Requirement Ambiguity Avoidance Framework (SRAAF) to facilitate requirement engineers write unambiguous software requirements. The framework aims to reduce ambiguities early in the RE process by selecting the most effective elicitation techniques tailored to the project's situational attributes, including characteristics of the project, stakeholders, and the requirement engineer. Specifically, SRAAF begins by evaluating various situational factors related to the project, the stakeholders, and the requirement engineers. By using this evaluation, the framework recommends the most appropriate elicitation techniques tailored to these conditions, aiming to gather clearer, more precise requirements from the start and thus avoid introducing ambiguity during elicitation. The framework then applies the W6H technique (What, Who, Where, Which, When, How, Why) to verify each elicited requirement systematically, ensuring completeness and clarity before formalizing requirements in the SRS. This verification step helps identify and eliminate vague or

incomplete information proactively. By integrating structured verification activities, SRAAF seeks to prevent ambiguity before it enters the SRS, thereby reducing the costly and time-consuming process of ambiguity detection and resolution after documentation (Gupta & Deraman, 2019; Dar et al., 2022).

The rationale behind this framework is that ambiguity in software requirements is a pervasive problem that often manifests later in the development lifecycle, leading to significant rework and increased costs. While there are many existing techniques and tools to detect and resolve ambiguity, the authors recognize the need for a more proactive approach to avoid ambiguity from the outset.

The SRAAF framework aims to address this gap by guiding requirement engineers through the selection of the most appropriate elicitation technique, followed by a structured verification process using the W6H technique. This two-pronged approach is intended to help ensure that software requirements are unambiguous and complete, rather than having to identify and resolve ambiguity after the fact.

Nevertheless, a few potential gaps or areas for improvement in the SRAAF can be identified. It is noted that SRAAF, while promising, is not yet fully implemented or integrated with advanced NLP tools, which limits its current practical use. Furthermore, the paper presents the conceptual framework and the rationale behind its components but does not provide any empirical validation or case studies demonstrating the framework's effectiveness in real-world settings. Rigorous testing and evaluation of the framework's ability to reduce requirement ambiguity and improve software development outcomes would strengthen the claims made in the work.

Next, the work mentions evaluating various attributes to select the appropriate elicitation technique but does not provide a comprehensive or structured process for this selection. These attributes include factors such as project size and complexity, the expertise

and number of stakeholders involved, communication constraints, and the skill set or experience of the requirement engineer. However, the work does not specify how these attributes are weighted or scored relative to each other, nor does it describe a formal decision-making algorithm that would make the technique selection process more transparent and reproducible. This leaves the approach somewhat heuristic and dependent on the practitioner's judgment. The work mentions that new evidence related to attributes and matrices can be added iteratively, implying extensibility, but does not clarify if or how this is implemented systematically.

Besides, the work does not discuss how the framework can be adapted or extended to accommodate new elicitation techniques or changing project/stakeholder requirements over time. Enhancing the framework's flexibility and ability to incorporate new evidence or techniques would improve its long-term viability and applicability. Furthermore, the work does not explore how the proposed SRAAF framework can be integrated with or complement existing requirements engineering practices, tools, and methodologies. Providing guidance on how the framework can be seamlessly incorporated into common software development lifecycles would increase its practical relevance and adoption.

Finally, the work does not define clear metrics or measures to evaluate the effectiveness of the framework in reducing requirement ambiguity and improving software development outcomes. Establishing quantifiable criteria and benchmarks would enable more rigorous assessment of the framework's impact and guide future refinements. Addressing these gaps through further research, empirical studies, and practical implementation would strengthen the SRAAF framework and enhance its value for software requirement engineers and project stakeholders.

2.5.2 DARA Framework (Detection and Resolution of Ambiguities in Requirements Analysis)

The issue of ambiguity in SRS has long been a significant challenge in the field of requirements engineering. Traditional approaches to addressing this problem have often been limited in their capabilities, relying on either rule-based heuristics or basic machine learning techniques. However, recent advancements in the field have resulted in more comprehensive and effective frameworks for detecting and resolving ambiguity in requirement documents.

One such notable framework is presented in the work by Osama and Aref (2018), in which they proposed an automated approach called Detecting and Resolving Ambiguity (DARA) for identifying and resolving ambiguities in requirement documents. The framework aims to automatically identify and resolve various types of ambiguities in SR to enhance their clarity and reduce fault-prone error. Specifically, DARA seeks to detect ambiguous expressions in multiple domains including lexical, referential, coordination, scope, and vague ambiguities and then resolve the to produce more precise requirements, thereby improving software quality and reducing misunderstandings that can lead to defects during development. The framework accomplishes this through three main modules: a text preprocessing module that prepares the requirements text for analysis, an ambiguity detection module that leverages linguistic and rule-based techniques to locate ambiguous terms and constructs, and an ambiguity resolution module designed to clarify or eliminate the detected ambiguities.

The text preprocessing module performs essential NLP tasks, such as sentence splitting, tokenization, part-of-speech tagging, and syntactic parsing, to prepare the input text for further analysis. The ambiguity detection module then uses dictionaries of ambiguity indicators to identify different types of ambiguities, including lexical,

referential, coordination, scope, and vague ambiguities. Finally, the ambiguity resolution module automatically resolves the detected ambiguities based on a set of predefined rules.

The authors implemented the DARA framework as a tool and evaluated its performance on a collection of real-world requirement specification documents, demonstrating the approach's effectiveness in addressing the critical issue of ambiguity in software requirements. This work provides a structured and automated solution to a longstanding problem in requirements engineering, which can have significant practical implications for improving the quality and clarity of software requirements.

While the DARA framework proposed by Osama and Aref (2018), represents a valuable contribution to the RE field, there are several areas that require further investigation and research. Firstly, extensive evaluations using comprehensive real-world datasets reflecting practical complexities are necessary to establish their effectiveness across varied domains.

Moreover, the research did not compare the DARA technique to other existing methods for ambiguity detection and resolution in requirements engineering. Conducting a comparative analysis would help position this work within the broader context of the field and highlight its unique strengths and weaknesses. Furthermore, while DARA contains an ambiguity resolution module, research is needed to develop more automated, explainable, and user-friendly resolution strategies that adapt to stakeholder inputs and domain-specific rules. Quantitative metrics such as the Ambiguity Density Index proposed by DARA require further validation and benchmarking against standard ambiguity taxonomies to promote broader adoption and comparability across tools. Another vital area for enhancement is the systematic incorporation of domain knowledge, allowing tailored ambiguity detection customized to specific industries or application areas.

Finally, the study focused primarily on the technical implementation and evaluation of the DARA, without extensively exploring the user experience and practical integration of the framework into existing requirements management workflows. Gathering feedback from requirements engineers and other stakeholders on the usability and usefulness of DARA could provide valuable insights for improving the tool's adoption and real-world impact.

2.5.3 A Framework for Detecting Ambiguity in SRS (Sabriye and Wan Zainon's POS Tagging Approach)

Another framework was introduced in a previous study by Sabriye and Wan Zainon (2017). The aim of the framework is to automatically detect ambiguities in SRS, particularly focusing on lexical, syntactic and semantic ambiguities that commonly arise because these documents are written in NL. By applying POS tagging techniques combined with a set of linguistic and rule-based considerations, the framework highlights potentially ambiguous parts within the SRS to facilitate early identification and correction. This helps software analysts and developers recognize unclear or ambiguous requirements that could otherwise lead to misunderstandings or incorrect implementations. The framework thus aims to improve the quality and clarity of requirements by catching ambiguity systematically and automatically, reducing reliance on manual and error-prone detection processes.

Based on the information presented in the paper, the framework employs part-of-speech (POS) tagging as the core technique to analyze the grammatical structure of the requirements statements in the SRS combined with linguistic and rule-based analysis. By examining the POS tags assigned to each word, the framework aims to identify ambiguous phrases, words, or sentence constructs that could lead to multiple interpretations of the requirements. The approach relies on several key heuristic's rules applied to the POS-tagged text. These heuristic focus on identifying structures and word usages commonly

linked with ambiguity in NL requirements. The framework is designed to automatically process the SRS document and highlight the potentially ambiguous parts that need to be clarified, rather than relying solely on manual human review. This automated analysis approach is intended to improve the efficiency and consistency of the ambiguity detection process compared to manual methods. The process can be summarized as follows:

1. Preprocessing and Sentence Splitting: The natural language SRS document is split into individual sentences to handle them discretely.
2. POS Tagging: Each sentence is tagged automatically with its grammatical parts of speech such as nouns, verbs, adjectives, adverbs using a POS tagger. This tagging assigns syntactic labels that make it possible to analyze sentence structure.
3. Detection of Ambiguities via POS Patterns: The framework examines POS tag sequences to detect indicators of ambiguity. For example, it flags:
    a) Lexical ambiguity where a word can serve multiple parts of speech in different contexts, suggesting multiple meanings.
    b) Syntactic ambiguity by identifying passive voice usage and other grammatical constructions that can give rise to multiple interpretations.
    c) Coordination and scope ambiguities through analysis of conjunctions and sentence boundary structures.
4. Rule-Based Identification: Using a set of predefined linguistic rules and patterns, the framework highlights sentences or phrases likely to be ambiguous due to their POS tag patterns.
5. Output Highlighting: Ambiguous parts are flagged or highlighted for further human review and correction.

The framework was validated by comparing its ambiguity detection results with human evaluations, showing improved ability to detect especially lexical ambiguities and complex sentences combining multiple ambiguity types. This automated detection approach helps reduce reliance on manual inspection, which is error-prone and time-consuming, ultimately facilitating clearer, less ambiguous requirements documentation.

This framework integrates linguistic and semantic techniques, beginning with the normalization of SRS input and the application of POS tagging to facilitate linguistic analysis. The framework detects ambiguous terms by comparing them against a curated database of computer science terminology known to have multiple interpretations. The system further enhances semantic analysis by leveraging word embedding techniques, which capture word meanings based on contextual usage, and linked data resources, which offer external knowledge on possible word senses.

A key feature of the framework is its interactive prototype tool, allowing users to upload SRS in .txt or .doc format. The tool highlights potentially ambiguous terms, provides alternative meanings upon user interaction, and enables users to select the appropriate interpretation, thereby facilitating disambiguation. The final output includes a detailed report on detected ambiguities and generates an ambiguity-corrected version of the document. Evaluation on open-source SRS datasets demonstrated the framework's effectiveness in identifying and mitigating ambiguity. By combining POS tagging, semantic vector representation, and linked data, this framework represents a practical model that enhances the clarity and quality of SRS documentation.

However, the work does not provide details on the particular POS tagging algorithms or tools utilized, nor does it describe how the framework determines what constitutes an ambiguous construct based on the POS tags. The validation and evaluation plan for the proposed framework is also not fully elaborated in this conference publication. Without access to a more detailed technical description or an implementation of the framework, it is difficult to fully assess the novelty and effectiveness of the proposed approach.

2.5.4 Unifying Framework

The paper by Gervasi and Zowghi (2005), focuses on the pervasive issue of ambiguity in RE. It begins by emphasizing the prevalence of ambiguity in requirements and its significant impact on the success of software projects, often leading to misunderstandings, errors, and rework that increase costs and delays. The paper then provides a comprehensive definition of ambiguity, distinguishing it from related concepts like vagueness and uncertainty.

The core contribution of the work is divided into three parts: it proposes a unifying framework for understanding and managing ambiguity in requirements engineering, which consists of the sources of ambiguity (i.e., language issues, stakeholder differences, incomplete information), the types of ambiguity (i.e., lexical, syntactic, semantic, pragmatic), and the consequences of ambiguity (i.e., misunderstandings, errors, rework).

Additionally, the paper discusses various strategies and techniques for identifying, analyzing, and resolving ambiguity in requirements, such as structured questioning, prototyping, and formal methods. Finally, the authors highlight the implications of their framework for RE practice and education, as well as areas for future research in this critical domain. The research does not explicitly highlight any major gaps or limitations in its approach. However, based on the information provided in the paper, there are a few potential gaps or areas for further exploration. Firstly, the paper presents a conceptual framework for understanding and managing ambiguity in requirements engineering, but it does not provide extensive empirical validation of the framework. Conducting case studies, surveys, or controlled experiments to assess the applicability and effectiveness of the proposed framework in real-world requirements engineering projects would strengthen the paper's contribution.

Secondly, the work mentions the need for developing automated tools for ambiguity detection and resolution, but it does not provide details on the development or evaluation of such tools. Exploring the design and implementation of software tools to support the identification and resolution of different types of ambiguity could be a valuable area for future research.

Furthermore, the paper discusses various strategies and techniques for managing ambiguity, however, it does not provide guidance on how to prioritize or weigh different approaches based on the specific context and project constraints. Investigating frameworks or decision-making models to help requirements engineers navigate the trade-offs and optimize the use of ambiguity management techniques could be a helpful extension.

Besides, the paper focuses primarily on the technical aspects of ambiguity in RE, but it does not include deeply into the organizational and cultural factors that can influence the emergence and handling of ambiguity. Exploring the roles of stakeholder collaboration, communication practices, and organizational processes in the effective management of ambiguity could provide additional insights into how ambiguity in SRS arises, persists and can be mitigated throughout the development lifecycle. Stakeholder collaboration ensures diverse perspectives are incorporated early, reducing misunderstandings and incomplete requirements. Effective communication practices such as regular feedback loops, clearly defined terminology, and documentation standards help clarify ambiguous requirements and align expectations among team members and clients.

Moreover, well-structured organizational processes, including requirement reviews, validation protocols, and the use of traceability tools, provide systematic ways to detect, address, and monitor ambiguity over time. Investigating these social and procedural factors can complement technical approaches by highlighting how human interaction and organizational context influence ambiguity resolution, ultimately enhancing requirement clarity, reducing costly rework, and improving project outcomes.

2.6 REQUIREMENTS ENGINEERING FRAMEWORK OVERVIEW

RE plays a crucial role in software engineering by systematically identifying, analyzing, documenting, and managing the needs and constraints of stakeholders for a software system. As Sommerville (2011) emphasizes, it establishes the foundation for all subsequent development activities and significantly influences the overall success of software projects. The RE framework generally encompasses four main activities that contributes to building a clear and manageable set of requirements that guide development as described by Pohl (2010):

1. Requirement Elicitation & Analysis: Gathering and understanding what stakeholders need and expect from the system.
2. Requirement Specification (Documentation): Clearly recording these needs in a structured and agreed-upon form.
3. Requirement Verification & Validation: Checking that the requirements are correct, complete, and meet stakeholder expectations.
4. Requirement Management: Handling changes to requirements and ensuring ongoing alignment throughout the project.

Among the stages, several scholars highlight that requirement elicitation and requirement documentation as particularly critical because ambiguity frequently arises during these phases often due to communication challenges between stakeholders and analysts, limitations of traditional elicitation techniques, and insufficient attention to documentation processes, all of which contribute to unclear and incomplete requirements that can propagate errors and lead to project failures (Dar, Imtiaz, & Lali, 2022; Ferrari, Spoletini, & Gnesi, 2016; Serna & Serna, 2023).

They explain that unclear or vague requirements during these phases can potentially lead to misunderstandings, defects in the software, and costly rework later. This happens

because elicitation is the process of gathering information from stakeholders about what the software should do, and if this information is unclear or incomplete, misunderstandings easily occur. As explained in various studies, elicitation involves techniques like interviews, workshops, and prototyping to uncover stakeholders' real needs and expectations, but if these techniques are not applied carefully, important details can be missed or misinterpreted.

Similarly, the documentation phase is where the gathered requirements are formally recorded. If requirements are written unclearly or inconsistently, different team members may interpret them differently, causing defects and costly rework. Clear, precise documentation acts as a contract between stakeholders and developers, ensuring everyone shares the same understanding of what the system should do. Therefore, ambiguity often originates in these early phases because they involve translating sometimes vague or conflicting stakeholder ideas into concrete, shared knowledge.

Therefore, managing ambiguity here through rigorous elicitation methods and careful documentation is essential to avoid costly misunderstandings and to set a solid foundation for the entire development process. By focusing on these early phases within the requirements engineering framework, organizations can significantly improve the quality of requirements, reduce misunderstandings, and enhance the likelihood of project success.

2.6.1 Requirement Elicitation

Requirement elicitation is the process of actively engaging with stakeholders including clients, end-users, domain experts, and business analysts to gather comprehensive information about their needs, expectations, and constraints (Kotonya & Sommerville, 1998). This phase is an important part of developing a system and can be very challenging due to the diversity of stakeholder perspectives, incomplete knowledge, and the often-tacit nature of requirements. In this phase, stakeholders describe what they want or expect the system to do. However, understanding and gathering these requirements is not easy. Many researchers continue to highlight that requirements are often misunderstood, despite their critical importance in software development. Recent studies underscore this persistent challenge, noting that stakeholders frequently struggle to articulate their needs clearly, leading to ambiguous, inconsistent, or incomplete requirements (Raj et al., 2025; Sami et al., 2024).

As requirement engineers, they need to find out where the requirements come from, gather them from the right people or documents, understand them clearly, write them properly, and make sure everyone involved understands them. Key sources include stakeholders such as end-users, business owners and subject matter experts who provide insights into needs and expectations. While existing documents such as process flow diagrams, standards, user stories, use case, prototypes, and user requirement specifications. Getting requirements from stakeholders is not just about asking questions. It is a knowledge-based process, meaning that the information acquired depends a lot on what the stakeholders know and how well they can explain it. The process becomes successful when all parties, both stakeholders and engineers can communicate, collaborate and understand each other clearly.

Communication means sharing and receiving information. Collaboration means working together to come up with ideas or make decisions. Understanding happens when

everyone has the same idea or meaning in their mind. Without communication, collaboration and understanding, many problems can occur during the requirement elicitation process (Cockburn & Highsmith, 2001; Davidson et al., 2003).

Some of the common problems include missing requirements, where stakeholders may forget to mention some needs or may not even realize what is possible with the new system. Sometimes, they do not want to be involved actively, which is also known as reluctant participants. Misunderstandings or misperceptions can also happen because stakeholders may use their own work terms or explain things based on their experiences, which the requirement engineers might not understand correctly and thus lead to ambiguity. According to Avison & Fitzgerald (2006); Sommerville (2004) disagreements between stakeholders about what the system should do can also happen. According to Nuseibeh & Easterbrook (2000), stakeholders may express their needs in vague, inconsistent, or conflicting terms, which can introduce ambiguity. Gathering clear and precise information from stakeholders is very complicated. Additionally, communication gaps between technical teams and non-technical stakeholders, as well as tacit knowledge that stakeholders may not explicitly state, contribute to ambiguity.

To solve these problems, different methods have been developed. Traditional methods include interviews, group discussions, observations, and reviewing documents. Newer methods include Joint Application Design (JAD), group tools, software tools, and system prototypes. Based on Hoffer et al., (2004), Agile methods also suggest working closely with users throughout the project to get regular feedback. Dar et al. (2022) highlight the emerging use of gamification in requirements engineering as a promising approach to increase stakeholder involvement and reduce ambiguity during requirements elicitation. Gamification incorporates game elements such as points, badges, leaderboards, challenges, and interactive simulations to make the elicitation process more engaging and motivating for participants. This active engagement encourages stakeholders to contribute more effectively, express requirements more clearly, and collaborate better throughout the

process. Studies show that gamified platforms can increase participation rates, improve the quality and creativity of requirements produced, and foster better communication among diverse stakeholders (Dar et al., 2022; Lombriser et al., 2016). For instance, tools like Gamify4Req and REfine integrate gamification APIs to incentivize clear and concise requirement expression while supporting collaboration and consensus building. Experimental results indicate stakeholders find gamified environments more motivating and are more engaged in producing requirements, which helps reduce ambiguity and improves overall requirements elicitation performance. However, challenges remain including managing large participant numbers and ensuring the ethical use of gamification (Dar et al., 2022; Snijders et al., 2018). Overall, gamification represents a valuable addition to traditional elicitation techniques by transforming the process into a more interactive, enjoyable, and productive experience. They also point out that continuous validation and refinement of requirements with stakeholders help ensure clarity and completeness, minimizing misunderstandings. Furthermore, Sinpang et al., (2017) explores techniques such as fuzzy inference systems to detect ambiguous terms early in the elicitation phase, helping analysts to identify and resolve unclear requirements.

Despite these advancements, significant gaps still remain in fully addressing ambiguity in RE. While gamification offers significant benefits, it is not a universal solution as gamification is not equally effective for everyone or every context. What motivates one group of stakeholders might not motivate another, and some individuals may not respond positively to gamified elements at all. The effectiveness of gamification varies depending on the specific elements used and the target audience.

While continuous validation and refinement of requirements with stakeholders is widely recognized as a best practice in requirements engineering, it also comes with several notable drawbacks such as increased time and effort. Continuous validation requires frequent stakeholder involvement, review cycles, and iterative feedback sessions. This can significantly extend the project timeline and increase costs, especially if stakeholders are

numerous or have limited availability. Furthermore, the advantages in handling uncertainty and imprecision, fuzzy inference systems also have drawbacks. First, making a fuzzy system is not easy as one needs to write good rules and decide what words like “fast” or “high” mean. This takes a lot of time and knowledge. If it is too hard, people may not want to use it. Second, fuzzy systems can be slow, especially if there are many rules or a lot of data. This can be a problem for big projects or systems that need fast answers. Third, sometimes the system gets confused. Two rules might say different things, like “do it” and “don’t do it.” This can lead to wrong answers. Fixing this can make the system even more complicated.

Even with all these techniques. problems still happen. This is because requirement elicitation is a human-centered activity. Li et al., (2005) point out that, unlike other stages in system development that can be automated, this process needs direct communication, usually face-to-face, to avoid misunderstandings.

2.6.1.1 Tacit Knowledge on Ambiguity Awareness

In RE, understanding stakeholder needs often involves more than just listening to what they say. Much of what stakeholders know about their work processes, goals, and system expectations is not explicitly stated. This kind of implicit understanding is known as tacit knowledge, a form of knowledge that individuals possess but find difficult to express, formalize, or communicate directly. Tacit knowledge is rooted in personal experience, intuition, expertise, and context, making it largely exist in the mind but not consciously known or felt and often unspoken. As stated by Dampney, Busch & Richards (2022); Gourlay (2006), It is typically acquired through practice, social interaction, observation, imitation, and hands-on experience rather than formal instruction.

Polanyi (1966) who is a pioneer in the study of tacit knowledge, famously stated, “We can know more than we can tell”. He described tacit knowledge as knowledge that is personal and context-specific, encompassing skills and insights that cannot be fully articulated but are essential for effective action. For example, the ability to ride a bicycle, play a musical instrument, or make complex decisions based on experience involves tacit knowledge. Sternberg et al. (2000) further characterize tacit knowledge as:

1. Acquired with little or no formal instruction or environmental support,
2. Procedural in nature, meaning it relates to knowing how rather than what,
3. Practically useful in everyday tasks and problem-solving.

Because it is highly personal and context-dependent, tacit knowledge is difficult to arrange according to a plan and transfer through traditional documentation or verbal communication. However, in the RE field, it plays a critical role in uncovering and resolving ambiguities during requirements elicitation. Ambiguity in requirements typically arises when stakeholder’s inputs are vague, open to multiple interpretations, or incomplete. However, ambiguity is not always a flaw as it can be a sign that valuable tacit knowledge remains hidden. Ferrari, Spoletini & Gnesi (2016) argue that ambiguity often emerges when stakeholders are unable to express their knowledge clearly or are unaware of the implicit assumptions shaping their expectations. In such cases, ambiguity acts as a signal, prompting requirements analysts to explore further through probing questions, clarifications, and dialogue. From this perspective, ambiguity becomes an opportunity rather than a threat. Analysts or engineers who recognize its value can use it to uncover hidden constraints, contextual information or previously unspoken insights. This process of surfacing tacit knowledge through ambiguity improves the completeness and precision of the final requirements, reducing the risk of miscommunication and errors during development.

From the viewpoint of a requirement engineer, tacit knowledge includes not only what stakeholders implicitly understand but also the unspoken expertise that engineers themselves bring to the project. This might include knowledge of similar systems, common

pitfalls in certain domains, or insight from previous experience with similar clients. Such knowledge is instrumental in helping engineers interpret ambiguous requirements, spot inconsistencies and identify missing information.

Benfell (2020) explains that tacit knowledge is embedded in the mental models and cognitive frameworks that engineers use to interpret stakeholder communication. These internal models guide how engineers make sense of unclear or conflicting requirements, often without them being fully aware of the process. By reflecting on these intuitive judgements and externalizing them, whether through discussions. diagrams or structured notes, engineers can improve the clarity and consistency of the requirements documentations. Moreover, engineers often act as translators between what stakeholders say and what they mean or need. Stakeholders may be unfamiliar with the technical constraints or unaware of available features, leading them to provide vague or incomplete input. Requirement engineers use their tacit understanding of domain knowledge, user behavior, and system design to bridge these gaps and formalize requirements that accurately reflect stakeholder needs.

The framework proposed by Mohamed, Basir, and Abdul Salam (2015) emphasizes the critical role of requirements engineers in converting tacit knowledge into explicit, actionable requirements. This process often involves using structured elicitation techniques informed by knowledge management models, such as Nonaka and Takeuchi's SECI model. The SECI model outlines four modes of knowledge conversion:

1. Socialization (tacit to tacit): Shared experiences and informal learning (e.g., shadowing users),
2. Externalization (tacit to explicit): Expressing intuitive knowledge through dialogue, metaphors, or models,
3. Combination (explicit to explicit): Integrating different sources of documented knowledge,

4. Internalization (explicit to tacit): Learning through practice and absorbing new insights into behavior.

By applying this model, engineers can intentionally guide the process of making tacit knowledge visible and usable in system development. Several techniques are effective for surfacing and utilizing tacit knowledge during requirements elicitation:

Table 5: Techniques to utilize tacit knowledge

Technique	Description
Ambiguity-led interviews	Engineers identify vague statements and ask probing questions to uncover deeper meaning (Ferrari et al., 2016).
Prototyping and role-playing	Stakeholders interact with mock interfaces or simulate scenarios to expose implicit workflows (Mohamed et al., 2015).
Observation and job-shadowing	Engineers watch stakeholders in their natural work environment to identify undocumented behaviors (Maiden & Rugg, 1996).
Rationale capture	The “why” behind each requirement is recorded to preserve context and decision-making logic (Mohamed et al., 2015).

Workshops (e.g., JAD, EasyWinWin)	Facilitated group discussions help uncover assumptions and enable collaborative clarification (Grünbacher & Briggs, 2001).

These techniques help ensure that both stakeholder and engineer tacit knowledge is acknowledged and incorporated into the final requirements specification. In summary, tacit knowledge is a critical but often hidden resource in RE. It supports how stakeholders think about their needs and how engineers interpret and structure those needs into formal specifications. Ambiguity, when managed properly, is a gateway to discovering this tacit knowledge. Through structured techniques such as interviews, observation, prototyping, and rationale capture, requirements engineers can make the implicit explicit, reducing ambiguity, improving requirement clarity, and contributing to the overall success of software development projects.

2.6.2 Requirement Documentation

Following elicitation, the collected requirements must be formally documented in a clear, precise, and structured manner, typically in a Software Requirements Specification (SRS) document (IEEE Std 830-1998). The SRS serves as a definitive reference that describes both functional requirements detailing the behaviors and services the system must provide and non-functional requirements specifying qualities such as performance, security, usability, and reliability (Sommerville, 2011). Proper documentation ensures a shared understanding among all project stakeholders, including developers, testers, and clients, thus mitigating the risk of misinterpretation (Pohl, 2010).

Requirement documentation is a crucial phase in the RE process where the elicited and analyzed requirements are formally recorded, typically in the form of a SRS. This

specification serves as a primary communication tool between stakeholders including clients, users, and project managers and the technical team responsible for developing the software. The main objective of requirement documentation is to ensure that all involved parties share a consistent and precise understanding of the system's expected functionalities, constraints, and goals. However, Sommerville (2011) states that ambiguity frequently emerges during this stage, often leading to misinterpretations, design flaws, or software that fails to meet user expectations.

One major reason ambiguity occurs in requirement documentation is the use of natural language, which, while flexible and widely understood, is inherently imprecise. Words and phrases such as "efficient," "user-friendly," or "support" can have multiple meanings depending on the context or the stakeholder's perspective. Furthermore, when the individuals responsible for writing the requirements lack sufficient domain knowledge, they may inadvertently misrepresent user needs, resulting in vague or incomplete statements. Stakeholders themselves often contribute to ambiguity by providing inputs that are informal, filled with unexplained acronyms, or based on implicit assumptions that are not clearly articulated. If these inputs are documented without sufficient clarification, the resulting requirements can be difficult to interpret accurately.

Ambiguity also arises when inconsistent terminology is used throughout the document such as referring to the same concept by different names or using the same term for different meanings thereby confusing the developers and testers. Additionally, overgeneralized or underspecified requirements such as "the system shall allow users to manage data" offer little guidance about the specific functionalities involved, leaving room for multiple interpretations.

To reduce ambiguity in requirement documentation, several strategies can be employed. One effective approach is the use of controlled or structured natural language, where predefined templates are applied to standardize requirement statements. For

example, using a structure such as “The system shall [perform function] when [condition] is met” helps ensure consistency and clarity. Another potential technique is to include a glossary of terms in the SRS. The glossary defines domain-specific terms, acronyms, and abbreviations, thereby minimizing confusion. Additionally, employing syntactic ambiguity detection tools or frameworks such as those based on POS tagging or linguistic pattern analysis can assist in automatically identifying vague or problematic sentences during the review process.

Peer reviews and requirement walkthroughs involving both technical and non-technical stakeholders are another practical method for uncovering ambiguous statements and aligning interpretations. Visual aids such as Unified Modeling Language (UML) diagrams, data flow diagrams, or interface mockups can further enhance understanding by translating textual requirements into visual representations. Furthermore, training requirement engineers and documentation authors on how to write clear, testable, and unambiguous requirements can significantly improve the SRS quality. Finally, documentation authors should avoid weak or subjective terms by replacing terms such as “better performance” or “as needed” with measurable and observable criteria, as recommended in the IEEE Standard for Software Requirements Specifications (IEEE, 1998).

2.6.2.1 Requirement Attributes

Requirement attributes represent key properties assigned to individual requirements that provide essential context for effective management throughout the software development lifecycle. According to Penmetsa and Lingampalli (2024), these attributes serve as critical metadata supporting traceability, prioritization, and decision-making throughout project phases. Among these attributes, Sawyer (2009) emphasizes that quality attributes are particularly important because they ensure requirements are clearly defined, and complete, consistent characteristics that help reduce risks and misunderstandings during development.

Quality attribute requirements, described as a specific subset of non-functional requirements, focus on how a system should perform rather than what it should do (Bass, Clements, and Kazman, 2012). They address system-level concerns such as performance, reliability, and usability, and because they are measurable and testable, they play a foundational role in guiding architectural and design decisions. Furthermore, Bass et al. (2012) highlight their importance in risk mitigation and prioritization, enabling project teams to focus on high-impact system qualities from the earliest stages.

Ambiguity presents a significant challenge in maintaining requirement quality; Zowghi and Gervasi (2003) explain that ambiguity occurs when a single requirement statement has multiple possible interpretations. This ambiguity can undermine the requirements engineering process by causing stakeholder misunderstandings, misaligned system designs, and costly corrections downstream. Therefore, as Zowghi and Gervasi (2003) argue, managing ambiguity is essential to ensure requirements convey a singular, clear meaning. Supporting this, IEEE Standard 830-1998 explicitly identifies unambiguity as a fundamental quality attribute of good requirements, underscoring that requirements must be expressed so that they have only one interpretation (IEEE, 1998).

Sawyer (2009) identifies clarity as a critical quality attribute that ensures requirements are expressed unambiguously and understood consistently by all stakeholders. By avoiding vague or subjective terms, clear requirements promote effective communication. Ambiguity often arises from vague language or requirements that can be understood in multiple ways; therefore, ensuring clarity through precise language and avoiding subjective terms is essential to eliminate ambiguity early in the requirements engineering process (Zowghi & Gervasi, 2003).

Similarly, Wiegers and Beatty (2013) stress the importance of completeness, where a requirement must fully describe all necessary functions, constraints, and conditions, since incomplete requirements may result in missing functionality or incorrect system behavior. Sommerville (2016) further highlights consistency as an essential attribute to prevent contradictory requirements that could cause confusion and errors during implementation.

Sommerville (2016) also discusses testability as the ability to verify whether a requirement has been met through testing or inspection; thus, requirements must be sufficiently specific and measurable for objective verification. Additionally, Wiegers and Beatty (2013) explain that traceability, the linking of requirements to their origins and related design and testing artifacts, facilitates impact analysis and effective change management.

Regarding feasibility, Sawyer (2009) notes that requirements must be realistically achievable within technical, budgetary, and time constraints to ensure project success. Furthermore, Sawyer (2009) emphasizes the need for requirements to be understandable by all stakeholders, including those without technical expertise, to encourage collaboration and consensus. Lastly, Wiegers and Beatty (2013) describe modifiability as the ease with which requirements can be updated or changed as the project evolves, supporting flexibility in response to new insights or changing business needs.

Collectively, these quality attributes, as detailed by these authors and standards, provide a strong foundation for crafting quality requirements that guide successful software development and minimize costly errors. Ambiguity in requirements is widely recognized as a source of defects, delays, and increased costs in software projects as it often comes from unclear, incomplete, or inadequately validated information. Conducting ambiguity reviews and systematically applying quality attributes throughout requirements engineering helps detect and resolve ambiguities early, preventing costly issues in later stages of development.

In summary, quality attributes act as foundations that constrain the expression of requirements, ensuring they are clear, complete, consistent, and testable. This structured approach significantly reduces ambiguity, leading to improved communication, fewer misunderstandings, and ultimately higher-quality software outcomes.

2.6.2.2 Knowledge on Ambiguity

The study by Chantree, de Roeck, Nuseibeh, and Willis (2006) highlights that the awareness of ambiguity is still lacking on the part of readers. Nearly two decades later, this challenge remains relevant as more recent studies also report that ambiguity in natural language requirements continues to pose significant difficulties due to factors such as diverse stakeholder backgrounds and limited domain knowledge (Araújo et al., 2025; Dalpiaz, van der Schalk, & Lucassen, 2018). Hence, despite technological progress, ambiguity awareness remains an ongoing issue in RE. Chantree et al., (2006) also discussed on Macaulay's (1996) study that identified two common scenarios where requirement engineers may encounter issues with textual requirements:

1. Requirement engineers recognize the presence of ambiguity in requirement statements but choose to overlook it.

2. Requirement engineers fail to identify occurrences of ambiguity when reading or writing textual requirements.

The research emphasizes that it is vital for requirement engineers to actively explore and comprehend ambiguity in requirements to ensure clarity, mitigate risks, align with stakeholder needs, assure quality, and facilitate effective communication and collaboration throughout the requirements engineering process.

Gervasi et al., (2019), emphasized that ambiguity in natural language requirements is a defect to be avoided and that understanding it is key to designing effective detection methods. Overlooking or being unaware of ambiguous requirement statements can have significant consequences, leading to misalignment with stakeholder expectations, increased project risks, and breakdowns in communication during system development.

Hence, the research underscores the importance of requirement engineers developing a keen eye for ambiguity and the necessary skills to analyze, clarify, and document requirements in a precise, unambiguous manner. This competency is essential for requirement engineers to fulfill their role effectively and contribute to the successful delivery of software systems.

The study by Haron and Ghani (2015) provides insights into ambiguity awareness among industrial IT practitioners. They assess the level of awareness from industrial IT practitioners towards the occurrence of potential ambiguity in SRS. The result shows the level of ambiguity awareness among them is very low. Thus, it is important to explore the ambiguity of knowledge because it will help in writing a quality SRS without ambiguous words.

According to Nuseibeh and Easterbrook (2000), the individual designated as requirements engineer shoulders the primary responsibility of eliciting, analyzing,

documenting, and validating a comprehensive set of requirements for a system. This pivotal role is also known by various titles, including requirements manager, business analyst, system analyst, or simply an analyst.

Functioning as a vital link between the stakeholders and the software development team, the requirements engineer facilitates the smooth flow of requirements. Regular communication with both stakeholders is essential, allowing the required engineer to address challenges and foster collaboration effectively. Throughout the software development life cycle, the requirements engineer remains engaged at various stages. Once the baseline requirements are established, the focus shifts to managing the requirements specification. Furthermore, the requirements engineer assumes the responsibility of overseeing task execution to ensure the fulfillment of all requirements is verified.

The commonly held belief that anyone can write down requirements, given the use of a natural language, has been challenged by recent research. As Gupta et al. (2019) observe, practitioners frequently question what specific skills are required to elicit complete, correct, and unambiguous requirements. This suggests that the process of requirements specification is not as straightforward as it may appear and demands a certain level of expertise from those responsible for documenting the system's functional and non-functional requirements.

A study conducted by Coughlan and Macredie (2014) found out that without training, expertise or motivation, unprepared requirements engineers tend to face difficulties to understand and to follow good requirements methods which eventually leads to poor SRS. Thus, to avoid such a case happening, Zowghi and Paryani (2003) further highlight the importance of writing skills for requirement engineers, stating that they must possess a range of capabilities to produce precise, concise, consistent, clear, and unambiguous statements of requirement.

During the study phase in computer science and software engineering programs at universities, it is indeed essential to ensure that the RE curriculum adequately covers the full range of skills required for a competent engineer. This holistic approach to RE education is crucial to prepare students for the realities of their future roles in software development.

There are three key skill areas that every requirement engineer must develop. The first one is interviewing and groupware skills for effective requirement elicitation and validation. Effective requirement engineering begins with strong interviewing and group collaboration skills. One essential ability is active listening and being fully present, understanding stakeholder needs, and identifying implicit requirements. This skill allows requirement engineers to extract accurate and complete requirements during stakeholder discussions. Additionally, facilitation techniques such as brainstorming, structured workshops, and use of groupware tools (e.g., shared documents, online whiteboards) enable collaborative requirement gathering, especially in distributed or cross-functional teams. Alongside these, it is crucial to manage stakeholder expectations and address conflicting interests by applying negotiation and interpersonal communication skills. This ensures alignment and reduces misunderstandings throughout the requirement development process.

Secondly, analysis and modeling skills for problem-solving. Requirement engineers must possess strong analytical and modeling skills to interpret and refine stakeholder inputs. Techniques for requirement analysis and decomposition such as functional decomposition, use case analysis, and gap analysis help break down complex needs into manageable, testable requirements. Proficiency in requirement modeling tools and notations (e.g., UML diagrams, data flow diagrams, user stories) further supports the visualization and validation of requirements. These skills are grounded in critical thinking and structured problem-solving, enabling engineers to identify ambiguities, inconsistencies, and potential design issues early in the development lifecycle.

Lastly, effective technical writing skills for specifying requirements are essential for specifying requirements clearly and unambiguously, ensuring that all stakeholders have a common understanding, which helps reduce misunderstandings, errors, and costly rework during software development. The ability to produce precise, concise, and unambiguous requirement statements ensures that all stakeholders interpret the requirements in the same way. This involves mastering technical writing practices that avoid vague terms, redundant information, and complex sentence structures. Moreover, knowledge of industry-recognized requirement specification standards (such as IEEE 830 or ISO/IEC/IEEE 29148) and templates ensures consistency and completeness. Finally, requirement engineers must be able to communicate complex technical information clearly to diverse audiences including developers, testers, and business stakeholders using language tailored to their level of technical understanding.

Furthermore, the study conducted by Zowghi and Paryani (2003) emphasized that communication skills come in two forms: 1) Oral communication skills are needed that would equip students to interact with the clients or users in a language that they can understand. Moreover, interviewing skills are essential that would enable requirement engineers to ask the right types of questions from the stakeholders who take part in requirement elicitation and 2) Written communication skills are essential such that engineers will learn how to write requirement specifications that exhibit quality of attributes such as being precise, concise, unambiguous, consistent, complete, and thus correct. In Macaulay (1996) studies, some of the desirable skills for requirement engineers are mentioned:

1) Interviewing skills: Ability to effectively communicate with individuals and groups to elicit essential requirements information.
2) Listening skills: Capacity to thoroughly comprehend stakeholder needs and uncover implicit requirements.

3) Analytical skills: Proficiency in investigating, evaluating, and decomposing high-level information into comprehensive, lower-level requirements.
4) Facilitation skills: Capability to lead productive requirements elicitation workshops and discussions.
5) Observational skills: Ability to validate data and expose new areas for further requirements gathering.
6) Writing skills: Expert in communicating requirements information clearly and effectively to diverse stakeholders.
7) Interpersonal skills: Competence in negotiating priorities and resolving conflicts among project stakeholders.
8) Modeling skills: Proficiency in representing requirements information using graphical representations to complement textual documentation.

This expanded list underscores the breadth of skills that requirement engineers must possess to excel in their role. The combination of interpersonal, analytical, writing, and modeling capabilities is essential for effectively eliciting, analyzing, and specifying requirements that meet the needs of all project stakeholders. Moreover, considering the human perspective in all phases of RE process, the requirement engineers should understand that the process is deeply rooted in human cognitive, emotional, motivational, and social processes. The paper "Understanding the Problems of Requirements Elicitation Process: A Human Perspective" by Apshvalka, Donina, and Kirikova (2009) highlights several key areas engineers should consider in human perspective mitigating common problems during any of the RE phase:

1. Recognize Elicitation as a Knowledge-Intensive Process: Requirement elicitation is fundamentally a knowledge-intensive process where value is created through the fulfillment of participants' knowledge requirements. Engineers must recognize that the success of the process depends heavily on the knowledge held by various participants and the productivity of their collaboration.

2. Focus on Communication, Collaboration, and Understanding: The core of successful elicitation lies in effective communication, active collaboration, and mutual understanding among different stakeholders and between stakeholders and the requirements engineer. Engineers need to be aware of common communication problems, such as implicit knowledge, misperception, and disagreement, which can hinder the process.
3. Address the Human-Centered Nature of Problems: Many problems in requirements elicitation, such as missing requirements, reluctant participation, misperception, and disagreement, are directly attributable to human factors. Engineers should understand that while technical solutions exist, human cognitive, emotional, motivational, and social processes are often the underlying causes of misunderstandings and incompletely expressed information.
4. Acknowledge the Subjectivity of Knowledge: Knowledge is inherently personal and resides within human minds. As Davenport and Prusak (1998) state, knowledge is a "fluid mix of framed experience, values, contextual information, and expert insight." Engineers must understand that stakeholders' knowledge is subjective and influenced by their unique experiences and perspectives, which requires careful interpretation and alignment.
5. Be Aware of Natural Language Limitations: Despite the importance of face-to-face communication, natural language itself is prone to misinterpretation. Requirements engineers must be cautious and employ techniques that go beyond simple verbal exchanges to ensure clarity, as reliance solely on natural language can lead to misunderstandings.
6. Understand Cognitive and Social Processes: The alignment of different conceptual systems and effective human communication involve complex cognitive and social processes. Engineers should strive to understand how these processes work and influence each other to interpret the usability of different existing requirement elicitation methods and techniques more effectively.

7. Identify and Address Stakeholder Characteristics: Stakeholders come with different interests, expectations, and domain knowledge. Their relevant knowledge, experience, and expectations about the future information system are crucial. Engineers need to actively identify these characteristics and how they might influence the requirements and the elicitation process.

By considering these human aspects, requirement engineers can better choose appropriate elicitation approaches and methods, proactively address potential problems, and improve the quality of the elicited requirements. Possessing essential skills such as clear communication, analytical thinking, and technical knowledge is fundamental for requirements engineers. However, the true challenge lies in applying these skills effectively to produce clear, unambiguous requirements that guide the development process. Effective requirement writing is essential for minimizing ambiguity and ensuring that all stakeholders share a consistent understanding of the intended functionality. One of the most important principles is to ensure that each requirement conveys only one clear meaning. According to INCOSE (2023) guidelines, requirements should be atomic, each addressing a single concept or functionality to avoid compound or conjoined statements that lead to confusion or multiple interpretations. Ambiguity is often introduced through the use of vague or subjective words, such as “fast,” “user-friendly,” or “appropriate.” Robertson & Robertson (2012); Gervasi & Zowghi (2005) state that these terms should be avoided and replaced with specific, measurable criteria. For instance, rather than stating that a system should be “fast,” it is more effective to specify that “the system shall respond within three seconds”.

Simplicity and clarity in language also play a critical role. Requirements must be written using straightforward, unambiguous sentences. According to Pohl (2010), any domain-specific terminology, technical jargon, or acronyms used in software requirements should be clearly defined upon first use or included in a dedicated glossary. This ensures consistent understanding among stakeholders, especially when teams have diverse technical backgrounds. Additionally, as mentioned by Wiegers & Beatty (2013) regular

peer reviews and stakeholder walkthroughs are effective practices for identifying ambiguity, errors, and inconsistencies. Collaboration during these reviews enables early clarification of intent and improves requirement quality.

Beyond these collaborative checks, adhering to specific linguistic guidelines during the writing process itself is fundamental to achieving maximum clarity and preventing various forms of ambiguity. One common source of ambiguity is the improper use of pronouns. Pronouns like “it,” “this,” or “they” should always refer to a clearly identifiable noun. When in doubt, it is safer to repeat the noun to eliminate potential confusion (Cleland-Huang et al., 2006). Furthermore, separating requirement statements from their rationale enhances readability and ensures that each requirement remains concise and testable. Passive voice should also be avoided, as it obscures the actor responsible for an action. Lastly, special attention should be given to quantifiers and conditional statements (e.g., “all,” “at least,” “if-then”) as these often introduce logical ambiguity. Using structured templates such as the EARS (Easy Approach to Requirements Syntax) or Rupp’s Boilerplate can help enforce consistency and syntactic clarity throughout the document (Mavin et al., 2009).

By following these writing guidelines, requirement engineers can significantly reduce syntactic and semantic ambiguity. This not only ensures that requirements are clearly understood and implemented as intended but also improves traceability, testability, and maintainability throughout the software development lifecycle.

2.6.2.3 Requirement Pattern

According to Palomares Bonache (2016), requirement patterns are defined as recurring structures in requirements specifications that can be abstracted and reused to improve the quality and consistency of requirements. Expanding on this concept, Araújo et al. (2025) emphasize that requirement patterns capture domain knowledge and best practices and can be used to guide the elicitation and formulation of new requirements, as well as to analyze and improve existing ones. These patterns are not specific to a particular system or project but represent generic solutions to recurrent problems that can be applied across multiple software systems within the same domain. The key aspects of this definition of requirement patterns in software requirements include:

1. Recurring structures: Requirement patterns represent common, reusable elements in requirements specifications.
2. Abstraction and reuse: Patterns capture and abstract domain knowledge to enable reuse across different software projects.
3. Quality and consistency: Requirement patterns can improve the quality and consistency of requirements.
4. Elicitation and analysis: Patterns can guide the process of eliciting new requirements as well as analyzing and improving existing ones.

Requirement patterns, also known as requirement templates or boilerplates, have long been an integral part of requirement writing. It is an approach for constructing requirements which uses templates and glossaries. This way of documenting requirements is not difficult and helps to limit the downside from the documentation using natural language which is ambiguity (Arora et al. 2013, 2015).

Hull et al., (2011) referred requirements templates as boilerplates. Boilerplate is an easy and simple approach to standardize the requirements of NL. To formulate a requirement with a boilerplate, it is necessary to select the appropriate boilerplate from a

collection and fill in the blanks (attributes) with data. To understand the above, suppose there is a collection of templates like the following: "The<system>shall be able to<function> <object>of type<qualification>within<performance> <units>", The<system>shall be able to<function> <object>not less than<performance>times per<units>", The<system>shall<function> <object>every<performance> <units>". A requirement engineer chooses the appropriate template structures and provides in the words in angle brackets with data. For example, the last template can become as follows: The<coffee machine>shall<produce> <a cold drink>every<5> <seconds>".

In their paper "Automated Checking of Conformance to Requirement Boilerplates via Text Chunking: An Industrial Case Study", Arora et al. (2013) explore this specific application of requirement boilerplates: The authors state: "Requirement boilerplates can be utilized as a foundational framework for conducting preliminary checks on requirements." The boilerplates offer templates for the grammatical structure of requirement sentences to ensure requirements conform to expected formats and conventions. Arora et al., (2013) further explain that their proposed approach leverages "text chunking" techniques to automatically analyze requirements and assess their compliance with the boilerplate structure. This allows for efficient identification of:

1. Missing information or incomplete requirement.
2. Inconsistencies in terminology and phrasing.
3. Potential ambiguities in the stated requirements.

The authors highlight that this automated boilerplate conformance checking serves as an important preliminary step in the requirements validation process, helping to catch issues early before they propagate downstream. Their RUBRIC tool demonstrates the practical application of this approach in an industrial setting, showing how boilerplates can improve the quality and traceability of requirements. By leveraging the boilerplate framework, organizations can enhance the overall quality, clarity, and consistency of their requirements, setting the stage for more effective downstream development activities.

According to Anuar, Ahmad, and Emran (2015) boilerplates can also be applied to various categories of requirements, including capability, function, timeliness, mode, and operational constraints. There are numerous boilerplate templates that have been suggested in the literature. Figure 1 illustrates a sample of widely recognized boilerplates, used in requirements gathering which is referred to as Rupp's boilerplate proposed by Pohl and Rupp (2011), and the Easy Approach to Requirements Syntax (EARS) boilerplate that was introduced by Mavin, Wilkinson, Harwood, and Novak (2009). The EARS boilerplates consist of several key elements that help structure and standardize the way requirements are documented:

1. Condition Clause: This specifies the situational context or condition under which the requirement applies, often starting with phrases such as "When...", "While...", "If...".
2. Trigger Clause: This identifies the event or action that prompts the system to respond, typically beginning with words such as "Whenever...", "Upon...", "On...".
3. System Response Clause: This describes the specific behavior or action the system must perform, using phrases such as "the system shall..." or "the system must...".
4. Optional Inclusion Clause: This allows for the inclusion of additional details or constraints, introduced with words such as "provided that...", "subject to...", "unless...".

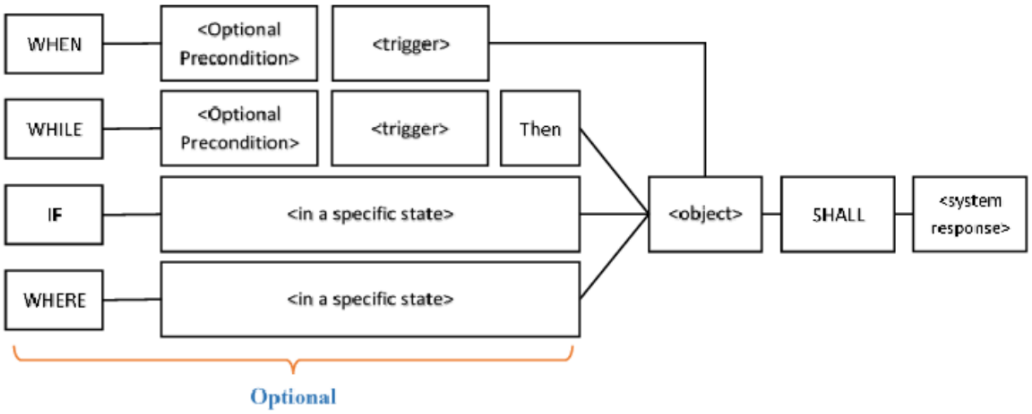


Figure 1: EARS Boilerplate

The structured format of the EARS boilerplate helps ensure requirements are unambiguous and easy to understand, consistent in their syntax and phrasing and traceable to specific triggers and system responses. By adhering to this standardized template, organizations can improve the quality and clarity of their requirements documentation, making it easier to validate, manage, and implement them effectively.

The EARS boilerplate is just one example of how a structured, template-based approach can enhance requirements engineering practices. Other boilerplate frameworks, such as the Volere template or the IEEE 29148 standard, follow similar principles to achieve comparable benefits. The key takeaway is that the use of requirement boilerplates, such as EARS, provides a systematic way to capture requirements in a clear, concise, and consistent manner, ultimately leading to improved requirements quality and downstream project success.

The requirement boilerplates proposed by Rupp and the International Requirements Engineering Board (IREB) follow a structured, template-based approach to documenting

requirements. Rupp's boilerplates offer a structured approach to writing requirements by defining a standardized syntax that reduces ambiguity and improves clarity. The boilerplate format is composed of five main elements: (1) Condition, (2) Actor, (3) Action, (4) Artifact, and (5) Quality Constraint. The Condition element sets the context under which the requirement is valid. It describes the situation or trigger that activates the requirement and often begins with phrases such as "If…", "When…", or "Whenever…". This element ensures that requirements are only applied under clearly defined circumstances, thereby avoiding misinterpretation.

The Actor element identifies the entity responsible for executing the requirement. This can be a human stakeholder, a system component, or an external actor interacting with the system. Clearly specifying the actor ensures that the requirement is assigned to the correct participant in the system. The Action element outlines the functionality or behavior that the actor must perform. It uses precise and active verbs to describe what the system should do in response to the given condition. This component forms the core of the requirement and directly addresses the functional expectations. The Artifact element refers to the object, data, or information that is involved in the action. It could be a digital file, a document, a physical product, or any other entity relevant to the system operation. Including the artifact helps define what the action is applied to, making the requirement more concrete. The final element, Quality Constraint, specifies any non-functional aspects associated with the action. This includes requirements related to performance, usability, security, reliability, or other quality attributes. By incorporating constraints, this element ensures that the system not only functions correctly but also meet quality expectations.

The structured nature of Rupp's boilerplates contributes significantly to producing high-quality requirements. It promotes consistency, enhances understandability, and improves traceability by clearly linking actors, actions, artifacts, and constraints within a uniform format. This helps requirement engineers to systematically capture and validate requirements while minimizing syntactic and semantic ambiguities.

By adhering to this standardized template, organizations can improve the quality, consistency, and testability of their requirements documentation. The boilerplate approach also facilitates requirements reuse and traceability throughout the development lifecycle. Rupp's requirement boilerplates provide a systematic way to capture requirements in a clear and concise manner, ultimately leading to enhanced requirements engineering practices and improved project outcomes.

According to Pohl and Rupp (2011), the first step in applying the requirements template involves determining the degree of legal obligation associated with each requirement. This step is essential for understanding the priority and criticality of requirements during specification and implementation. Requirements are categorized into four levels based on their legal or strategic necessity: legally obligatory, urgently recommended, future, and desirable.

Legally obligatory requirements are those that must be implemented without exception. These are driven by laws, regulatory standards, or binding contractual agreements. Non-compliance with such requirements can lead to serious legal consequences, financial penalties, or rejection of the system by regulatory authorities. Common examples include compliance with data protection laws (e.g., General Data Protection Regulation (GDPR)), adherence to accessibility standards, or fulfillment of safety regulations in critical domains such as healthcare or aviation.

Urgently recommended requirements, while not legally mandated, are strongly advised due to their strategic or operational importance. These requirements are often tied to essential user needs, core system functionalities, or business-critical processes. Although not enforced by law, failing to implement them could lead to project failure, user dissatisfaction, or loss of competitiveness. Therefore, they are typically prioritized right after the legally obligatory requirements.

Future requirements represent enhancements or feature that are not needed in the current version of the system but are planned for future releases. These are often identified during elicitation as part of long-term strategic planning. Documenting future requirements provides a foundation for system evolution and allows teams to design extensible architectures that accommodate later changes more easily.

Desirable requirements are optional and usually provide added value rather than essential functionality. These may include features that enhance usability, aesthetic improvements, or additional tools that make the system more pleasant to use. While beneficial, these requirements are generally implemented only if time, budget, and resources permit, and they are typically given the lowest priority. By classifying requirements based on their legal obligation and strategic importance, requirement engineers can better manage scope, prioritize development efforts, and ensure compliance with critical regulations. This classification also supports better communication with stakeholders regarding expectations and trade-offs.

This categorization helps organizations and development teams prioritize and manage requirements based on their relative importance and obligation level. It supports more strategic decision-making around which requirements must be fulfilled, which are highly recommended, and which can be addressed in future iterations. The key is to strike the right balance between meeting mandatory requirements, addressing urgent needs, planning, and incorporating desirable features - all while optimizing the use of available resources. The second step is the core of the requirement. In this step, the functionality of the requirement is determined with <process>, for example the system store and prints etc. The third step characterizes the activity of the system in three following categories: (a) autonomous system activity, for example the process is performed autonomously by the system (process verb), (b) user interaction, e.g., the process is provided as a service for the user by the system (provide), and (c) the interface requirement, e.g., an external event

triggers the system to execute the process (be able to), i.e., the system is waiting for a message or data to react. In the fourth step, the process-verbs are completed with objects. Some verbs can have more than one object. In the fifth step, logical and temporal conditions are determined. The system executes processes under these conditions. Figure 2 shows the diagram of the boilerplates of Pohl and Rupp. Another boilerplate template as illustrated in Figure 3 has been introduced by He Jiaying et al., (2024).

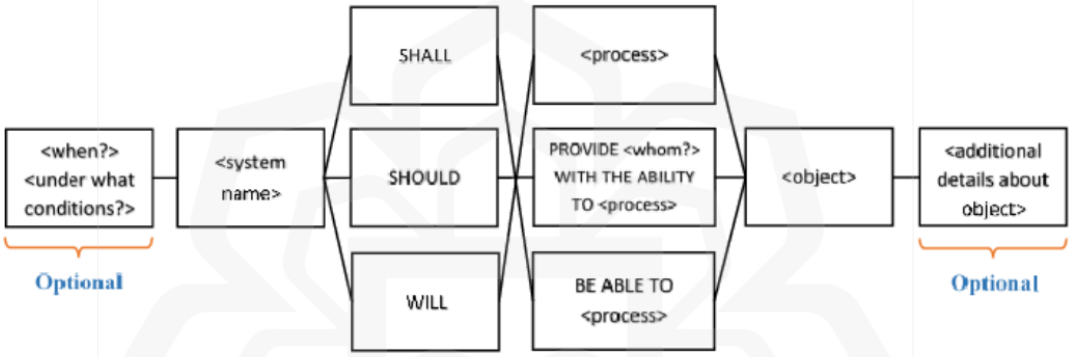


Figure 2: Rupp's Boilerplate

The key elements shown include:

1. "When" clause - Specifying the conditions under which the requirement applies.
2. "System name" - Identifying the system or component the requirement is related to.
3. "Shall" - Stating the mandatory or required behavior.
4. "Should" - Indicating a recommended or preferred behavior.
5. "Process" - Referring to the system function or process that needs to be provided.
6. "Be able to" - Describing the capability the system must have.
7. "Object" - Specifying the entity or artifact the requirement is focused on.

8. "Additional details about object" - Allowing for inclusion of supplementary information.

Requirement type	prefix	Body	Suffix
conditional (V1)	<System location> / *n <System status>	如(果)/ 只要 if / as long as	那么 / 则 then
		*n 【<System component> <System component status>】,	*n 【<System Components> (optional) 就 / 会 at once (optional) <System behavior>】
conditional (V2)	<System location> / *n <System status> <System location> / *n	只有 / 需要 Only/Required	<executive subject> (optional) *n 才可 can /will (optional) <User behavior> <System component> <Action description>
		【 <User behavior> <action object> 】*n / <System component> <System component status>】	
		时/后, when/after,	
State-Driven (V1)	<System status>	<System component > 随着 / 跟 / 与 / 跟随 / 根据 follow <System status> 而/进行 while/proceed (optional)	<system behavior> <Action description>
State-Driven (V2)	<System location> / *n		有 has <System component>
State-Driven (V3)	<System status>	【<System component > <system behavior>】 / 【 <interaction behavior> <interaction element> 】后 then, / 并 / 且 and	<System behavior> <System Components> (optional) *n
Autonomous (V1)	<System location> / *n	<System component > 默认/自动/定时 will automatically/timely	<System behavior> <System Components> (optional) *n
Autonomous (V2)	<System status>	<System component > (optional) 默认 by default(optional) 有/是/在/显示 have/be/at/show	<System component > <System component description> (optional)
Autonomous (V3)	<System location> / *n	<System component > (optional)	每 every <duration> <System behavior> 一次 once, <action description> (optional)
Autonomous (V4)	<System status>	<System behavior>	为 <duration > 一次
User-Interaction (V1)	<System location> / *n <System status>	<User behavior> <System Components> *n	<System behavior> *n <System Components> (optional)
		时 / 后 when/then (optional)	
User-Interaction (V2)	<System location> / *n <System status>	*n 将 <system properties > <User behavior> 到 <System Components>	<System behavior> *n <System Components> (optional)
		时 / 后 when/then (optional)	
Interface	<System location> / *n <System status>	<System Components/Users> (optional) *n <System behavior> <System Components> (optional),	<System behavior> *n <System Components> (optional)
		时/之后/后, when/ after,(optional)	
Optional Function	<System location> / *n <System status>	<user> (optional) 可 can	<System Components> (optional) *n 将/可 will (optional) <System behavior> <system properties > (optional)
		<User behavior> *n <System Components>	
Functional Containment(V1)	<System location> / *n	<System Components>	*n 不 / 不可 cannot <System behavior> <system properties > (optional)
Functional Containment(V2)	<System status>		只 only <System behavior> *n <System Components> (optional)
Functional Containment(V3)	<System location> / *n <System status>	<System Components>	为 is / 无 have no *n <system properties >

Figure 3: The boilerplate for Chinese software requirements

Initially, the authors reviewed international and Chinese standards as well as existing boilerplate frameworks such as EARS and Rupp's to identify key structural elements. They then involved senior software engineering experts who analyzed a large corpus of real Chinese requirements to categorize common ambiguity types and requirement scenarios. Using these insights, the authors developed a modular template structure consisting of prefixes, main bodies, and suffixes. Prefixes set the context, main bodies describe system behaviors with flexible placeholders, and suffixes add conditions or constraints.

These templates are tailored to conform with Chinese grammar while aligning with international standards, ensuring both linguistic accuracy and practical applicability. The boilerplates are also categorized into different types based on requirement contexts, enabling precise and consistent expression across varied scenarios. The design was further validated through expert feedback and practical reconstruction of requirements, demonstrating improvements in clarity, completeness, and verifiability. The authors identified several key issues in Chinese SRS that negatively impact understanding, implementation, and project success. Ambiguity is a major problem, occurring at multiple linguistic levels including syntactic, phonological and semantic making the requirements difficult to interpret. Structural problems, such as poorly organized and inconsistent requirement statements further reduce the readability. Thus, the authors have developed a set of chinese-specific software requirement boilerplate templates for writing requirements, designed to enhance clarity and reduce ambiguity in natural language requirements. It combines linguistic rules and international standards with experts' agreement to provide a usable, standardized template for Chinese software projects. aiming to improve project success rates and communication. Based on this recent research, this study's development of linguistically adapter boilerplates remains highly relevant and emerges as a highly effective approach, providing a much-needed standardized framework to improve clarity,

reduce ambiguity, and enhance the overall quality of SRS. By enforcing standardized syntactic and semantic patterns, they help mitigate the ambiguity often associated with natural language descriptions. This structured approach not only improves the precision of requirements but also facilitates better communication among stakeholders, as noted by Antoniou and Bassiliades (2024). This not only reduces the potential for ambiguity but also makes requirements more amenable to automated processing, parsing, and analysis, a significant advantage that supports various requirements validation and management techniques.

Furthermore, boilerplates have been recognized for their effectiveness in addressing key quality challenges such as disambiguation and complexity. Researchers often analyze and apply existing structured boilerplates, whose standardized nature contributes to improved testability, verifiability, and traceability which are essential elements for successful system development. Arora et al., (2014) emphasize that boilerplates, when properly applied, serve as effective tools for reducing ambiguity and enabling automated analysis through natural language processing technique.

To support this, Anuar, Ahmad, and Emran's (2015) research found that the boilerplate technique contributes the most in dealing with the issue of disambiguation compared to other key requirements quality attributes, such as testability, verifiability, and traceability. The result suggests that boilerplate technique has been found beneficial to overcome one of the quality problems in SRS, which is ambiguity.

For a boilerplate to be effective, one needs to check whether the boilerplate has been properly applied by the requirement analysts. It is vital to check whether a set of requirements indeed conforms to the boilerplate. Usually, conformance to boilerplates is typically verified manually through requirements review. Nonetheless, as Arora, Sabetzadeh, Briand, and Zimmer (2014) discussed in their studies, this should preferably

be executed automatically, as manual checking of conformance to a boilerplate can be laborious and error prone.

2.6.2.4 Ambiguity Rules

Ambiguity in software requirements occurs when a requirement can be interpreted in multiple ways, which often leads to misunderstandings, implementation errors, and increased costs during software development. To address this challenge, ambiguity rules are applied as guidelines or techniques aimed at detecting, preventing, or resolving different types of ambiguities in requirement specifications. These rules help to identify lexical, syntactic, semantic, and pragmatic ambiguities early in the requirements engineering process, thereby improving clarity and reducing risk.

Gleich, Creighton, and Kof (2010) emphasize that one effective way to detect syntactic ambiguity in software requirements is by using POS tagging. Pham (2020) defines POS as the process of assigning a word in a text with its corresponding part of speech based on its definition and its relationship with adjacent words in a phrase, sentence, or paragraph. This aligns with the general understanding in NLP, where each word is labeled according to its grammatical role, such as noun, verb, adjective, or adverb, to support syntactic and semantic analysis. This tagging helps identify structures in sentences that are commonly linked to ambiguity.

Certain grammatical forms like passive voice, adjectives, and adverbs are known to cause ambiguity because they can make sentences unclear or open to multiple interpretations. For example, in the sentence “Failure of any other physical unit puts the program into degraded mode,” each word can be assigned a POS tag:

1. Failure/NN (noun),
2. of/IN (preposition),

3. any/DT (determiner),
4. other/JJ (adjective),
5. physical/JJ (adjective),
6. unit/NN (noun),
7. puts/VBZ (verb),
8. the/DT (determiner),
9. program/NN (noun),
10. into/IN (preposition),
11. degraded/VBN (past participle),
12. mode/NN (noun).

Tagging each word in this way enables the identification of patterns that may indicate ambiguity. For instance, the use of passive voice which often appears when a verb like “be” is followed by a past participle (tagged as VBN) can hide the actor performing an action, making it unclear who is responsible. Similarly, Berry (2006) explains the use of adjectives (JJ) and adverbs (RB) can introduce subjectivity or vagueness, such as words like fast, efficient, or clear, which might be interpreted differently by different readers. The most important POS tags to focus on when detecting ambiguity are:

1. VB (any verb form),
2. VBN (past participles, e.g., been, done),
3. NN (any noun),
4. JJ (adjectives),
5. RB (adverbs).

These tags help to identify specific words or structures that commonly contribute to ambiguous requirements. For example, a requirement like “The system shall be able to quickly process input data” contains the adverb (RB), which is vague unless a specific performance metric is provided. Fortunately, POS tagging tools such as TreeTagger are highly accurate, achieving up to 96% precision. TreeTagger is a widely used POS tagging

tool developed by Helmut Schmid (1995) at the Institute for Computational Linguistics of the University of Stuttgart. The tool annotates text with POS and lemma information and supports many languages, including English, German, French, Chinese, and more. The software is freely available for research and educational purposes, and executables and parameter files for various languages can be downloaded from the official website with supporting documentation and parameter files for many languages provided are maintained by the university. These tools can automatically analyze requirement sentences and flag potentially ambiguous parts, helping requirement engineers improve the clarity and precision of specifications. In summary, by applying POS tagging and following rules that focus on verbs, passive structures, adjectives, and adverbs, it becomes easier to detect and address syntactic ambiguity in SRS.

By using POS tagging to label grammatical roles, especially verbs, passive structures, adjectives, and adverbs, ambiguities embedded in the linguistic structure of requirement statements can be systematically identified. This is critical because ambiguity in software requirements often arises from linguistic constructs that allow multiple interpretations, complicating accurate understanding and implementation. Ambiguity in software requirements often arises from linguistic constructs that allow multiple interpretations, complicating accurate understanding and implementation. For example, multiple parse trees occur when a single sentence can be structurally interpreted in different ways, leading to ambiguous meanings. Jurafsky and Martin (2021) illustrate this phenomenon by showing how sentences like “The administrator saw the user with the telescope” can be parsed so that the phrase “with the telescope” modifies either the administrator or the user, thus altering the meaning significantly. Such structural ambiguity challenges automated parsing and requires careful disambiguation during requirement analysis.

Conjunctions such as “and” and “or” further contribute to ambiguity due to their logical flexibility. Gervasi and Zowghi (2019) explain that the conjunction “and” may

imply that multiple actions occur simultaneously or sequentially, while “or” can be interpreted as either inclusive or exclusive, depending on the context. Without explicit clarification, this can cause confusion about whether all or some of the connected requirements must be fulfilled, which impacts system design and testing.

The use of adjectives and adverbs often introduces vagueness into requirements. Gleich, Creighton, and Kof (2010) highlight that modifiers such as “fast” or “easy” are inherently subjective and lack clear, measurable definitions unless supplemented with precise criteria. This ambiguity leaves room for divergent interpretations among stakeholders, making it difficult to verify whether the requirement has been met.

Similarly, passive constructions obscure responsibility by omitting the agent performing the action. Gleich et al. (2010) argue that passive voice, as seen in statements like “Data shall be processed,” creates uncertainty about who is responsible for processing, thereby reducing clarity and traceability. Clear assignment of actors is essential to avoid misunderstandings in the implementation phase.

The presence of uncertainty modal verbs such as “should”, “might”, and “could” expresses degrees of possibility rather than obligation, contributing to ambiguous priorities. Pohl and Rupp (2011) note that these modal verbs weaken the enforceability of requirements, making it unclear whether a feature is mandatory or optional. This ambiguity can lead to inconsistent development efforts or misaligned expectations. Moreover, vague quantifiers like “some”, “several”, and “many” lack explicit numerical boundaries, thus failing to clearly define scope. According to Gervasi and Zowghi (2006), such imprecise terms hinder the ability to measure system capacity or performance accurately, which may result in under- or over-engineering.

Referential ambiguity also arises from the use of pronouns such as “it”, “this”, or “they” when antecedents are unclear. Pohl and Rupp (2011) emphasize that unclear

pronoun references reduce requirement traceability and comprehension, as stakeholders may have different understandings of what the pronouns refer to. Finally, the use of placeholders like “tbd” (to be determined), “etc.”, “e.g.”, and “i.e.” signals incomplete or informal requirements, which can cause misunderstandings if left unresolved. Gervasi and Zowghi (2006) identify these as markers of under-specified requirements that require further elaboration to avoid confusion during later development stages.

Table 6: Ambiguity Detection Rules using POS Tagging

No	Paper	Rules	Method
1	A Framework for Detecting Ambiguity in Software Requirement Specification (Sabriye et al., 2017)	Check if any sentence in the POS tagger has more than one parse tree Check any sentence in the POS tagger containing AND and OR logics.	POS Tagging
2	A framework for software requirement ambiguity avoidance (Gupta & Deraman, 2019)	Check if W6H is relevant in any sentence	POS Tagging
3	An approach for detecting syntax and syntactic ambiguity in	Check if any sentence in the Tagged_Sentence is an adjective or adverb.	POS Tagging

	software requirement specification (Sabriye & Zainon, 2018)	This technique can be done by detecting any sentence in Tagged_Sentence that contains the adjective tags “JJ” and adverb tag “RB”.	
4	Automated Detection of Syntactic Ambiguity Using Shallow Parsing and Web Data (Khezri, 2017)	1) Look for occurrences of an Adjective followed by two consecutive nouns in a sentence. 2) Look for a pattern of an adjective followed by more than one noun phrase, connected by a conjunction.	POS Tagging
5	Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources (Gleich et al., 2010)	1) Detect for the Passive is voice by searching the occurrence of the verb “to be”, followed by the past participle, but no further verbs are allowed to occur between “be” and the participle. 2) Look for a word combination.	POS Tagging

In conclusion, applying these ambiguity rules enables requirement engineers to systematically identify and address common sources of syntactic ambiguity, thereby improving the clarity, completeness, and quality of software requirements.

2.6.2.5 Ambiguity Term Repository or Ambiguity Glossary

Effective detection and management of ambiguity in software requirements often rely on comprehensive resources that catalog commonly ambiguous words and phrases. One such resource is an ambiguity term repository or glossary, which helps analysts identify potentially problematic terms early in the requirements analysis process

Gleich et al. (2010) found that the list of ambiguous terms compiled in the Ambiguity Handbook is incomplete, as their research identified a significantly larger set of terms that are commonly perceived as ambiguous. For instance, the study determined that any adjective in comparative form, such as "better" or "faster", can result in misinterpretations among stakeholders. The researchers proposed that the ambiguous terms should be systematically collected from previous studies and stored in an ambiguity glossary. This glossary could then be utilized by requirement engineers to aid in the detection of ambiguity when writing requirement statements.

The findings of Gleich et al. (2010) suggest that the existing resources for identifying ambiguous terminology are insufficient, and that a more comprehensive approach is necessary. By compiling a robust ambiguity glossary, requirement engineers can enhance their ability to produce clear, unambiguous requirements documentation, which is essential for aligning stakeholder expectations and facilitating effective communication throughout the system development process.

Moreover, the research by Gleich et al. (2010) also highlights the need for requirement engineers to have access to specialized tools and resources to support the identification and mitigation of ambiguity in requirements specifications. The development of a centralized ambiguity glossary represents a practical solution to address this challenge,

as it can be leveraged by requirement engineering practitioners to improve the quality and clarity of their work.

Table 7: Ambiguous Glossary

Source of the glossary	**Ambiguous Terms**	**Reference**
Ambiguity Handbook	Misuse of language – all, each, every Positioning of words – only, also, even, and, or Quantifiers – a, all, any, each, one, some, the Pronouns and reference – that, which	Berry, Kamsties, and Krieger (2003)
Research paper	All, each, every, and, or	Feiman et al., (2016)
Research paper	Better, faster, could, should, might, usually, normally 100 percent, he, she, it, tbd, etc, this, up to, everybody, both, a, his, her, its, their, they,	Gleich et al, (2010)

Many of the techniques used to reduce ambiguity during requirement documentation have also proven effective when applied earlier in the requirements engineering process, particularly during the elicitation phase. Since ambiguity often originates from unclear or imprecise stakeholder input, addressing it at the point of capture can significantly improve the clarity and quality of documented requirements. This overlap between elicitation and documentation highlights the importance of treating ambiguity not as an isolated issue but as a concern that spans the entire requirements engineering lifecycle (Kof, 2005; Gervasi & Zowghi, 2005).

Ambiguity in documentation frequently results from unstructured or informal stakeholder expressions. When these expressions are recorded without further refinement, they may lead to vague or contradictory requirements. To mitigate this, researchers and practitioners advocate the use of writing guidelines and structured documentation techniques, such as boilerplate templates. Templates like the EARS proposed by Mavin et al. (2009) and the Rupp's Requirement Template (Rupp et al., 2014) offer syntactic structures that guide the formulation of consistent and unambiguous requirements. Although primarily used in documentation, these approaches have proven useful during elicitation as well. For example, Mohedas et al. (2022) demonstrated that novice requirement engineers who used recommended interviewing practices such as standardized templates and guided prompts were more successful in eliciting relevant and complete requirements from stakeholders. Their study found a clear correlation between the use of these structured techniques and the quality of information incorporated into the final requirements, highlighting the value of such approaches across the requirements engineering lifecycle.

Another effective strategy is the development of a shared terminology base through the creation of a glossary or ambiguity term repository. This practice ensures consistent interpretation of domain-specific terms across the elicitation and documentation stages. Misunderstandings often arise when different stakeholders use the same term with different meanings, especially in large or cross-functional projects. By standardizing terminology early, lexical and semantic ambiguity can be significantly reduced (Sommerville & Sawyer, 1997).

Visual modeling tools such as use-case diagrams, mockups, and process flows also support ambiguity reduction. While commonly employed to supplement textual requirements during documentation, they can also be introduced during elicitation to enhance understanding. Visual representations enable stakeholders to identify missing,

conflicting, or unclear elements in their verbal descriptions, thus facilitating clearer and more complete requirements. Gemino and Parker (2011) found that use case diagrams significantly improved users' comprehension and problem-solving performance, supporting their role in ambiguity reduction.

Furthermore, enhancing the skills of requirement engineers in recognizing and resolving ambiguity is critical for both elicitation and documentation. Training engineers to apply ambiguity rules, detect vague expressions, and use probing techniques empowers them to manage ambiguity proactively (Kamsties, 2007). Similarly, the use of peer reviews and walkthroughs typically associated with documentation can also be adapted during elicitation sessions. These collaborative activities provide an opportunity to validate requirements early and clarify issues before they are formally recorded (Wiegers & Beatty, 2013).

In summary, techniques such as writing guidelines, structured templates, ambiguity glossaries, visual aids, and early review mechanisms contribute not only to the production of unambiguous documentation but also to more effective and accurate elicitation. The integration of these practices across both phases supports the early detection and resolution of ambiguity, leading to higher-quality software requirements.

2.6.2.6 Requirement in Latent Ambiguity

Latent ambiguity in software requirements refers to ambiguity that is not immediately apparent during initial reading or review but only becomes evident during later stages such as design, development, or validation. As noted by Berry, Kamsties (2003), this form of ambiguity often arises from unstated assumptions, implicit domain knowledge, or subtle linguistic cues that require contextual interpretation beyond the surface level of the requirement. For instance, a statement like "The report shall be generated in real time" may

appear clear but can lead to divergent interpretations among stakeholders, depending on their understanding of what "real time" entails in terms of performance expectations or technical feasibility.

Detecting such ambiguity requires more than just formal syntax checking as it demands reasoning that mirrors human interpretive or meta-cognitive awareness. As Bashir et al., (2025) emphasize, effective ambiguity detection involves generating explanations that resemble human reasoning, interpreting context, identifying vague expressions, and clarifying intent. Similarly, Raj et al. (2025) emphasize that resolving ambiguity in requirements demands contextual understanding and the ability to distinguish between multiple plausible interpretations. Engineers who engage in such reflective analysis are better equipped to identify subtle inconsistencies, vague phrases, or hidden dependencies in requirements. For example, when presented with a requirement such as "The system shall provide a fast login process," a meta-cognitively aware analyst may pause and reflect "What does 'fast' mean? Is it measurable? Acceptable to all stakeholders?" leading them to clarify it as "The system shall authenticate users within 2 seconds under normal load conditions.".

Despite the value of reflection, manual review alone is often insufficient especially in large-scale projects. As such, automated tools play a complementary role. Gleich, Creighton, and Kof (2010) demonstrate how part-of-speech (POS) tagging and rule-based linguistic analysis can identify common syntactic patterns associated with ambiguity, such as passive constructions, vague adjectives, modal verbs, and pronouns. These tools serve as effective filters that flag potentially ambiguous statements for deeper review. For instance, in the requirement "The system shall automatically generate reports as needed," automated detection highlights vague terms like "automatically" and "as needed." Although syntactically correct, these expressions lack precision about the conditions under which report generation occurs. This leads to clarification, such as: "The system shall generate a report when a user selects the 'Generate Report' button from the dashboard."

To address the limitations of both manual and automated methods, researchers advocate a hybrid approach. Pohl and Rupp (2011) argue that ambiguity detection is most effective when tool-based flagging is combined with collaborative human reviews. Activities such as stakeholder walkthroughs, peer inspections, and validation workshops allow for multiple interpretations and expose hidden ambiguities that any one individual might overlook. For example, a requirement stating, “The system shall notify the administrator in the event of a failure” may prompt one stakeholder to ask, “Which administrator is the database admin or the security admin?” Another may inquire, “What type of failure?” Such discussions reveal role ambiguity and event ambiguity, which are often missed in solo reviews. The requirement is then refined to: “The system shall notify the system administrator via email when a component in the authentication module fails to initialize.” However, collaboration is not immune to human error. Cognitive biases such as confirmation bias and anchoring can significantly distort judgment by causing reliance on pre-existing beliefs and initial impressions. As Chattopadhyay et al. (2022) demonstrate, these biases are prevalent in software development tasks and often lead to suboptimal decisions, requiring developers to backtrack and revise their work. Their field study revealed that nearly half of developer actions were influenced by at least one cognitive bias, and biased actions were significantly more likely to result in reversals indicating wasted effort and reduced efficiency.

In requirement reviews, these biases may cause stakeholders to unconsciously ignore ambiguity or assume shared understanding. Additionally, inattentional blindness is the failure to notice unexpected issues due to focused attention can lead participants to overlook ambiguous wording while concentrating on other priorities. As Narkter et al., (2025) demonstrate, even when individuals report no awareness of unexpected stimuli, they may still retain perceptual sensitivity to its features, suggesting that attention plays a critical role in conscious detection. For instance, a requirement like “The system shall back up all data daily” might be assumed sufficient until reviewers, prompted by structured checklists,

question: “What constitutes ‘all data’? What time is ‘daily’ scheduled for?” This leads to a clearer revision: “The system shall back up user profiles, logs, and configuration files every day at 2:00 AM server time.” Mitigating these psychological limitations requires deliberate techniques such as rotating review roles, using ambiguity checklists, and fostering a culture of critical questioning.

All meta-cognitive awareness, automated analysis, collaborative review, and bias mitigation form a layered strategy for detecting latent ambiguity. Unlike syntactic ambiguity, latent ambiguity is often obvious from contextual gaps, undefined terms, or misplaced assumptions. The framework developed in this study operationalizes this strategy through the Requirement in Latent Ambiguity component, which augments rule-based analysis with human-centered inspection processes. This integration enhances the ability to reveal ambiguity that is implied, deferred, or context-dependent, ultimately improving requirement clarity, reducing misinterpretation, and supporting more reliable software development.

2.7 REDUCING AMBIGUITY DURING REQUIREMENT ELICITATION AND REQUIREMENT DOCUMENTATION PHASE

Many techniques used to reduce ambiguity in requirements documentation have also proven effective when applied earlier in the RE process, especially during elicitation and documentation. Since ambiguity often stems from unclear or imprecise stakeholder input, addressing it at the point of capture can significantly improve the clarity and quality of the documented requirements. This overlap between elicitation and documentation emphasizes that ambiguity should not be treated as an isolated issue but rather as a concern spanning the entire requirements engineering lifecycle (Kof, 2005; Gervasi & Zowghi, 2005).

Ambiguity in documentation frequently arises from unstructured or informal stakeholder expressions. When these expressions are recorded without refinement, they may lead to vague or contradictory requirements. To mitigate this, writing guidelines and structured documentation techniques, such as boilerplate templates, are widely recommended. Templates like the EARS (Mavin et al., 2009) and Rupp's Requirement Template (Rupp et al., 2014) provide syntactic structures that help formulate consistent and unambiguous requirements. Although primarily used during documentation, these templates have also shown value during elicitation by guiding stakeholders to articulate requirements more clearly through template-driven prompts in interviews or workshops (Friedrich et al., 2011).

Another effective strategy is the creation of a shared terminology base, such as a glossary or ambiguity term repository. This ensures consistent interpretation of domain-specific terms throughout elicitation and documentation. Misunderstandings often occur when different stakeholders use the same term with varying meanings, particularly in large or cross-functional projects. Early standardization of terminology helps reduce lexical and semantic ambiguity significantly (Sommerville & Sawyer, 1997).

Visual modeling tools including use-case diagrams, mockups, and process flows also play a crucial role in reducing ambiguity. While commonly used to supplement textual requirements during documentation, introducing these visual aids during elicitation enhances stakeholder understanding. Visual representations enable stakeholders to identify missing, conflicting, or unclear elements in their verbal descriptions, facilitating clearer and more complete requirements.

In addition to tools and templates, improving the skills of requirements engineers in recognizing and resolving ambiguity is essential. Training engineers to apply ambiguity rules, detect vague expressions, and use probing questions empowers them to manage ambiguity proactively. Collaborative techniques such as peer reviews and walkthroughs, typically associated with documentation, can be adapted for elicitation sessions. These activities provide early opportunities to validate requirements and clarify issues before formal documentation (Wiegers & Beatty, 2013).

In summary, writing guidelines, structured templates, shared glossaries, visual aids, and early review mechanisms contribute not only to producing unambiguous documentation but also to more effective and accurate requirement elicitation. Integrating these practices across both phases supports early detection and resolution of ambiguity, leading to higher-quality software requirements.

2.8 THEORETICAL FRAMEWORK

Figure 4 illustrates the theoretical framework underpinning the study. It comprises six interrelated components that collectively support the process of ambiguity detection which are Requirement Attributes (Kamsties et al., 2001), Writing Guidelines (INCOSE (2023), Robertson & Robertson (2012), Pohl (2010), Requirement Pattern (Arora et al., 2013, 2015), Ambiguity Rules (Berry et al., 2001), Ambiguity Term Repository(Gleich et al., (2010) & Berry et al., 2003), and Requirement in Latent Ambiguity (Dar et al., (2023). Each element contributes to the systematic identification and management of syntactic ambiguity in Software Requirements Specifications (SRS). Their integration reflects a defined architecture, in which components are clearly structured and functionally connected to ensure methodological rigor and practical applicability.

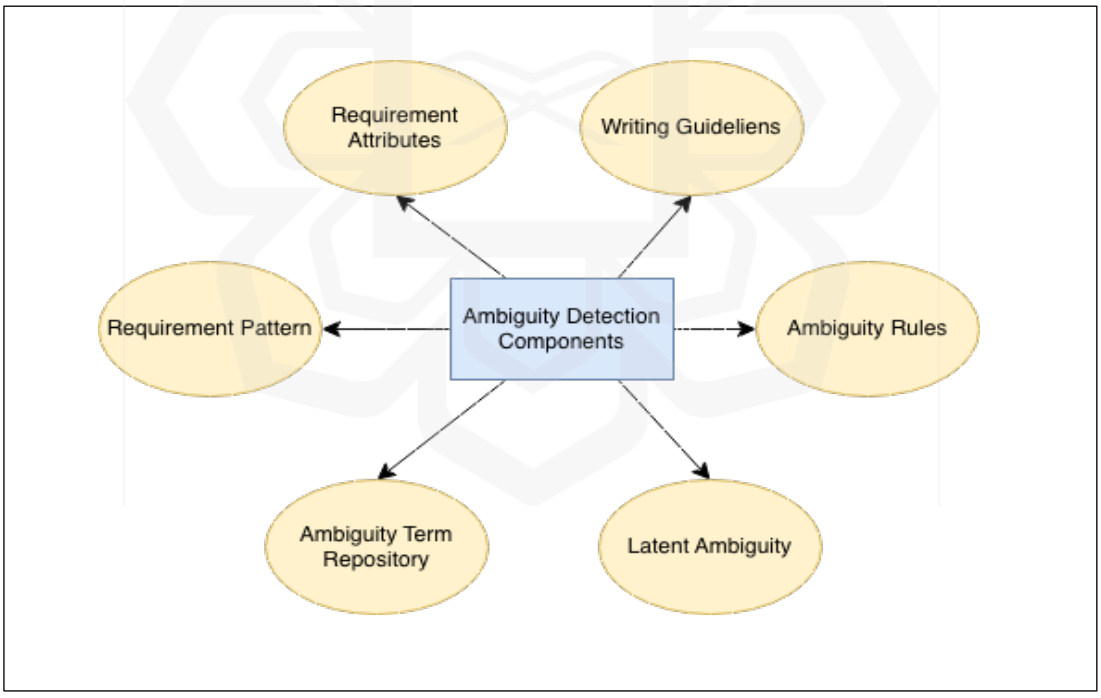


Figure 4: Theoretical Framework

Requirement Attributes define the essential qualities of well-formed requirements, such as clarity, relevancy, completeness, and applicability. These attributes serve as the foundational criteria against which ambiguity can be assessed. Writing Guidelines translate these attributes into actionable drafting principles, offering structured advice on how to phrase requirements to minimize misinterpretation. These guidelines ensure that requirement authors consistently apply best practices during documentation.

Requirement Patterns provide standardized templates that operationalize the writing guidelines. By offering repeatable structures for expressing requirements, patterns reduce variability and enforce uniformity, which in turn enhances readability and reduces the likelihood of syntactic ambiguity. Ambiguity Rules function as diagnostic mechanisms that identify specific syntactic constructs known to cause confusion, such as vague quantifiers, ambiguous modifiers, or compound statements. These rules are supported by the Ambiguity Term Repository, which serves as a curated knowledge base of problematic terms and phrases frequently associated with ambiguity in technical documentation.

Requirement in Latent Ambiguity addresses a more dimension of ambiguity cases where the ambiguity is not immediately visible but emerges during interpretation, especially in collaborative or cross-functional environments. This element ensures that the framework does not only detect overt syntactic issues but also accounts for contextual and interpretive challenges that may compromise requirement clarity. These six components form a coherent and integrated framework. Their interrelationship ensures that ambiguity is addressed at multiple levels from initial drafting to post analysis providing both preventive and corrective mechanisms. For example, attributes and guidelines shape how requirements are written, patterns enforce consistency, rules and repositories detect issues, and latent ambiguity captures what might otherwise be missed. By grounding the framework in a defined architecture and validating its components through thematic grouping, the study establishes a robust theoretical basis for improving clarity, precision, and reliability in requirements engineering.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 INTRODUCTION

This chapter explains the research methodology used in this study. The chapter begins by describing the overall research design and approach adopted to guide the study. Next, the chapter presents the data collection methods, including the techniques and tools used to gather relevant information. The chapter also outlines how the collected data was analyzed to answer the research questions.

To ensure the quality of the findings, the chapter discusses steps taken to address validity, reliability, and ethical considerations. Finally, the chapter highlights the limitations of the chosen methods and provides details about the design and evaluation of the proposed framework.

3.2 RESEARCH DESIGN

The design of the Syntactic Ambiguity Detection Framework (SADF) was guided by a structured and evidence-based approach. The framework was developed through peer-reviewed literature in the fields of software requirements engineering and ambiguity detection. This process enabled the identification of six core elements: Requirement Attributes, Writing Guidelines, Requirement Pattern, Ambiguity Rules, Ambiguity Term Repository, and Requirement in Latent Ambiguity. These elements were subsequently refined through expert validation to ensure both theoretical grounding and practical relevance.

The relationships between these elements were determined by analyzing their functional roles and how they interact within the ambiguity detection process. Requirement Attributes define the essential qualities of well-formed requirements, such as clarity, completeness, and consistency. Writing Guidelines translate these attributes into practical drafting principles, while Requirement Patterns provide standardized templates that operationalize the guidelines. Ambiguity Rules and the Ambiguity Term Repository serve as diagnostic mechanisms, identifying syntactic issues and problematic terms. Requirement in Latent Ambiguity addresses subtle ambiguities that may not be immediately visible but emerge during interpretation. These elements are not isolated; rather, they form a coherent and integrated system in which each component reinforces the others.

The framework is structured according to the concept of defined architecture, as described by Bass, Clements, and Kazman (2012), which emphasizes the organization of system components, their relationships, and the principles guiding their design. The use of thematic grouping to derive and validate the framework components follows established qualitative research methodologies (Braun & Clarke, 2006; Grbich, 2013), ensuring methodological rigor and transparency. This approach provides a robust theoretical foundation for improving clarity and precision in requirements engineering.

This study adopts a descriptive research design within a quantitative research approach, which is suitable for systematically collecting and analyzing data from requirements engineering (RE) professionals. The goal is to quantify expert perspectives on the detection of syntactic ambiguity in software requirements and to identify patterns, trends, and relationships among key framework components. This approach allows for objective measurement of expert feedback through structured surveys, enabling the evaluation of the framework's clarity, relevance, and practical applicability in the context of SRS development.

An exploratory research design was chosen to investigate the usefulness and effectiveness of the proposed framework. The exploratory nature of this study supports the aim of understanding how experts interpret and assess requirement ambiguity in real-world requirements, and how the framework aligns with industry practices. Data was gathered using a survey, which included a mixture of structured and open-ended questions designed to capture detailed feedback from RE and SE experts. These experts were asked to evaluate the framework and provide their experiences, observations, and suggestions based on their professional background. 25 domain experts in Requirements Engineering reviewed the survey items to assess their relevance and clarity. Their feedback was used to refine item wording and ensure alignment with the intended constructs.

In this study, a formal survey protocol was implemented to ensure ethical integrity and methodological thoroughness throughout the validation of the Syntactic Ambiguity Detection Framework (SADF). Based on guidance from Cohen, Manion, and Morrison (2018), this research methods used the protocol encompassed a structured approach to participant selection, survey administration, and data analysis. Participants were carefully chosen based on their expertise in Requirements Engineering, with a minimum of three years' experience in software development or academic research related to Software Requirements Specification (SRS), ensuring that feedback was grounded in professional practice. The survey was administered online via a secure platform, with clear communication of the study's purpose and informed consent procedures. To enhance reliability, the instrument underwent two iterations, pilot and final to allow refinement based on expert feedback. Quantitative data from Likert-scale items were analyzed using descriptive statistics and factor analysis to assess construct validity, while qualitative responses from open-ended questions were thematically coded to capture the best insights. Results were reported using tables, figures, and reliability indices such as Cronbach's alpha. This protocol ensured that the survey process was ethically sound, methodologically robust, and directly aligned with the study's objective to validate SADF in a real-world context.

The design process of the survey involved both structured and open-ended questions to ensure each component of the framework was assessed for clarity, relevance and practical applicability. The survey questions were primarily based on the framework developed through the literature review. The questions were designed to capture expert perceptions of each element's usefulness and clarity in detecting syntactic ambiguity. A preliminary survey was conducted with a small group of practitioners (n = 10) to evaluate item comprehension and response consistency. Results from this pilot informed minor revisions to improve the questions quality. Open-ended questions were included to allow respondents to elaborate on their experiences, suggest improvements, and identify any overlooked aspects of syntactic ambiguity. These qualitative responses were analyzed thematically and used to refine the framework components. The inclusion of open-ended items ensured that the survey captured both structured feedback and obtain expert insights.

In this study, several measures were taken to ensure that survey questions were correctly placed within the relevant elements of the Syntactic Ambiguity Detection Framework (SADF). First, each Likert-scale item was explicitly mapped to one of the six framework elements, ensuring clear alignment. A pilot study was conducted to refine the wording and confirm that respondents interpreted the items as intended. During the main survey, expert participants provided both structured ratings and open-ended feedback, which allowed for the identification of any misplacements or overlooked aspects. These steps ensured that the survey questions were appropriately aligned with their corresponding framework elements and that the validation process was both systematic and rigorous.

The survey method enables the collection of diverse viewpoints in a flexible, scalable manner while still preserving the depth and richness of open-ended responses. This approach ensures that the framework validation is grounded in expert judgment and reflects practical relevance in real-world scenarios. Based on Figure 5, there are five phases in the research methodology

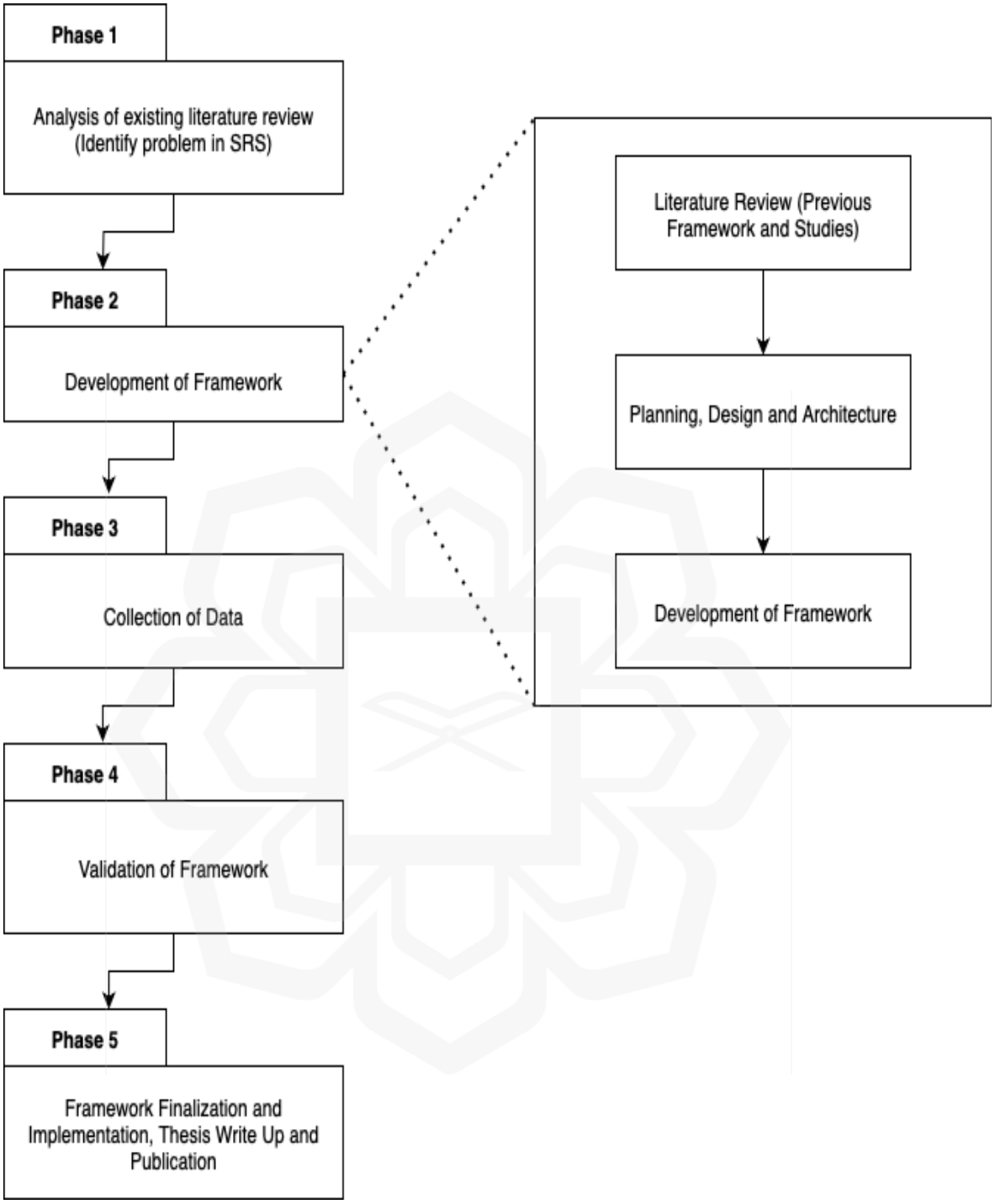


Figure 5: Research Methodology

3.2.1 Phase 1: Analysis of Existing Literature Review

The first phase involved conducting a comprehensive literature review focusing on ambiguity in software requirements specifications, with particular emphasis on syntactic ambiguity. This review aimed to gather and synthesize a broad spectrum of relevant studies to establish a clear understanding of the state of research in this area. To achieve this, relevant academic databases such as IEEE Xplore, ACM Digital Library, Scopus, SpringerLink, and Google Scholar were searched using keywords including "syntactic ambiguity in software requirements," "ambiguity detection framework," and "requirement specification."

Selection criteria prioritized peer-reviewed articles published after 1990, while excluding non-English and non-research-based publications to ensure both quality and relevance. Each selected study was systematically reviewed and summarized with attention to its objectives, methodologies, findings, definitions of ambiguity, and any proposed frameworks. In addition, critical aspects such as the strengths, limitations, and future directions highlighted by each study were captured to provide a balanced evaluation.

Thematic grouping was employed as the principal method to guide both the development and validation of the Syntactic Ambiguity Detection Framework (SADF). In line with the procedures outlined by Grbich (2013) and Braun and Clarke (2006), the process began with familiarization, where the literature and survey responses were read repeatedly to gain a comprehensive understanding of the data. From this stage, recurring issues and ideas were identified, such as the importance of requirement attributes, the role of writing guidelines, and the presence of latent ambiguities. These recurring ideas were then grouped into broader categories that reflected common patterns across the data.

The next stage involved refining these categories to ensure that they were coherent and distinct. Overlapping ideas were merged, while categories that lacked sufficient support were discarded. This refinement led to the definition of clear themes, which in the development phase

became the six framework elements: requirement attributes (Kamsties et al., 2001), writing guidelines (INCOSE (2023), Robertson & Robertson (2012), Pohl (2010), requirement pattern (Arora et al., 2013,2015), ambiguity rules (Berry et al., 2001), ambiguity term repository(Gleich et al., (2010) & Berry et al., 2003), and requirement in latent ambiguity (Dar et al., (2023). In the validation phase, thematic grouping was applied to the open-ended survey responses, where themes such as clarity of the framework, practical usability, and suggestions for improvement emerged. These themes confirmed the relevance of the six elements and provided insights for refinement.

Finally, the results of the thematic grouping were synthesized and reported through tables and figures, ensuring transparency and traceability from the raw data to the framework components. By following this structured process, the study ensured that the SADF was both theoretically grounded in the literature and empirically validated through expert feedback, thereby demonstrating methodological rigor and credibility.

Through critical analysis across these themes, research gaps and opportunities were systematically identified. The insights gained were synthesized into a comprehensive overview that not only highlights the limitations of existing work but also clearly articulates the need for a new, structured framework to address syntactic ambiguity in Software Requirements Specifications. This process ensured that the framework development was firmly grounded in the literature while remaining responsive to practical challenges in requirements engineering.

3.2.2 Phase 2: Development of Framework

The primary objective of this framework is to provide a structured, comprehensive method to systematically detect syntactic ambiguity that commonly arise during requirement elicitation and requirement documentation. This framework is conceptual in nature but designed with practical applicability in mind, incorporating both linguistic and engineering perspectives to enhance requirement clarity and precision.

The development of the syntactic ambiguity detection framework was guided by the findings from an extensive literature review and the identification of gaps in current approaches to managing ambiguity in SRS. It involves a thorough review of existing frameworks and studies related to ambiguity detection in SRS. This review focused on analyzing the strengths and limitations of current approaches and other linguistic-based frameworks. The purpose was to identify gaps, best practices, and foundational concepts that could inform the design of a more comprehensive and effective framework. Insights gained from this literature review helped in understanding the types of ambiguities commonly encountered, the role of requirement quality attributes, and the importance of integrating both automated and manual detection techniques.

Following the literature review, a systematic planning and design phase was undertaken to define the framework's structure and components. This involved determining the key elements necessary to detect syntactic ambiguity effectively. The framework is composed of six key components that collectively address various dimensions of ambiguity detection. First, requirement quality attributes such as clarity, consistency, completeness and unambiguity are fundamental elements integrated into the framework to serve as guiding criteria for evaluating requirements. These attributes ensure that detected ambiguities are assessed with respect to the SRS quality. Second, the knowledge on the ambiguity component incorporates writing guidelines and requirements documentation skills, which are critical in recognizing common linguistic constructs that contribute to ambiguity. This includes providing best practices for clear requirement writing as established in the existing literature.

Third, requirement patterns play a significant role by utilizing standardized templates such as the Easy Approach to Requirement Syntax (EARS) and Rupp's Boilerplate, which are known to reduce ambiguity through structured sentence patterns. These patterns form a baseline for syntactic regularity within requirements. Fourth, derived from linguistic analysis and expert knowledge, these rules identify specific syntactic structures and terms that often lead to ambiguous interpretations. The ambiguity term repository, or glossary, complements these rules by cataloguing frequently ambiguous words and phrases, thus serving as a reference for both automated and manual detection efforts.

Finally, the framework addresses latent ambiguity, which refers to ambiguities not immediately apparent but that present during subsequent phases of development. To detect these, a layered strategy is implemented, combining meta-cognitive awareness, automated tool support, collaborative review process, and mitigation of cognitive biases to enhance detection accuracy beyond surface level analysis. The development process involved an iterative approach, beginning with analysis of theoretical constructs from existing ambiguity detection methods and requirement engineering standards. In the development phase, the framework was constructed based on a defined architecture, which in software engineering refers to a formally articulated structure that specifies the components of a system, the relationships among those components, and the principles guiding their design. This understanding is derived from established scholarly definitions. For example, Bass, Clements, and Kazman (2012) define software architecture as "the structure or structures of a system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." Similarly, Clements et al. (2010) emphasize that architecture provides the set of structures needed to reason about a system, including elements, relations, and properties. Within the context of this study, adopting a defined architecture ensured that the Syntactic Ambiguity Detection Framework (SADF) was systematically grounded in recognized practices, with each of its six elements clearly structured and interrelated to provide a coherent foundation for detecting syntactic ambiguity in Software Requirements Specifications.

The components were elaborated with detailed ambiguity rules formulated from linguistic analyses, and expert validation. The ambiguity term repository was populated with terms identified from literature and practical experience as commonly ambiguous. In addition, writing guidelines and requirement documentation skills were incorporated to enhance the framework's applicability. Special attention was also given to latent ambiguity by integrating meta-cognitive strategies and automated tool support to detect subtle ambiguities that may not be apparent in initial reviews. Throughout the development, initial versions of the framework were drafted and refined through continuous review feedback from preliminary validation and alignment with research objectives.

Diagrams illustrating the framework's overview were developed to clarify the relationships among components and the flow of ambiguity detection ambiguities. Wherever applicable, modeling techniques will be employed to visualize and organize the framework structure.

3.2.3 Phase 3: Collection of Data

The syntactic ambiguity detection framework developed in Phase 2 establishes a comprehensive, structured foundation that guides the subsequent data collection phase. By clearly defining the types of syntactic ambiguities, quality attributes, linguistic rules, and ambiguity term repositories, the framework not only clarifies the scope of ambiguity detection but also informs the specific criteria and methods to be applied during data gathering.

The third phase of the study focused on the collection of data through structured surveys to validate the proposed framework for detecting syntactic ambiguity in SRS. This phase aimed to obtain expert feedback on the framework's components, structure, and applicability in real-world RE practices. The survey questions were formulated based on the components of the proposed framework and employed a combination of 5-point Likert-scale ratings and open-ended sections. The Likert scale ranged from 1 (Strongly Disagree) to 5 (Strongly Agree), enabling participants to quantitatively express their level of agreement with specific statements. The open-ended sections

allowed participants to elaborate on their opinions and provide constructive suggestions, ensuring responses remained both focused and relevant to the framework's evaluation. The survey was designed to evaluate the clarity, relevance, and effectiveness of each component.

The target participants for the survey included individuals with practical or academic experience in RE, software development, or related fields. A target sampling method was used to identify participants with knowledgeable perspectives. The selection criteria emphasized familiarity with software requirements processes and exposure to requirement specification documents. To ensure the quality of the survey, a pilot test of the survey was conducted with a small group of participants to evaluate the clarity and structure of the questions, refine the language, and adjust ambiguous wording. Based on the pilot results, minor refinements were made to improve readability, eliminate ambiguities, and ensure that each item effectively captured the intended feedback. Table 8 provides the summary of the analysis of the data collected from the structured survey conducted as part of the pilot validation.

Table 8: Summary of data collected during Pilot Study

Aspect	**Description**	**Findings**
Participant Profile	Roles and experience level of respondents	1) Background of the participant: QA/Test Engineers (38%), Lecturers (25%), Requirement Engineers (13%), Others (25%). 2) Most had university-level RE training. 3) Experience ranged from <1 year to >6 years.

Survey Method	Structured survey with Likert-scale and open-ended questions	1) Distributed online; included pilot feedback section. 2) Focused on evaluating each framework component.
Feedback	Thematic issues from open-ended responses	1) The survey is clear and understandable. However, there is a request for clearer terminology (e.g., syntactic ambiguity, boilerplate) where the participant suggested to include what happens after applying the framework. 2) Need for measurable success metrics. 3) Recommendation to split some compound survey questions. 4) Proposed AI integration and feedback loop for improvement.
Pilot Survey Feedback	Issues identified during pilot study	1) UI display inconsistencies on mobile vs desktop. 2) Absence of “Not familiar” option in pattern questions. 3) Suggestion to improve phrasing in survey items for clarity.

The finalized survey was distributed online, ensuring ease of access and convenience for respondents. Participation was voluntary, and respondents were informed of the purpose of the research and assured of the confidentiality of their responses. The survey responses were collected systematically, and only fully completed submissions were considered for analysis.

The data obtained through this phase served as a critical input for assessing the framework's practicality, strengths, and areas for improvement, ultimately contributing to its refinement and validation. The detailed results of the survey will be presented in Chapter 5.

3.2.4 Phase 4: Validation of Framework

The fourth phase of this research involved validating the proposed framework for detecting syntactic ambiguity in SRS. The main objective of this phase was to assess the framework's clarity, relevance, practical applicability, and potential to improve ambiguity detection in SRS. This validation phase was essential to determine whether the framework met the expectations and needs of professionals involved in RE and software development.

To achieve this, a structured survey was designed as in Appendix 1 and distributed to a group of domain experts, including academic researchers, requirements engineers, and software practitioners with relevant experience in the field. The validation was conducted via an online structured survey designed to evaluate each component of the framework. The survey consisted of Likert-scale items and open-ended questions, enabling both quantitative assessment and insights. The questions targeted specific aspects of the framework, including the clarity of its structure, the relevance and sufficiency of its components, the framework's coverage of ambiguity types, and its practical utility in real-world software requirement tasks. Participants were selected using purposive sampling to ensure the inclusion of individuals with relevant experience and knowledge in requirements engineering. The main objectives of the survey are:

1. To investigate RE practitioners' perceptions and practices in detecting ambiguity by utilizing the proposed framework.
2. To identify and explore factors that may contribute as new elements to the framework, enhancing its ability to detect syntactic ambiguity.
3. To compare the effectiveness of RE practitioners in detecting ambiguity with and without the utilization of the proposed framework.

As part of the evaluation process, the survey will be answered by RE experts. The RE experts will comprise from several companies as well as RE lecturers from the International Islamic University Malaysia (IIUM). The respondents included business analysts, QA/test engineers, lecturers, and requirement engineers, most of whom had formal training in software requirements practices. Their years of experience ranged from less than one year to more than six years, providing a diverse range of perspectives. The selection of the expert panel members was guided by the following criteria:

1. Extensive experience and expertise in the RE field, as evidenced by their professional roles and affiliations.
2. Representation from both industry practitioners (business analysts) and academic researchers (RE lecturers) to ensure a balanced perspective.
3. Affiliation with reputable organizations, such as well-established IT companies and the International Islamic University Malaysia, to ensure the credibility and relevance of the expert evaluation.
4. Experts who have experience in documenting the SRS.

Following this procedure, the experts were given a structured survey to gather feedback and insights including:

1. Perceived clarity and understandability of the overall framework structure and purpose.
2. Relevance of individual framework components (e.g., requirement quality attributes, writing guidelines, ambiguity rules) in supporting ambiguity detection.
3. Comprehensiveness in covering common types of syntactic ambiguity encountered in real-world software requirements.
4. Ease of use and practical applicability during the requirements elicitation and documentation process.
5. Effectiveness of the framework in identifying ambiguities that are often overlooked during manual reviews.

6. Impact on efficiency, specifically in reducing the time and effort required for writing or reviewing requirements.
7. Comparison with existing practices (e.g., manual techniques or tool-based approaches) used by participants for ambiguity detection.
8. Suggestions for improvement, including missing components, potential integration with automated tools, and recommendations for real-world adoption.

The validation process emphasized expert judgment and feedback. These question items in the survey were designed to assess both the technical robustness and the practical utility of the framework, providing valuable insights for refinement and future implementation. The survey questions are divided into two parts. The first section gathered demographic and participant background information such as the respondent's role, years of experience and familiarity with ambiguity in SRS. The sample questions are as below:

1. What is your role in the organization?
2. Do you have formal training on Requirement Engineering?
3. Tick any requirement patterns (boilerplates or structures) that you are familiar with?
4. How many years of experience do you have in software requirement engineering?

The second section focused on evaluating the framework through both five Likert-scale and open-ended responses. The Likert-scale items measured the framework against five key criteria including comprehensiveness in addressing ambiguity types, ease of application in practical settings, and relevance to existing RE workflows. Table 9 provides the sample of the five Likert-scale questions designed:

Table 9: Five Likert-Scale Questions

Questions	Scale					Feedback
	1	2	3	4	5	
RELEVANCY The proposed framework is clear and understandable.						
SUFFICIENCY The framework components are relevant to detecting functional requirement ambiguity.						
COMPLETENESS The framework covers the general range of syntactic ambiguity types typically encountered in real-world software requirements.						
PRACTICABILITY The framework is easy to understand and can be practically applied during the elicitation and documentation phases.						
USABILITY The framework has the potential to reduce the risk of misinterpretation in software requirements.						
RELEVANCY The framework includes relevant components for identifying and resolving ambiguous terms.						

EFFECTIVENESS The framework effectively identifies ambiguities in software requirements that are commonly missed during manual reviews processes.						
SUFFICIENCY The framework provides sufficient examples or guidance for practical use.						
PRACTICABILITY Compared to your current ambiguity detection practices (e.g., manual review or existing techniques) the framework offers an improvement.						
USABILITY The framework helps reduce the time and/or effort involved in writing or reviewing software requirements.						

For Likert-scale questions, the expert needs to answer the questions based on the level of agreement with the statements (Scale: 1 Strongly Disagree. 5 = Strongly Agree). Meanwhile, the open-ended items were designed to capture additional feedback, suggestions for improvement, and the perceived strengths or weaknesses of the framework. The sample of the open-ended questions for the survey are as below:

1. What improvements or additions would you recommend to enhance the framework's usability, coverage or effectiveness? Please specify any improvements or additions you would recommend to enhance the framework.

2. Have you used any existing techniques or tools for ambiguity detection, including manual methods? (Yes/No)? If yes, please explain how this framework compares to your previous experience?
3. Do you foresee any challenges in implementing this framework in a real-world software project? If yes, please describe them.
4. Based on your experience, are there any components important or relevant for detecting ambiguity that are missing in this framework?

The feedback was analyzed partially to identify recurring opinions, concerns, or suggestions. A high average score (e.g., above 4.0 out of 5) on the Likert items will be interpreted as positive endorsement of the framework. Additionally, constructive comments from experts will be used to refine and improve the framework further. While the validation offers valuable insights into the strengths and applicability of the proposed framework, it is acknowledged that there are limitations. These include the relatively small sample size and the subjective nature of expert evaluations, which may limit generalizability. Nevertheless, the expert feedback provides an essential step in confirming the relevance and potential of the framework to support ambiguity detection in SRS practices.

3.2.5 Phase 5: Framework finalization, Thesis Write Up and Publication

This final phase encompasses the consolidation of all previous phases, leading to the completion of the proposed syntactic ambiguity detection framework and the formal documentation of the research outcomes. After the validation phase, which involves collecting feedback from domain experts through a structured survey, the results were thoroughly analyzed to identify recurring suggestions, strengths, and potential shortcomings of the framework. These insights served as the basis for revising and refining the framework. Any gaps or weaknesses highlighted by the respondents were addressed to enhance the framework's robustness, applicability, and alignment with practical requirements engineering scenarios.

The framework finalization includes integrating expert recommendations into the framework components, refining terminology, clarifying any ambiguous definitions, and ensuring that the structure of the framework supports ease of use by practitioners and researchers alike. Attention will also be given to improving the supporting tools or templates, such as ambiguity glossaries, pattern guidelines, and rule repositories, where applicable.

Following the finalization of the framework, the thesis write-up process will commence. This involves compiling all elements of the research from problem identification, literature analysis, methodology, framework development, validation findings, and conclusions into a structured thesis document. The write-up will follow the standard formatting and academic writing guidelines set by the institution, with clear referencing, figures, and tables to support understanding. Each chapter will be reviewed and revised iteratively to ensure clarity, coherence, and academic standards.

Simultaneously, efforts will be made to distribute the research findings to a wider academic audience. The core contributions of the research, particularly the novel framework for detecting syntactic ambiguity in Software Requirements Specification (SRS) will be drafted into a conference paper or journal article. Targeted publications include conferences and journals specializing in software engineering, requirements engineering, or applied computer science. Submissions will follow the respective publication formats, and the results of the validation phase will serve as evidence of the framework's effectiveness.

The aim of this phase is not only to conclude the research project successfully but also to contribute meaningfully to the body of knowledge in RE by offering a practical, validated solution for addressing syntactic ambiguity in SRS. The publication efforts also ensure that the work can be cited, extended, or applied by other researchers or practitioners in the RE field.

CHAPTER FOUR

FRAMEWORK DEVELOPMENT AND VALIDATION

4.1 INTRODUCTION

This chapter presents the development of the proposed syntactic ambiguity detection framework, followed by its validation process and a summary of the results obtained. This chapter begins with an overview of the framework design, highlighting the core components and their purposes in supporting ambiguity identification.

The framework was developed based on insights from the literature review and analysis of ambiguity issues in requirements elicitation and requirement documentations. Following the design, the framework was validated through expert evaluation using a structured survey. The results of this validation are presented and discussed to assess the framework's effectiveness. usability, and relevance to real-world practices.

4.2 FRAMEWORK DESIGN

This section outlines the design and architecture of the syntactic ambiguity detection framework while discussing the major elements of the proposed framework and how these elements collectively collaborate with each other to detect ambiguity in SRS. The framework was constructed to guide requirements engineers in recognizing and mitigating syntactic ambiguities during the development of software requirements.

4.2.1 Framework Development and Component

The framework comprises a set of interrelated components designed to systematically analyze requirement statements, detect potential ambiguities, and support the refinement of unclear specifications. The aims are to enhance clarity and minimize misinterpretations during subsequent stages of software development.

The design of the framework is grounded in RE literature, particularly ambiguity detection techniques and linguistic guidelines. The inclusion of an ambiguity terms repository and writing guidelines is informed by prior studies highlighting the recurring use of vague or subjective terms in SRS documents. Pattern recognition and expert intuition were incorporated to accommodate both rule-based and experience-driven detection mechanisms. The modular structure of the framework allows for scalable integration into existing RE workflows. Furthermore, by combining rule-based detection with human expert validation, the framework balances automation with domain-specific knowledge. The framework is intended to be used by RE practitioners, particularly requirement analysts and engineers, to improve requirement clarity and reduce misinterpretation during the software development lifecycle. The overall structure of the framework is visualized in Figure 6 and is further described. The framework is divided into two main phases:

1. Requirement Elicitation
2. Requirement Documentation

Each phase comprises components that are supported by examples.

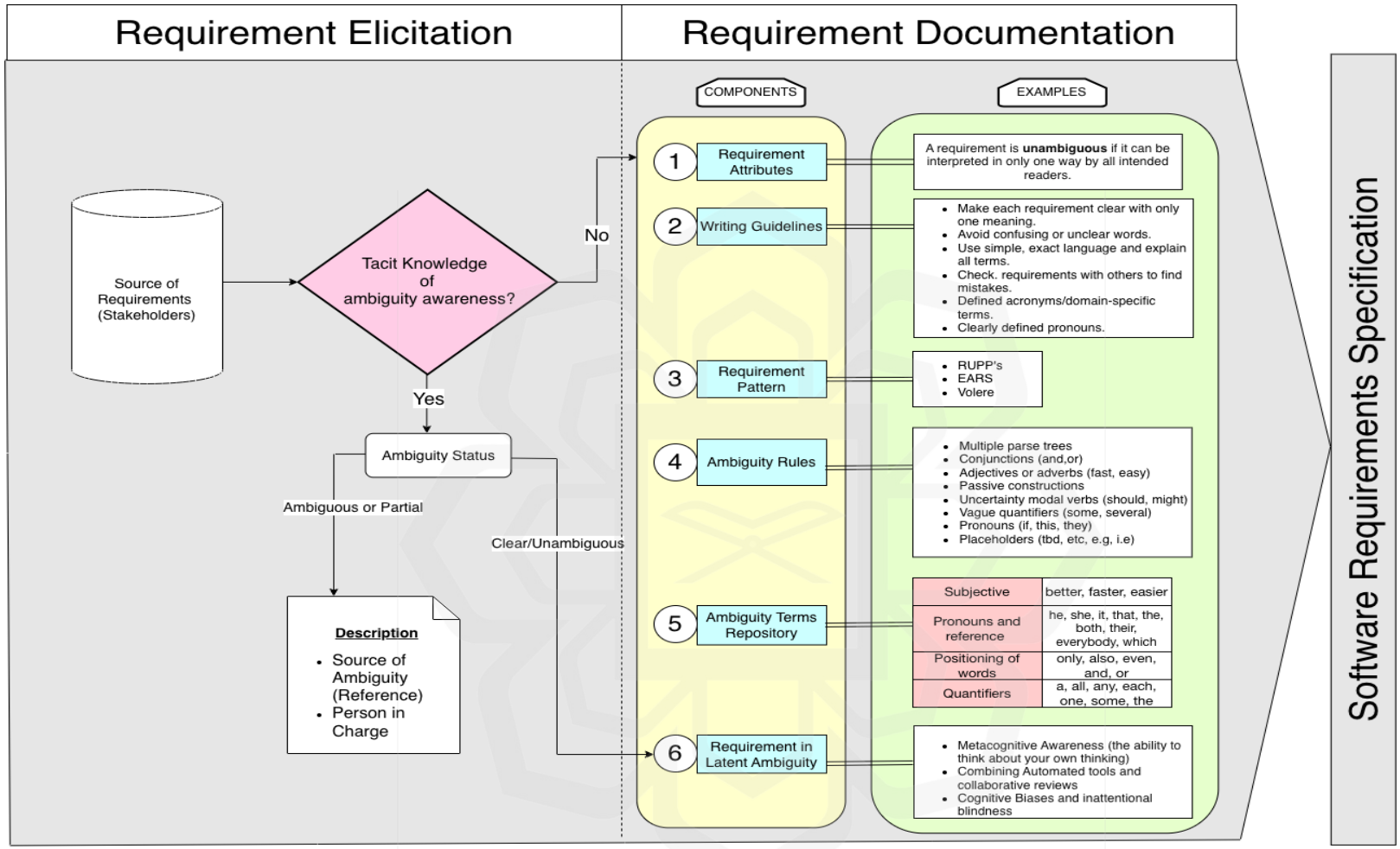


Figure 6: Syntactic Ambiguity Detection Framework in SRS

The framework components are as follows:

1. Requirement attributes: These are qualities or criteria that good requirements should satisfy to ensure clarity and quality. Key attributes typically include clarity, consistency, completeness, and unambiguity. These attributes guide the evaluation and detection of ambiguities by defining what aspects of the requirements should be assessed to determine if ambiguity exists (Kamsties et al., 2001).
2. Writing guidelines: These are best practices and recommended rules for composing SRS in clear, precise language. Writing guidelines help requirement authors avoid common sources of ambiguity by promoting straightforward sentence structures, consistent terminology, and avoidance of vague or complex phrasing. They function as preventive measures to reduce ambiguity at the source (INCOSE,2023 ; Robertson & Robertson ,2010) ; Pohl, 2010).
3. Requirement patterns: These are standardized templates or sentence structures designed to express requirements in a regular, predictable format that minimizes syntactic ambiguity. The template provides syntactic regularity that makes requirements easier to understand and less prone to multiple interpretations (Arora et al., 2003).
4. Ambiguity rules: These are explicit linguistic or syntactic rules derived from expert analysis and linguistic theory that identify common structures, phrases, or terms that tend to induce ambiguity in requirements. Ambiguity rules enable systematic detection by marking sentences or parts of the text that violate clarity such as ambiguous references, vague quantifiers, or complicated sentence constructs (Berry et al., 2001).
5. Ambiguity terms repository: This is a collection of glossaries of words and phrases known to frequently cause ambiguity in SRS. This glossary may include pronouns, vague quantifiers, modal verbs, or commonly overloaded technical terms. The repository supports detection efforts by flagging suspicious terms during both automated and manual reviews (Gleich et al., ; Berry et al., 2003).
6. Requirement in latent ambiguity: This requirement refers to ambiguity that is not immediately obvious within the text itself but revealed through broader context, interpretation, or later stages of development. This component addresses hidden

ambiguities by employing multi-layered strategies such as meta-cognitive review, collaborative assessments and automated tool support to uncover ambiguities that escape surface-level analysis (Dar et el., 2023).

The research flow began with a comprehensive review of existing studies, frameworks, and techniques related to syntactic ambiguity in Software Requirements Specification (SRS). From this review, six elements were consistently identified as critical for detecting and reducing syntactic ambiguity. Unlike studies that generate a large pool of elements before narrowing them down, this study found that the literature converged strongly on six recurring elements. These six were therefore adopted directly into the framework based on relevance, distinctiveness, empirical support, and usability. Table 10 maps each element to its supporting sources and explains the rationale for inclusion. These elements form the foundation of SADF, ensuring that syntactic ambiguities are systematically detected and addressed. Each element was supported by multiple sources in the literature. For example, Sabriye & Zainon (2018) and Gavran et al. (2021) highlighted the prevalence of syntactic ambiguity in natural language requirements; Gupta & Deraman (2019) and Osama & Aref (2018) emphasized rule-based detection; Mavin et al. (2009) and Rupp (2014) demonstrated the value of requirement patterns and boilerplates; while Hussain et al. (2023) and Ribeiro & Berry (2020) stressed the importance of requirement attributes and validation processes. The rationale for adopting these six elements directly is because of:

1. Consistency across studies: Each element appeared repeatedly in multiple independent sources.
2. Distinctiveness: The six elements represent unique constructs without overlap.
3. Practical applicability: Limiting the framework to six elements ensures usability and clarity for practitioners.

Table 10: Six Components of SADF and Supporting Literature

Framework Element	Description	Supporting Sources	Rationale for Inclusion
Requirement Attributes	Core properties of requirements (clarity, completeness, consistency) that influence ambiguity.	Pandey et al. (2010); Hussain et al. (2023); Ribeiro & Berry (2020)	Literature emphasizes that poor attributes (unclear, incomplete) are a root cause of ambiguity in SRS.
Writing Guidelines	Structured rules for phrasing requirements to reduce syntactic ambiguity.	Gavran et al. (2021); Damian et al. (2020); Rupp (2014)	Guidelines ensure requirements are expressed in precise, unambiguous language; boilerplates and controlled NL are widely recommended.
Requirement Pattern	Use of standardized templates (e.g., EARS, Rupp's boilerplate) to enforce clarity.	Mavin et al. (2009, EARS); Rupp (2014); Zhang et al. (2016)	Patterns provide repeatable structures that minimize syntactic variation and ambiguity.
Ambiguity Rules	Linguistic heuristics and detection rules for identifying ambiguous structures.	Gupta & Deraman (2019, SRAAF); Osama & Aref (2018, DARA); Gervasi et al. (2019)	Rule-based approaches are central to ambiguity detection; SADF integrates them systematically.
Ambiguity Term Repository	Glossary of terms prone to ambiguity (e.g., "fast," "user-friendly").	Sabriye & Zainon (2018); Damian et al. (2020)	Repositories help engineers recognize and avoid

Framework Element	Description	Supporting Sources	Rationale for Inclusion
			ambiguous terms during documentation.
Requirement in Latent Ambiguity	Identifying hidden ambiguities not obvious at first reading (e.g., scope, modifiers).	Gervasi et al. (2019); Hussain et al. (2021)	Latent ambiguities often surface later in development; detecting them early improves requirement quality.

The source of requirements component is recorded at the first phase and may originate from any requirements engineering (RE) sources such as use case scenarios, user interviews, documentation, or stakeholder input. These claims form the primary information to be examined. To ensure that the data acquired are comprehensive, thorough, and representative of the intended system functionality, this component facilitates the systematic collection of requirement statements from stakeholders and contextualizes the criteria for ambiguity detection.

At this stage, the tacit knowledge of ambiguity awareness components is crucial. This component incorporates the implicit expertise of experienced RE practitioners and domain specialists, acknowledging the limitations of automated detection. Their perceptions and contextual knowledge are essential in identifying subtle ambiguities that might otherwise remain unnoticed. Implicit knowledge, defined as internalized expertise or intuitive understanding (Busch, 2008), directly influences the ability of stakeholders or analysts to recognize ambiguity. As Elghariani et al. (2019) explain, implicit knowledge plays a critical role in Agile requirements engineering, where much of the decision-making relies on shared understanding and experience rather than explicit articulation. Capturing and integrating this implicit knowledge into requirements documentation enhances clarity and reduces the risk of misinterpretation.

The process then moves toward the ambiguity status classification component. As illustrated in Figure 6, the six elements of the Syntactic Ambiguity Detection Framework (SADF)

are integrated into a defined architecture that guides the process of ambiguity detection. Each arrow in the figure represents a decision pathway or flow of information between components. For example, the arrow labelled “No” leading to the term “Clear/unambiguous” indicates that when a requirement is assessed and found to be free of ambiguity, it does not proceed to further diagnostic checks. The implication of this pathway is that such a requirement already satisfies the defined attributes of clarity and precision and therefore is excluded from categorization under the ambiguity elements. Conversely, if the requirement does not meet this condition, it is directed into the framework’s diagnostic components such as Ambiguity Rules, the Ambiguity Term Repository, or Latent Ambiguity for further analysis.

Each identified requirement is systematically classified into one of two categories to assess its clarity and potential risk. This classification serves as a diagnostic tool that guides refinement activities and ensures the overall quality of the SRS prior to its transition as input into the design and implementation stages. For traceability and accountability, each ambiguous requirement is also documented with its Source of Ambiguity and the Person in Charge (PIC) responsible for its resolution. The classification categories are defined as follows:

1. Unambiguous requirements: These are requirements that possess a clear, singular interpretation by all intended readers. They align with the IEEE Std 830-1998 guideline, which defines a good requirement as one that is unambiguous, verifiable, and interpreted consistently by all stakeholders.
2. Ambiguous or partial requirements: These requirements exhibit incomplete, unclear, or vague descriptions that may lead to multiple interpretations or confusion. They require further clarification, rewording, or stakeholder input. Upon identification, such requirements are immediately assigned to the designated Person in Charge (PIC) for review and refinement. This ensures that ambiguities are resolved at the source and that each requirement evolves toward a clearer specification.

Finally, requirements that appear clear i.e. not ambiguous, undergo an additional check for latent ambiguity. This component is designed to detect any hidden ambiguities that might have

been overlooked through deeper linguistic and contextual analysis, extending beyond surface-level rule checking. Not all ambiguities are immediately visible by the requirement engineers. The framework includes a cognitive reflection that combines meta-cognitive awareness and collaborative reviews, encouraging stakeholders to consciously check their assumptions and blind spots as studied by existing researchers as below:

1. Naujoks-Schober and Dresel (2023) describe metacognitive monitoring as a context-dependent process involving metacognitive strategies and judgments that help learners evaluate and regulate their learning and comprehension. They emphasize that metacognitive monitoring enhances awareness of misunderstandings and supports improved self-regulation during learning tasks.
2. The study by Fantechi et al., (2024) and Femmer et al., (2017) demonstrating that combining automated detection methods (e.g., NLP-based detection) with expert human analysis enhances both recall and precision, thereby improving the overall effectiveness and accuracy of ambiguity identification.
3. The awareness of inattentional blindness and cognitive biases, coupled with metacognitive debiasing techniques, supports more objective and rigorous peer review processes by enabling reviewers to better recognize and counteract their perceptual and judgmental errors (Da Silva, Gupta, & Monzani, 2023; Kahneman, 2011).

Latent ambiguity refers to hidden ambiguities that are not immediately obvious to the author or reviewers but may later result in misinterpretation by developers, testers, or users. These typically arise due to cognitive biases, implicit assumptions, or syntactic complexity. If a requirement is initially categorized as unambiguous or determined to contain latent ambiguity, it is further analyzed and documented under requirement in the latent ambiguity component, where it is cross-checked against ambiguity rules and known ambiguous terms for validation. A latent ambiguity is dangerous, as it may only be detected later during design, implementation, or validation stages. Early detection of ambiguities in software requirements is crucial for preventing costly downstream corrections, as established by Kof (2005) and Gorschek et al. (2006), and further supported by recent studies emphasizing improved cost-efficiency and quality assurance

when ambiguity is identified and addressed during initial development phases (Fantechi, Gnesi, & Semini, 2024). As stated by Fantechi et al., (2024), this final stage is important because even experienced engineers may overlook fine ambiguities that automated tools, or initial reviews might not detect. Through meta-cognitive strategies such as questioning assumptions or reviewing requirements from different perspectives, or consulting peers to reveal any hidden uncertainties that might have been missed initially, requirement engineers increase their chances of catching hidden problems. Furthermore, cognitive biases often experienced when uncertain about ambiguity are mitigated through peer consultation, which encourages collaborative review and helps uncover unclear aspects that a single engineer might miss due to tunnel vision.

For less experienced engineers or situations requiring additional support, the framework offers several components during the documentation phase, emphasizing requirement attributes with unambiguity as a central quality criterion. Unambiguity ensures that each requirement can be interpreted in only one way by all intended readers, eliminating multiple interpretations that could lead to misunderstandings and errors during development.

Key attributes essential to high-quality requirements include clarity, completeness, consistency, and verifiability. These attributes are referenced in the IEEE Standard for Software Requirements Specifications (IEEE Std 830-1998), which defines a good requirement as one that is correct, unambiguous, complete, consistent, ranked for importance or stability, verifiable, modifiable, and traceable. Individual requirements can be analyzed based on specific linguistic features such as vagueness, passive voice usage, optionality, or imprecision, which are often associated with syntactic ambiguity. Emphasizing such attributes facilitates clear and effective communication among all stakeholders, enabling developers to implement features accurately on the first attempt, thus reducing rework, saving time, and lowering costs. Additionally, well-defined requirements simplify verification and validation processes, ensuring the final product aligns with intended needs and maintains high quality.

Renowned experts like Karl Wiegers reinforce the importance of clear and precise requirements in his book Software Requirements (Wiegers, 2013). Similarly, standards such as IEEE Std 830-1998 advocate clarity, correctness, and unambiguity as essential qualities for good requirements. Prioritizing these attributes helps achieve better project outcomes by fostering a shared understanding of deliverables among all involved parties.

Clear and unambiguous requirements are critical in engineering to ensure accurate implementation and avoid misinterpretation. To achieve this, engineers must adhere to structured writing guidelines which have been integrated into the framework. These guidelines are based on controlled language principles and established best practices, providing recommendation structures for rewriting ambiguous requirements into clear and concise statements. They also serve as educational material for requirements authors. These include best practices such as using simple language, avoiding vague terms, defining all acronyms, and using clearly defined pronouns. These recommendations are supported by several authors:

1. Karl Wiegers (2013) and Kof (2005) emphasizes the need to avoid vague language.
2. Mich et al. (2004) propose guidelines to minimize ambiguity in NL-based SRS.
3. Femmer et al. (2017) found that syntactic patterns improve the quality of requirements and reduce misinterpretation.

Writing guidelines help requirement engineers to construct requirements clearly and avoid confusing or vague language. First, each requirement should be formulated to convey only one meaning, eliminating ambiguity. For example, instead of stating, “The system should process data quickly,” a precise requirement such as “The system shall process 1000 transactions per second” provides measurable criteria, aligning with the IEEE 29148-2018 standard’s emphasis on unambiguous and testable requirements (IEEE,2018). A common pitfall in requirement specification is the use of confusing or subjective terms such as “user-friendly” or “efficient” which lack quantifiable benchmarks. Replacing these with observable metrics ensures clarity. For instance, “User shall complete the login process in less than 2 clicks” removes subjectivity, a practice supported by Wiegers and Beatty (2013), who caution against vague language in defining

expectations. Additionally, simple and exact language is critical. Technical terms, acronyms, or domain specific jargon must be explicitly defined. A requirement like "The Unmanned Aerial Vehicle (UAV) shall maintain a flight altitude of 50 meters" demonstrates this principle, ensuring all stakeholders share a common understanding, as advised by ISO/IEC/IEEE 15288:2023 (ISO/IEC/IEEE,2023).

Collaborative validation is equally important. Reviewing requirements with peers or stakeholders helps identify ambiguities or gaps early. For example, a software team might discover a missing edge case during a review session, reducing the risk of costly revisions later. This aligns with the INCOSE System Engineering Handbook's recommendation for iterative validation to mitigate misinterpretation risks (INCOSE, 2023). Furthermore, defining acronyms and domain specific terms such as specifying "Programmable Logic Controller (PLC)" on first use ensures clarity across multidisciplinary teams, a requirement underscored by IEEE 29148-2018 (IEEE, 2018). Finally, pronouns must have explicit antecedents to avoid confusion. For example revising "The sensor shall transmit data, and it must be encrypted" to "The sensor shall transmit data, and the data must be encrypted" eliminates ambiguity, adhering to technical writing best practices outlined in the Microsoft Manual of Style (Microsoft Corporation, 2012). Adhering to these guidelines reduces ambiguity, enhances verifiability and fosters collaboration.

Structured requirement patterns such as RUPP's templates, the EARS templates and Volere Requirements Specification Template, provide systematic methods for formulating requirements with precision and consistency. These patterns reduce ambiguity by standardizing the expression of functional and non-functional requirements. and enforce understanding. For instance, EARS employs conditional logic to standardize requirement expression through patterns like "When [trigger condition], the system shall [action]" or "If [precondition], then [outcome]" (Mavin, 2010). This syntax explicitly links triggers, preconditions, and outcomes, eliminating vagueness by mandating a clear cause-effect relationship. Similarly, RUPP's templates often associated with boilerplate language use structured phrasing such as "The system shall [action] under [condition]"

to ensure consistency in clause ordering, which simplifies interpretation for developers and testers (Rupp, 2023).

The Volere template, while broader in scope, complements these syntactic patterns by providing a comprehensive organizational framework. The template standardizes requirements documentation across functional, non-functional, and contextual dimensions embedding boilerplate-like atomic requirements such as “The system shall validate user credentials within 2 seconds” within a traceable structure (Robertson & Robertson, 2012: Volere, 2016). The primary advantage of these patterns lies in their ability to enforce uniform syntax or structure across requirements. For example, a requirement like “When the user clicks ‘Submit’, the system shall save the form data and display a confirmation message” (Mavin, 2010) adheres to EARS event-response structure, making it easier to validate and test. Volere extends this correctness by integrating measurable acceptance criteria (e.g., “95% of users shall complete onboarding within 5 minutes”) and traceability matrices, ensuring compliance with standards like ISO/IEC/IEEE 15288 (INCOSE, 2023). Structured requirement templates have been shown to significantly reduce defects and misinterpretations in software requirement specifications.

By providing a semi-structured format, these templates help minimize common issues such as ambiguity, vagueness, and inconsistency, which are prevalent in free-form natural language specifications. This structured approach enhances clarity and facilitates early error detection, thereby improving overall software quality and reducing rework costs. (Vallejo et al., 2020; Boehm & Basili, 2001). Research on adapted requirements engineering frameworks in Chinese and other languages demonstrates the cross-lingual applicability and effectiveness of structured pattern-based approaches, such as those resembling EARS syntax patterns (Großer et al., 2024). Volere’s flexibility allows it to integrate domain-specific terminology, as seen in healthcare IT regulatory standards like HIPAA (INCOSE, 2023). Tools leveraging structured patterns, including automated 'smell detectors,' help improve requirement quality by identifying poorly phrased or ambiguous statements early in development (Pereira dos Reis et al., 2022; Mourya & Singh, 2025). For example, a requirement like “The PLC shall activate

the emergency shutdown protocol if pressure exceeds 50 psi" not only follows RUPP's boilerplate structure but also aligns with domain-specific terminology and ensures clarity.

In conclusion, structured requirement patterns like EARS, RUPP's templates and the Volere transform vague natural language into precise, verifiable specifications. By enforcing syntactic strictness, conditional logic, and organizational traceability they align with global standards such as INCOSE guidelines and ISO/IEC/IEEE 15288, making them indispensable for modern requirements engineering (INCOSE, 2023; ISO/IEC/IEEE, 2023). Their adoption fosters actionable, testable requirements that streamline project execution and mitigate risks in complex engineering systems.

The ambiguity rules provide explicit examples of what to avoid, acting as practical checklists during requirement writing. The ambiguity rules encompass a predefined set of syntactic patterns that are known to introduce confusion or allow for multiple interpretations. These rules serve as formal criteria against which each requirement is evaluated. By targeting linguistic structures prone to multiple interpretations, these rules flag problematic phrases and enforce clarity. For instance, multiple parse trees (e.g., "monitor sensors and alarms using AI") are detected when modifiers or conjunctions create syntactic ambiguity, prompting restructuring to eliminate dual meanings (IEEE, 2018). Similarly, conjunctions (and/or) are examined to avoid combined clauses, such as revising "view and edit profiles and settings" into discrete actions using bulleted lists (Wiegers & Beatty, 2013). Adjectives/adverbs like "fast" or "easily" are replaced with quantifiable metrics (e.g., "within 2 seconds") to align with testability criteria in ISO/IEC/IEEE 15288 (ISO/IEC/IEEE, 2023).

Passive constructions, uncertainty modals (should/might), and vague quantifiers (some/several) are systematically revised to active voice, mandated outcomes (shall), and precise thresholds, respectively, ensuring accountability and measurability (INCOSE, 2022; Microsoft, 2020). Ambiguous pronouns (it/they) are resolved by restating antecedents, while placeholders (tbd/etc.) are replaced with explicit terms to eliminate deferred decisions. Research evidence

indicates that adherence to these rules improves requirement clarity and reduces defects and misinterpretations, transforming subjective statements into actionable, verifiable specifications (Vallejo, Mazo & Jaramillo, 2020: Boehm & Basili, 2001). By applying these rules, engineers ensure requirements are universally interpretable, aligning stakeholder understanding and complying with standards such as ISO/IEC/IEEE 29148:2018 (IEEE, 2018).

Finally, a repository of ambiguous terms serves as a vocabulary reference, helping requirement engineers identify and substitute unclear words or phrases. This is a curated database containing frequently ambiguous terms and phrases. When such terms appear in requirement statements, the framework begins to examine them more closely. Based on Kamsties, E., & Paech, B. (2002), this repository catalogs ambiguous terms, categorized into:

1. Subjective words: These are evaluative or comparative terms whose meaning depends on individual perception or context, making them inherently vague. For example, words like better, easier, fast, or user-friendly lack measurable criteria and can lead to different interpretations by stakeholders.
2. Pronouns and references: Pronouns or vague references (such as this, that, it, they) often cause confusion because they rely heavily on context or prior text to clarify their referent. Ambiguity arises when it is unclear to which entity the pronoun refers, especially in complex sentences or large documents.
3. Positional and logical connectives: Connectives like only, and, or, if, unless can introduce ambiguity when their logical scope or exclusive/inclusive meanings are not explicitly defined. Misinterpretation of these terms can lead to different understandings of requirement conditions or constraints.
4. Quantifiers: Quantifiers express amounts or frequencies but often lack precision in requirements. Words like all, some, few, several, or often are inherently vague because they do not specify exact numbers or thresholds, leading stakeholders to assume different interpretations.

These terms are known to introduce ambiguity, as discussed by Gervasi & Zowghi (2005) in their analysis of misunderstandings in requirements due to vague language and Kamsties, Berry & Paech (2001) who classified ambiguous words used in industrial SRS. Ambiguity in technical writing, particularly within SRS, often arises from the use of imprecise language, including pronouns, subjective terms, positioning words, and quantifiers. Pronouns such as 'he', 'she', 'it', 'that', 'their', and 'both' can create unclear references when the antecedent is not explicit, leading to confusion about which entity is being addressed (Microsoft Corporation, 2012). For instance, ambiguity occurs when a sentence like "The sensor detects a fault, and it must report it" leaves readers uncertain whether "it" refers to the sensor or the fault. Subjective terms such as 'better', 'faster', and 'easier' also contribute to ambiguity because they rely on personal interpretation and lack measurable criteria (Wiegers & Beatty, 2013). For example, the requirement "The system shall process requests faster" is insufficiently precise without defining specific time constraints or benchmarks. The positioning of words such as 'only,' 'also', 'even', and connectors such as 'and', and 'or' can alter the meaning of sentences depending on their placement, further complicating clarity (Robertson & Robertson, 2012). A statement such as "Only users with admin rights can access reports" may be interpreted in multiple ways, creating scope uncertainty.

Lastly, quantifiers including 'a', 'all', 'any', 'each', 'one', 'some', and 'the' can introduce ambiguity by failing to define scope or quantity definitively (Hull, Jackson, & Dick, 2011). For example, stating "The system must handle multiple requests" does not specify the exact number of requests the system should support. The term "multiple" is vague because it does not specify an exact number or range. Different stakeholders may interpret "multiple" differently and some may think it means 2 or more, others 10, or even hundreds leading to inconsistent understanding and potential implementation errors.

To mitigate these issues, technical writers are encouraged to replace vague pronouns with precise nouns, quantify subjective terms with measurable criteria, clearly position qualifying words, and specify exact quantities or scopes, thereby improving the ambiguity and effectiveness of requirements documentation (Microsoft Corporation, 2012; Wiegers & Beatty, 2013). After

passing through these components and the latent ambiguity check, requirements can be safely documented into the final SRS.

In summary, this syntactic ambiguity detection framework equips requirements engineers with a systematic method to identify, analyze and resolve ambiguities in software requirements. By combining tacit knowledge during elicitation, meta-cognitive strategies, collaborative peer review, and structured writing supports, the framework contributes to the production of higher quality requirements. Early ambiguity detection reduces costly rework, minimizes misunderstandings, and ultimately improves the success and quality of software development projects. By applying this classification framework, requirement engineers and analysts can proactively manage linguistic uncertainties in the SRS. This structured approach not only facilitates better communication and understanding across stakeholders but also mitigates the risk of error propagation into subsequent phases of the software development lifecycle. The proposed Syntactic Ambiguity Detection Framework (SADF) integrates structured requirement analysis, linguistic tools, and cognitive strategies to identify ambiguity in natural-language SRS. By combining syntactic rules, requirement patterns, repositories, and human judgment, the framework aims to improve the reliability, consistency, and clarity of software requirements.

4.3 FRAMEWORK IMPLEMENTATION AND VALIDATION

The proposed framework for detecting syntactic ambiguity in SRS was implemented with the objective of providing a structured and systematic, practical guideline to support requirements engineers in identifying and resolving ambiguous expressions. This implementation phase focused on operationalizing the core components of the framework, including requirement quality attributes, structured writing guidelines, requirement pattern or boilerplates, a comprehensive set of ambiguity detection rules, ambiguity glossary and requirement in latent ambiguity. A guideline on how to use the framework has been included in the Appendix 2.

This phase involved formalizing the framework elements into actionable procedures and reference materials that could be directly applied during the requirements elicitation and documentation processes. The framework was designed to guide users through common syntactic ambiguity problems, such as unclear pronoun references, ambiguous quantifiers, vague subjective terms and problematic word positioning by providing explicit definitions. examples and suggested corrections. Writing guidelines were developed to promote unambiguous and precise language, aligning with established best practices in technical writing. This structured approach aimed to reduce reliance on subjective interpretation and improve consistency across requirements documentation. To facilitate practical usage, the framework was packaged as a reference toolkit, including checklists and ambiguity term repositories enabling users to cross-check requirement statements during drafting and review phases. Though primarily manual, this implementation laid the foundation for potential future integration with automated tools, which could further streamline the ambiguity identification process.

Following the implementation, the validation phase was critical to evaluate the framework's effectiveness, usability and relevance in real-world settings. A quantitative approach was employed by using an online structured survey targeted at domain experts. The survey instrument was carefully developed to measure key evaluation dimensions such as clarity of the framework's structure, coverage of the ambiguity issues, relevance of components to practical ambiguity detection, ease of use during requirements engineering activities, and potential impact on improving ambiguity identification and reducing review effort. The survey consists of two sections. The first section gathers demographic and participant background information to contextualize responses, while the second comprises Likert-scale and open-ended questions aimed at capturing detailed expert opinions and suggestions. Likert-scale questions were scored on a five-point scale ranging from Strongly Disagree (1) to Strongly Agree (5), allowing quantitative analysis of expert agreement with statements related to the framework's attributes.

Experts were selected through purposive sampling to ensure inclusion of participants with demonstrated expertise in requirements engineering and software development. The panel

comprised academics, requirement engineers, business analyst, quality assurance or testers professionals from both industry and academia, including representatives from the International Islamic University Malaysia and various companies. The diverse expertise and experience of the panel ranging from novice to senior professionals ensured a comprehensive evaluation from multiple perspectives and enhanced the validity of findings. The validation process provided critical insights into the strengths and areas for improvement of the framework. Quantitative results revealed high levels of agreement on the clarity, relevance and practically of the framework, indicating its potential to significantly aid ambiguity detection is SRS documentation. Expert feedback highlighted the need for expanding the component to integrate with automation tools to increase efficiency and recommendation to add a few components that seem like related to the framework. Overall, the successful implementation and validation of the framework emphasize its applicability and value in supporting clearer, more precise software requirements documentation.

CHAPTER FIVE

RESULT, ANALYSIS AND DISCUSSION

5.1 INTRODUCTION

This chapter presents the key findings obtained from the research activities conducted throughout the study. The chapter includes the development, validation, and evaluation of the proposed syntactic ambiguity detection framework for SRS. The results are derived from the analysis of expert surveys, and the integration of theoretical principles from existing literature. The chapter is organized to first outline the obtained results, followed by a detailed analysis and discussion that connects these findings to the research objectives and questions. Additionally, the chapter explores how the identified framework elements help to address ambiguity in SRS, providing a comprehensive answer to the research questions.

5.2 DEMOGRAPHIC DATA

Data was collected via an online structured survey targeted at selected RE experts. For the quantitative analysis, descriptive statistics were used to summarize participants' responses to Likert-scale items. Measures such as frequencies and percentages were calculated to assess the perceived clarity, usefulness, and comprehensiveness of each framework component. These metrics provided an overview of how each part of the framework was evaluated by the respondents.

The analysis focused on open-ended responses, which were thematically analyzed to extract common patterns, concerns, and suggestions. Participant comments were grouped according to themes related to framework usability, clarity, applicability in practice, and perceived limitations. These insights were essential in understanding the rationale behind the quantitative ratings and in capturing expert perspectives that may not be fully reflected in numerical data. A total of 25 experts participated in the validation survey. As shown in Figure 7, their roles included

Business/System Analyst (32%), Software Engineers (20%), Requirement Engineers (16%), QA/Test Engineers (16%), Lecturers/Educators (4%), and others such as web developers, programmers, and IT support staff (12%).

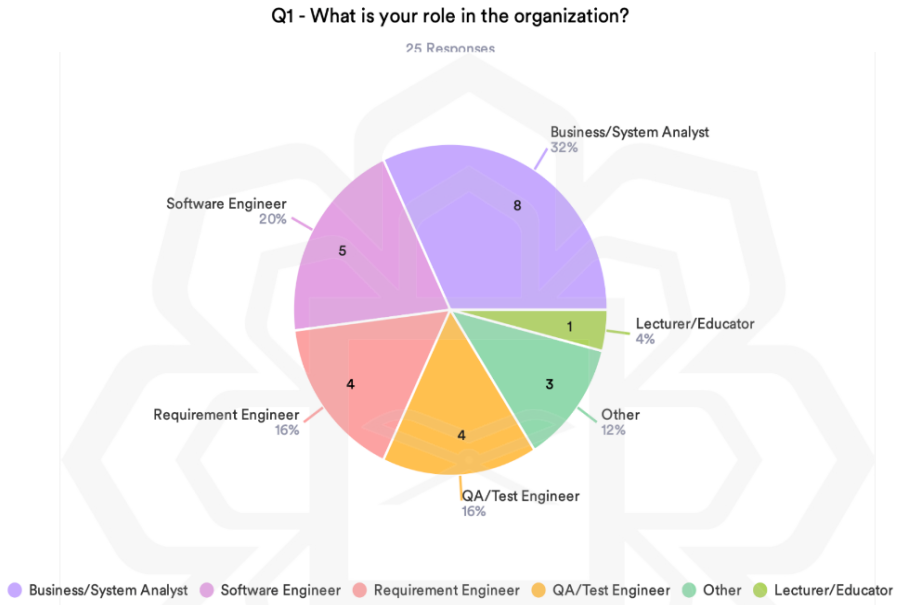


Figure 7: Participant Roles

Most participants possessed formal training in RE, with approximately 84% reported completion of university courses, professional workshops, certifications, or practical experience in the field based on pie charts in Figure 8.

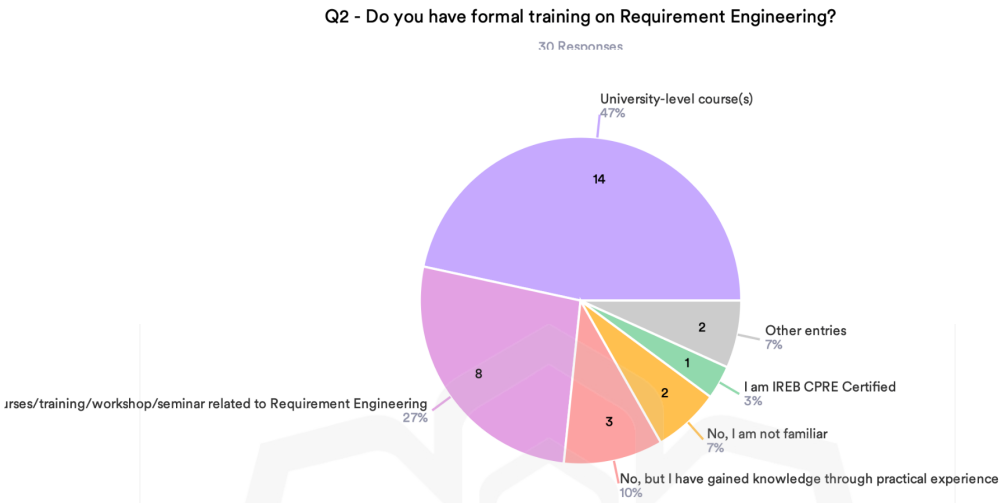


Figure 8: RE backgrounds

Participants were also familiar with various requirements engineering standards and patterns, including EARS (28%), IEEE 830-1998/ISO 29148 (26%), RUPP's (19%), User Story Templates (16%), Volere Requirement Shell (7%), and Planguage Requirement Shell (5%) as depicted in Figure 9.

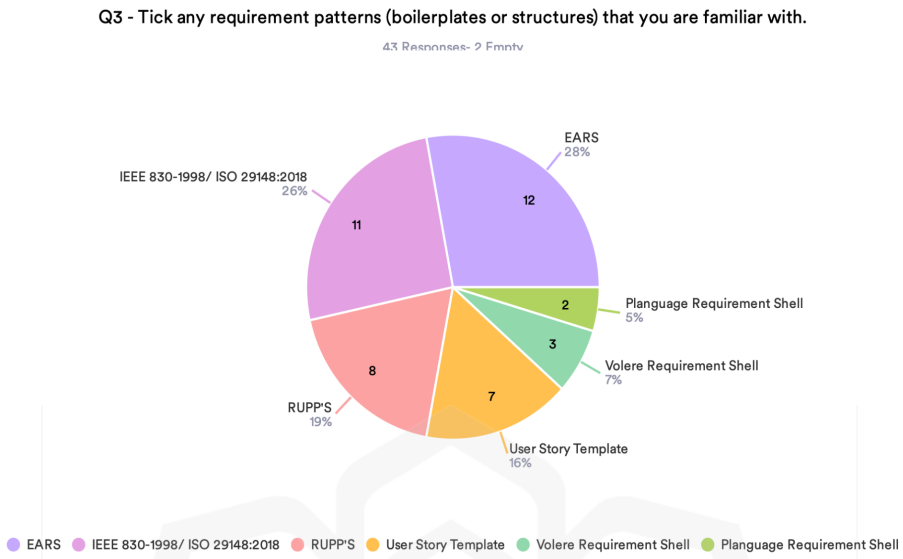


Figure 9: Awareness of Existing Requirement Pattern

As shown in Figure 10, majority of participants were experienced requirement engineers with 64% having more than six years of professional practice and 20% between four and six years of experience, ensuring that the feedback was informed by considerable domain expertise.

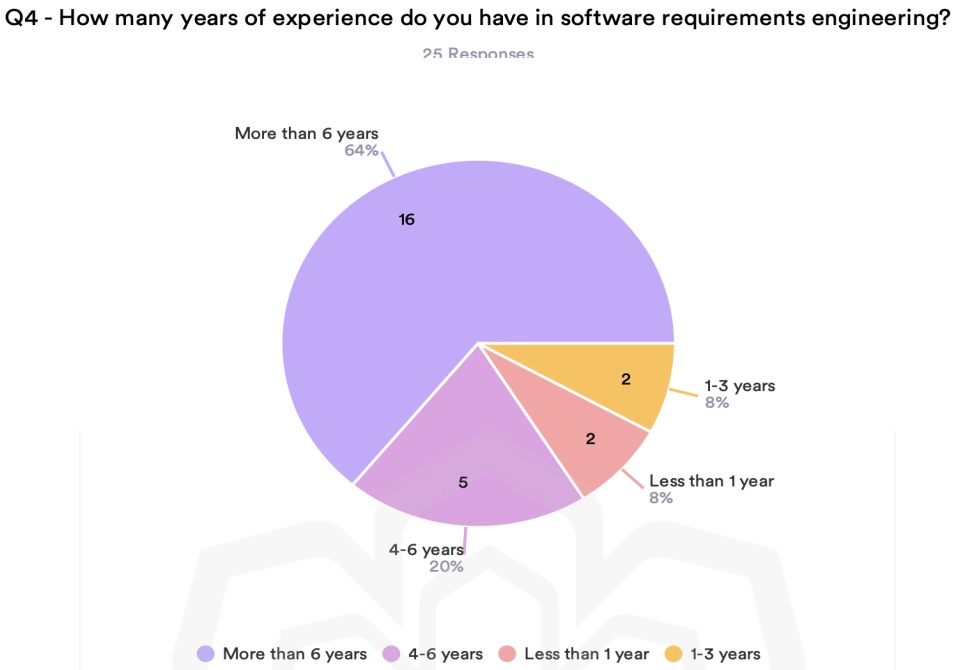


Figure 10: Years of Experience

5.3 ANALYSIS OF RESULTS

The quantitative analysis of the validation survey based on Figure 11 until Figure 20 shows a strong agreement regarding the relevancy, efficiency and practical applicability of the proposed framework for detecting syntactic ambiguity in SRS. While Figure 21 until Figure 24 shows the feedback result from RE experts.

As depicted in Figure 11, 88% of respondents strongly agreed that the framework is clear and understandable, while an additional 8% agreed, indicating that the framework effectively communicates its core concepts. This finding is strongly supported by recent studies that emphasize clarity and understandability as essential qualities in requirements frameworks. Singh,

Patidar, Singh, and Rocha (2025) conducted a comprehensive empirical study on schema understandability and concluded that frameworks incorporating structured metrics and intuitive design significantly enhance user comprehension. Similarly, Chazette, Brunotte, and Speith (2022) highlighted that explainability and transparency are increasingly recognized as critical non-functional requirements in software systems, directly influencing stakeholder trust and usability. These studies reinforce the importance of designing frameworks that are not only technically comprehensive but also cognitively accessible and understandable to practitioners

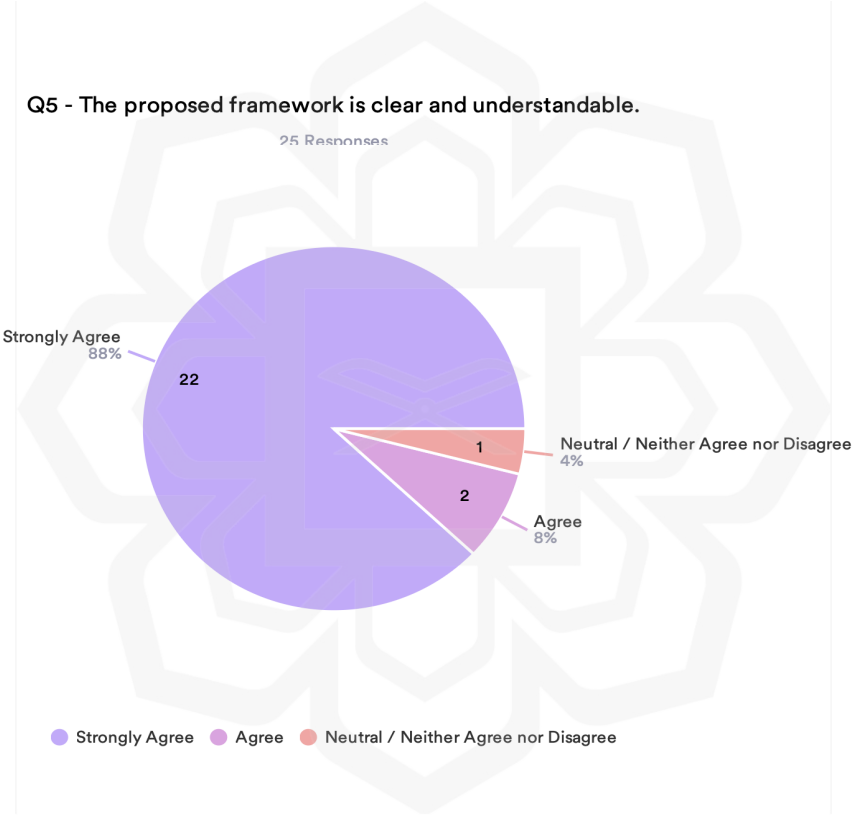


Figure 11: Clarity and Understandable

As depicted in Figure 12, a large majority of respondents where 84% of them are strongly agreed that the components of the framework are relevant to detecting functional requirement ambiguities.

This strong agreement indicates that the framework effectively addresses a key challenge in software and systems development due to the presence of unclear or imprecise functional requirements, which can lead to misunderstandings, defects, or project delays.

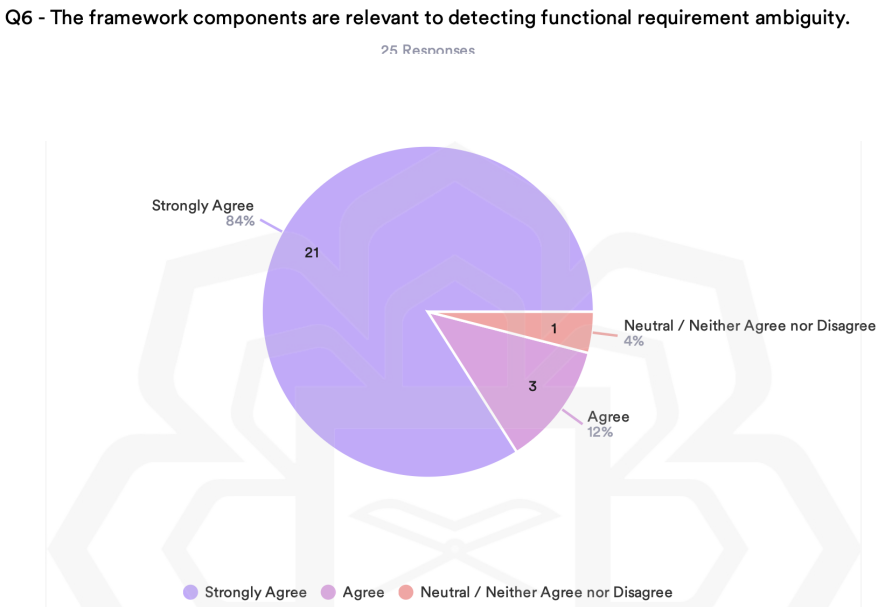


Figure 12: The Relevancy of Framework

This finding confirms that each component such as requirement attributes, writing guidelines, requirement patterns, ambiguity rules, and the ambiguity term repository contributes meaningfully to identifying and reducing ambiguity in functional requirements. Prior studies strongly support this result. For example, Sabriye and Wan Zainon (2017) demonstrated that linguistic approaches, including part-of-speech tagging and rule-based detection, are effective in identifying ambiguous functional requirements. Similarly, Gervasi and Ferrari (2019) emphasized the importance of integrating multiple techniques into a unified framework to comprehensively

address ambiguity in functional specifications. These studies reinforce the conclusion that the proposed framework's components are not only theoretically sound but also practically relevant for detecting functional requirement ambiguity across diverse contexts.

As illustrated in Figure 13, 72% of respondents strongly agreed that the framework adequately covers syntactic ambiguity types commonly encountered in real-world settings. This indicates a strong level of confidence in the framework's ability to identify and address ambiguities related to sentence structure, grammar, and phrasing issues that often lead to misinterpretation of functional requirements. The high agreement suggests that the framework includes relevant syntactic categories and detection methods that align with actual challenges faced by practitioners in the field. This reflects the integration of components such as ambiguity rules and the ambiguity term repository within the Figure 6, ensuring comprehensive coverage. These components ensure that the framework encompasses the syntactic categories most frequently reported by practitioners as sources of ambiguity. The finding is consistent with Gupta and Deraman's (2019) Software Requirement Ambiguity Avoidance Framework (SRAAF), which emphasized broad coverage but lacked empirical validation. By contrast, the SADF demonstrates both coverage and validation.

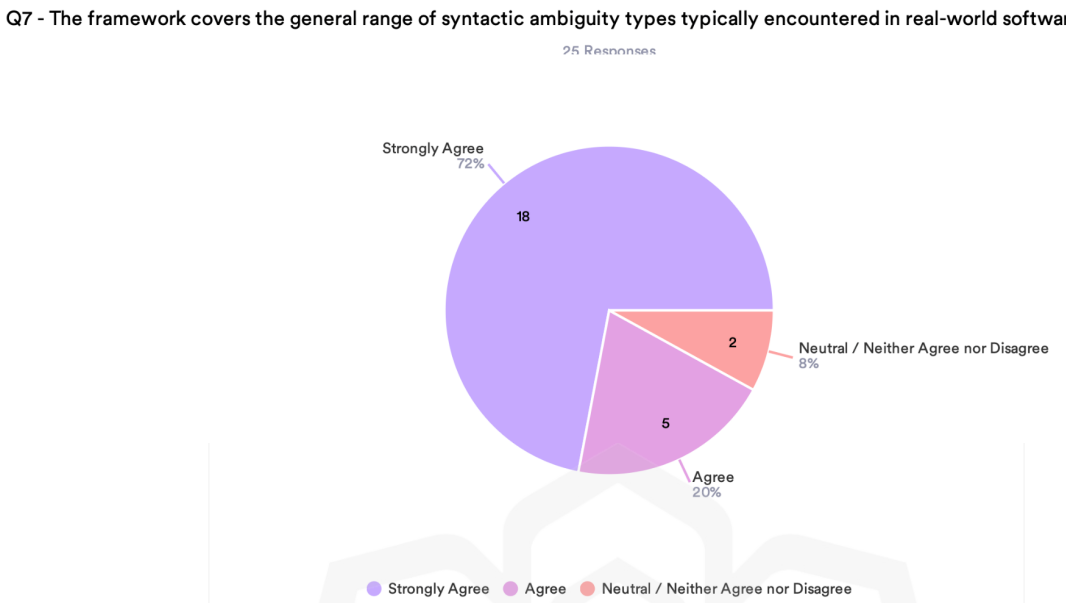


Figure 13: The Framework Coverage

In terms of usability, Figure 14 shows that 76% of respondents strongly agreed that the framework could be practically applied during the elicitation and documentation phases of requirements engineering. This finding demonstrates that the framework is not only theoretically robust but also functionally usable in real-world contexts. Its usability is achieved through components such as Requirement Attributes, which provide structured criteria for precise documentation; Writing Guidelines, which support practitioners in drafting clear and consistent requirements; and Requirement Patterns, which offer reusable templates that streamline elicitation activities. These elements collectively reduce ambiguity and enhance efficiency during early requirements processes. Prior studies support this conclusion. Pohl (2010) emphasized that requirements engineering frameworks must balance documentation and elicitation activities to ensure usability and stakeholder comprehension. More recently, Ayyaz and Ramzan (2024) highlighted that innovative elicitation techniques and structured frameworks improve practical application by reducing errors and ensuring that requirements are captured in ways that are both

understandable and actionable. Together, these studies reinforce the conclusion that the SADF is both easy to understand and practically applicable during elicitation and documentation.

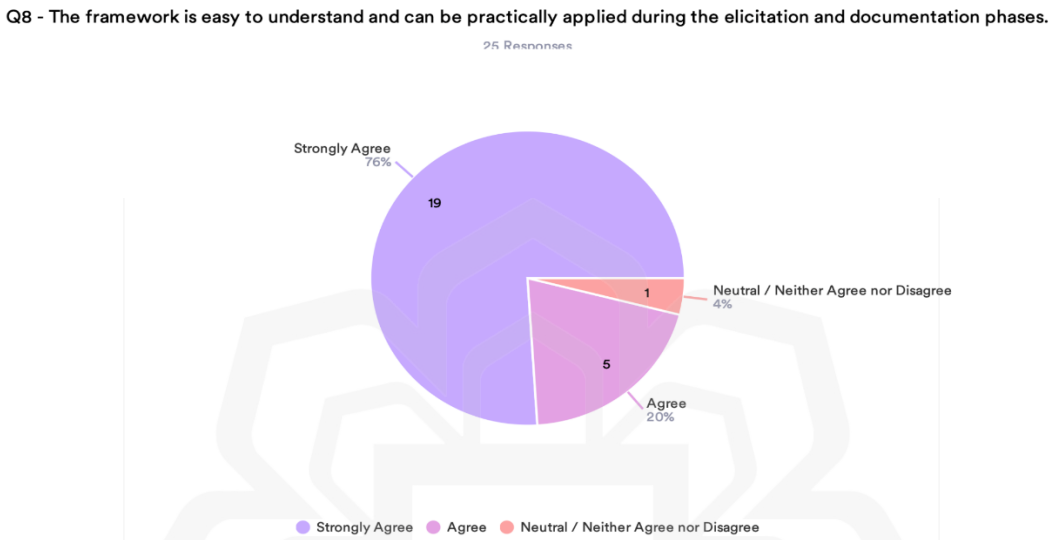


Figure 14: Understandable and Practical

Similarly, Figure 15 shows that 76% of respondents recognized the framework's potential to reduce misinterpretation risks. This reflects strong confidence in its ability to clarify ambiguous requirements and minimize misunderstandings among stakeholders during the development process. The reduction of risk is achieved through components such as Ambiguity Rules, which systematically flag problematic constructions, and the Ambiguity Term Repository, which standardizes terminology to eliminate vague or inconsistent expressions. Together, these criteria provide a structured mechanism for detecting and resolving ambiguity before requirements are finalized. Prior studies support this conclusion. Sabriye and Wan Zainon (2017) demonstrated that rule-based linguistic approaches, such as part-of-speech tagging, are effective in identifying

ambiguous terms, thereby reducing the likelihood of misinterpretation. Likewise, Gervasi, Ferrari, and Tolomei (2019) emphasized that ambiguity detection frameworks must integrate multiple criteria to comprehensively address risks of misunderstanding in functional requirements. These findings reinforce the conclusion that the SADF's ambiguity detection criteria are not only theoretically robust but also practically capable of reducing misinterpretation risks in real-world software projects.

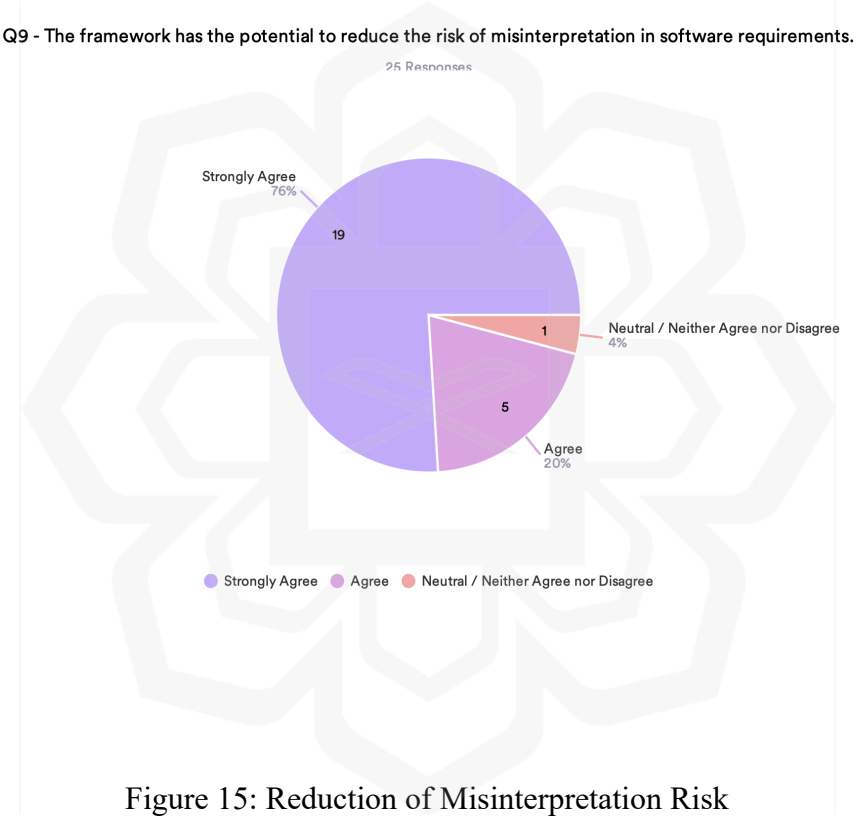


Figure 15: Reduction of Misinterpretation Risk

As shown in Figure 16, 88% of respondents strongly agreed that framework comprises relevant components for identifying and resolving ambiguous terms. This finding highlights a high level of confidence among participants regarding the framework's ability to address one of the most critical issues in communication and interpretation which is ambiguity. This finding validates

the integration of Requirement Attributes, Writing Guidelines, Requirement Patterns, Ambiguity Rules, Ambiguity Term Repository, and Latent Ambiguity within Figure 6. The result aligns with Gervasi et al. (2019), who emphasized the need for frameworks that unify multiple techniques, and with Sabriye and Wan Zainon (2017), who demonstrated the relevance of linguistic approaches.

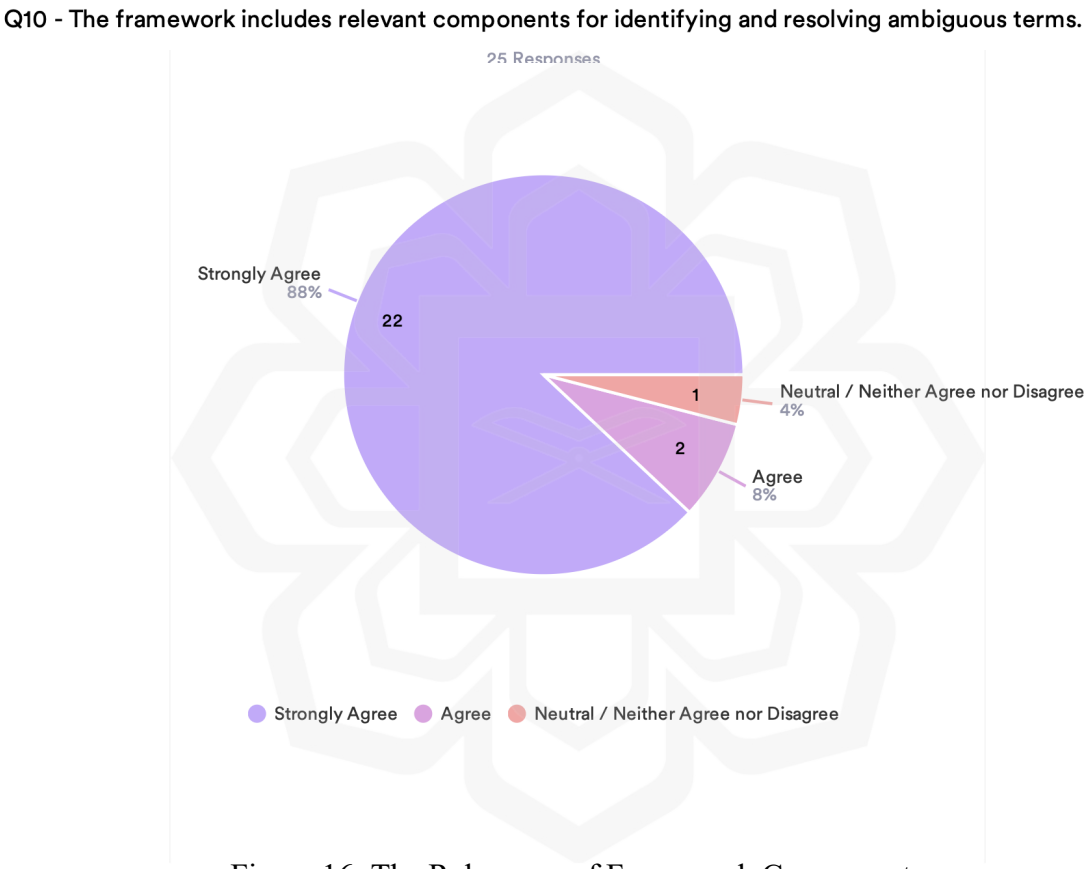


Figure 16: The Relevancy of Framework Components

Furthermore, as presented in Figure 17, 84% of respondents strongly agreed that the framework is effective in identifying ambiguous elements that are often overlooked in manual reviews. This strong agreement suggests that the framework significantly enhances the accuracy of ambiguity detection compared to traditional, manual evaluation method.

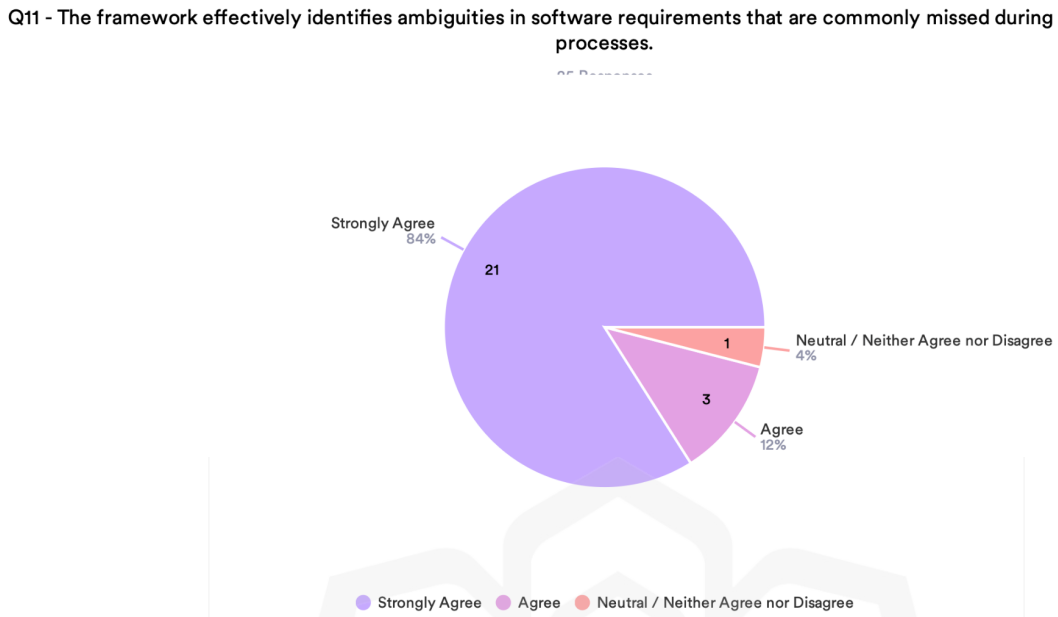


Figure 17: Ambiguity Detection VS Manual Review

Prior studies support this comparison: Sabriye and Wan Zainon (2017) highlighted that manual reviews often fail to detect syntactic ambiguities due to reviewer fatigue and inconsistency, while Gervasi, Ferrari, and Tolomei (2019) demonstrated that automated or structured approaches achieve higher detection rates than purely manual techniques. These findings provide a scholarly basis for stating that the SADF offers improved accuracy relative to manual reviews.

Figure 18 illustrates respondents' perceptions regarding the practical usability of the framework specifically whether it provides sufficient examples or guidance to support real-world application. According to the results, 76% of participant agreed, 20% strongly agreed, and 4% remained neutral.

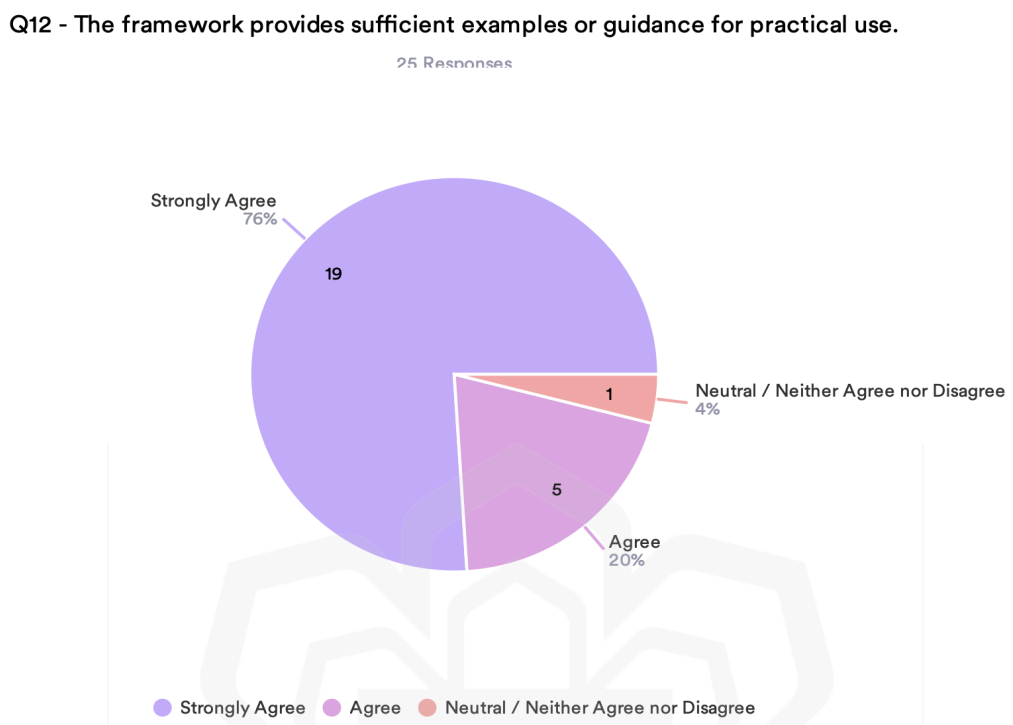


Figure 18: Sufficiency of Examples and Guidelines

These findings suggest that a combined 96% of respondents found the framework to offer at least adequate support for practical use, either through examples, guidelines, or instructional components. This high level of agreement reinforces the framework's accessibility and applicability for end users, especially those who may not be experts in ambiguity detection or formal evaluation processes.

As depicted in Figure 19, when compared to existing ambiguity detection methods, 88% of respondents strongly agreed that the framework offers clear improvements over current ambiguity detection practices, such as manual reviews or other existing techniques. This strong

feedback highlights the framework's perceived added value and advancement compared to conventional methods.

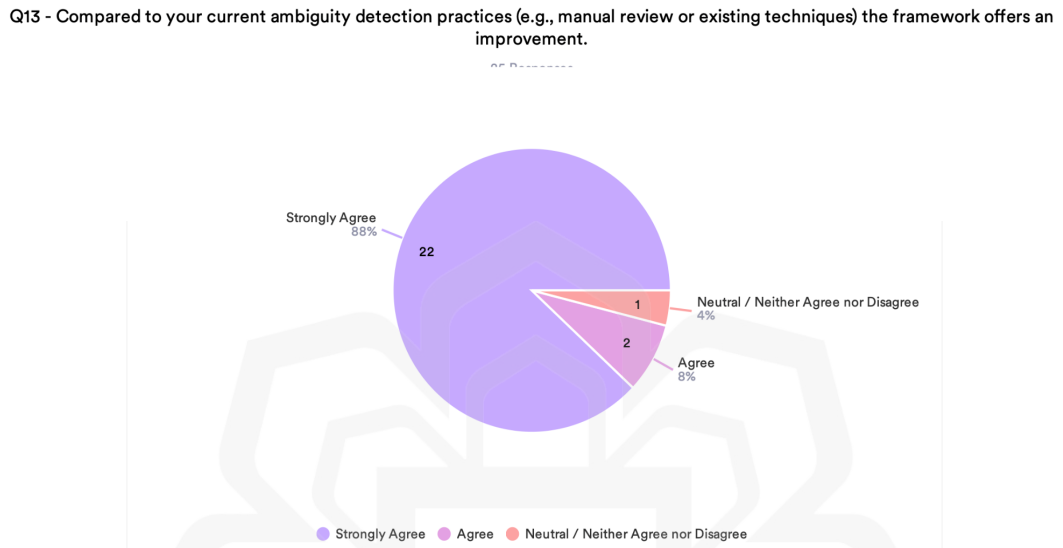


Figure 19: Current Practices vs Framework

Figure 20 illustrates respondents' views on whether the framework helps reduce the time and effort required in writing or reviewing requirements. According to the results, 84% of participants acknowledged this benefit. This indicates a strong perception that the framework streamlines tasks that are often time-consuming and cognitively demanding, especially in contexts where clarity and precision are critical. Traditional requirement reviews often involve repetitive, manual checks to identify ambiguous terms or unclear phrasing. The framework appears to simplify this process by offering structured guidance, predefined criteria, or automated checks that help users work more efficiently.

The implication is significant: if the framework reduces both effort and review duration without compromising quality, it can lead to higher productivity in requirements engineering, faster turnaround times in documentation cycles, and lower cognitive load for reviewers and authors. This benefit further reinforces the framework's practical value and supports its potential for widespread adoption, especially in environments with time constraints or high document volume.

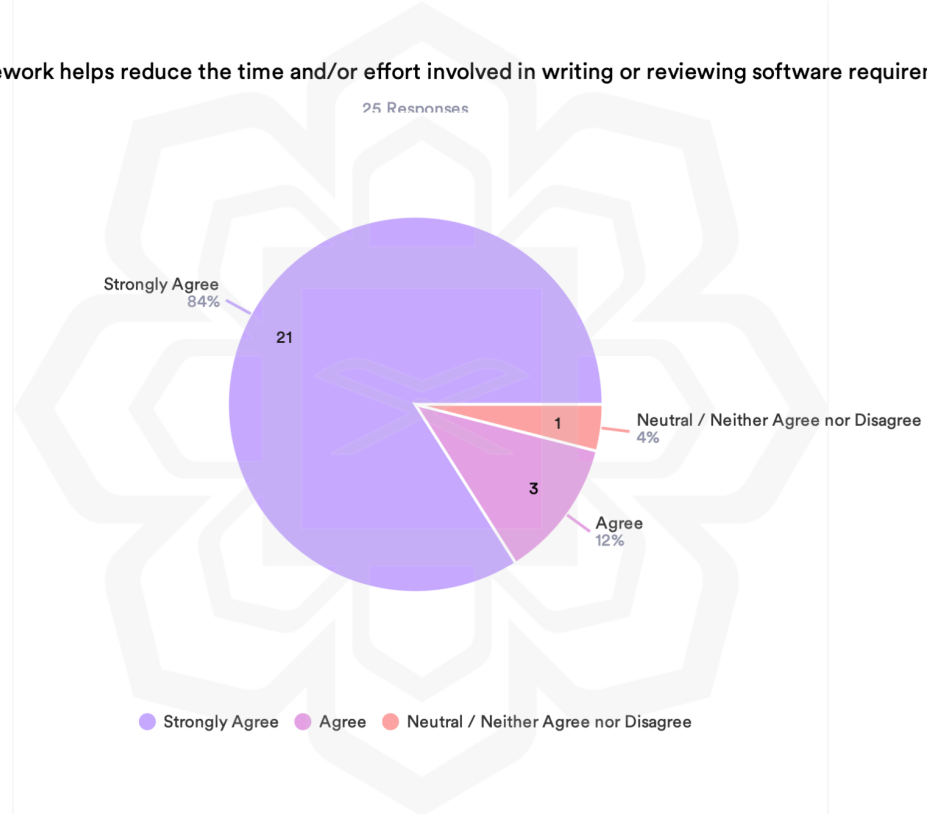


Figure 20: Reduction of Time and Effort

A summary of the analysis for Likert-scale questions are illustrated in Table 10.

Table 11: Summary of the Analysis for Likert-scale Questions

Items	**Description**	**Supporting Data**	**Figure**
Relevance of Framework Components	Users confirmed that the framework contains relevant elements for identifying and resolving ambiguity.	88% strongly agreed	Figure 15
Detection of Overlooked Ambiguities	The framework was seen as effective in uncovering ambiguous terms that are often missed in manual reviews.	84% strongly agreed	Figure 16
Practical Guidance and Usability	Respondents felt that the framework offers sufficient examples or instructions for real-world application.	76% agreed, 20% strongly agreed, 4% neutral	Figure 17
Improvement Over Existing Methods	Compared to current techniques (e.g.,	88% strongly agreed	Figure 18

	manual reviews), the framework was viewed as a significant improvement.		
Efficiency Gains	Participants acknowledged the framework's potential to reduce time and effort in writing or reviewing requirements.	84% agreed	Figure 19

RE experts' feedback provides valuable insights and suggestions for enhancing the framework's usability and effectiveness. The open-ended feedback in Figure 21 offers valuable insights into how the framework's usability can be improved.

Q15 - What improvements or additions would you recommend to enhance the framework's usability, coverage or effectiveness? Please specify any improvements or additions you would recommend to enhance the framework.

Data	Responses
Provide more real-world examples and case studies for each ambiguity rule, illustrating before-and-after scenarios that showcase effective ambiguity resolution.	1
Provide a clear, step-by-step guided pathway that helps novices navigate the framework without feeling overwhelmed, much like “wizards” or interactive checklists which break tasks into manageable chunks and progressively reveal advanced options only when needed	1
Incorporate automated ambiguity detection tools or plugins that can scan requirement documents and flag ambiguous terms and structures. This would reduce manual effort and increase review efficiency. or integrate all those components in an automated tools	1
Develop interactive digital checklists based on the ambiguity rules and repositories, making application in real-time editing more user-friendly	1
integrate with tools. prepare the guideline in pdf to refer.	1
Metacognitive Awareness Combining Automated tools and collaborative reviews Cognitive Biases and inattentional blindness need to explain more on this example. not really understand how it works	1
Clear Documentation & Examples •Include beginner-friendly guides, FAQs, and real-world use cases. Accessibility & Inclusivity •Include multilingual documentation or localization support.	1
no	1

Figure 21: Feedback of the SADF Framework for Improvement

Several recurring themes and suggestion emerged, highlighting both practical enhancements and technical additions. Respondents requested "before-and-after" scenarios to better illustrate how the framework resolves ambiguity in practice, which would help both novice and experienced user apply the rules with confidence. Another frequently mentioned improvement was the inclusion of a clear, step-by-step guided to support new users. Some found the framework initially overwhelming and expressed a desire for a structured onboarding process that simplifies navigation and clarifies where to begin. This was seen as essential for enhancing accessibility and reducing the cognitive load, for particularly for users that are unfamiliar with ambiguity detection. Participants also emphasized the need for automation and tool integration. Suggestions included incorporating plugins or automated tools capable of scanning requirement documents for ambiguous language, along with interactive digital checklists based on the framework’s rules. These additions would streamline the review process and promote consistent application of the framework across different teams.

Additionally, several respondents called for integration with existing tools and platforms, and for the development of downloadable guidelines such as PDF manuals that could be referenced offline. This reflects the importance of practical usability in various professional contexts. One advanced area of feedback addressed the framework's reference to metacognitive awareness and cognitive biases**,** such as inattentional blindness. While this was acknowledged as an important concept, respondents indicated that this section required clearer explanation and more relatable examples to make it actionable and understandable.

Lastly, participants recommended enhancing the framework's documentation and support materials**.** Specifically, they suggested including comprehensive user guides, FAQs, illustrative use cases, and multilingual support to make the framework more inclusive and easier to adopt across diverse user groups. A summary of the feedback is illustrated in Table 11.

Table 12: A Summary of the Feedback

Items	**Respondents Suggestion**	**Intended Benefit**
Real-World Examples & Case Studies	Include before-and-after examples for each ambiguity rule	Enhances clarity and practical understanding
Guided Step-by-Step Pathway	Provide a structured, novice-friendly walkthrough	Reduces user overwhelm and learning curve
Automation & Plugins	Integrate automated ambiguity detection tools or scanning plugins	Saves time and improves consistency
Interactive Digital Tools	Develop checklists and repositories based on rules	Supports repeatable, hands-on application
Tool Integration & Exportable Guides	Enable framework integration with platforms; provide downloadable guides in PDF	Increases usability across different working environments
Metacognitive and Collaborative Review	Clarify how the framework addresses cognitive bias and inattentional blindness	Strengthens team-based review and decision-making
Comprehensive Documentation	Add detailed guides, FAQs, use cases, and multilingual documentation	Improves accessibility, global usability, and support

Figure 21 presents user feedback on how the framework compares to any previous ambiguity detection techniques or tools they have used, including manual approaches. Among the respondents, six indicated that they had not used any prior methods, while the remaining participants confirmed they had experience with existing techniques.

Q16 - Have you used any existing techniques or tools for ambiguity detection, including manual methods? (Yes/No)? If yes, please explain how this framework compares to your previous experience?

Data	Responses
no	6
yes	2
Compared to those methods, this framework is specific and comprehensive; it provides examples of each of the components that people can refer to with their requirements.	1
Yes, I have previously used manual review techniques and collaborative checklist-based reviews for ambiguity detection, as well as basic grammar-checking tools. Compared to those methods, this framework is notably more systematic and comprehensive; it provides specific rules and categorized ambiguous terms, which allows for more consistent identification of issues. The explicit repository of ambiguous terms and integration of requirement patterns (such as EARS and Volere) is particularly helpful. However, manual methods are still time-consuming, and automated support in your framework would further improve effectiveness	1
yes. previously only one method use. this framework comprises few choices of items to be considered when dealing with ambiguity.	1
Yes. •Improved scalability and speed •Consistent results with lower human bias •Better integration with workflows (e.g., document reviews, requirement validation)	1
yes. this framework make me use less time to prepare the requirements	1

Figure 22: Existing Ambiguity Detection Techniques

Among those with prior experience, the responses reflected a positive comparative assessment of the framework. Several respondents emphasized that the framework is more specific and comprehensive than what they had used previously. One participant highlighted that it provides detailed examples for each component, which makes it easier for users to understand and apply in practice which is something that was often missing from earlier approaches.

Another participant noted that the framework is both systematic and well-categorized, particularly in how it groups ambiguous terms and establishes clear rules. This structured design supports more consistent identification of issues, reducing the subjectivity typically involved in ambiguity detection. Additional comments pointed out that the framework offers a range of elements to consider during ambiguity analysis, providing flexibility while maintaining focus.

Respondents also observed that the framework improves scalability and speed, allowing for more efficient ambiguity detection across larger or more complex documents. Notably, one user mentioned that the framework reduces the time needed to prepare requirements, which reinforces earlier feedback in Figure 19 about its ability to save time and effort. Overall, the feedback strongly suggests that the framework outperforms previous methods in terms of clarity, structure, usability, and efficiency. Table 12 shows the summary of respondent's comparison with previous methods.

Table 12: Summary of Respondents' Comparison with Previous Methods

Key Insight	**Respondent Feedback**
Specific and comprehensive	Offers concrete examples for each component, aiding clarity and understanding
Systematic and well-categorized	Organizes ambiguous terms and rules in a way that supports consistent identification
Flexible yet focused	Provides various elements to consider when dealing with ambiguity
Scalable and fast	Enables faster and more scalable ambiguity detection processes
Time efficiency	Reduces the time required to prepare requirement documents

Figure 23 captures respondents' reflections on the potential challenges in implementing the ambiguity detection framework (SADF) in real-world projects.

Q17 - Do you foresee any challenges in implementing this framework in a real-world software project? If yes, please describe them.

18 Responses- 7 Empty

Data	Responses
no	11
Training may be required to raise awareness and effective use of the framework.	1
One challenge is the potential for resistance from team members accustomed to existing requirements processes, especially if they perceive the framework as adding more steps or complexity. Training may be required to raise awareness and understanding of syntactic ambiguity and effective use of the framework. Additionally, projects with tight deadlines may find the full application of the framework difficult unless supported by tool integration. Adapting the framework to fit agile or rapid development workflows may also require additional customization.	1
Language evolves, and the framework might require regular updates to its rule sets or models.	1
If the project involves multiple languages or specialized domains, the framework may need extensions to handle domain-specific terminology and multilingual nuances effectively	1
Incorporating ambiguity detection into existing requirements engineering workflows may face resistance or require training. Analysts and developers need to understand and trust the tool's outputs to effectively act on detected ambiguities. but I personally feels too busy in working hour, so no time to attend long hour training	1
Requirement Engineers, business analysts, and stakeholders may be unfamiliar with the new framework. If the framework introduces too much complexity or requires special training, adoption may be slow or resisted. Engineers may prefer traditional reviews over using new detection tools or grammar rules.	1

Figure 23: Challenge in SADF Implementation

The responses reveal a mix of optimism and caution. 11 respondents indicated there would be no challenges, while others identified several key concerns that could affect adoption and effectiveness. A commonly mentioned challenge is the need for training and awareness. Respondents noted that effective use of the framework requires a solid understanding of syntactic ambiguity and how to apply the framework's rules. Not all team members may be familiar with these concepts, so training and onboarding materials would be necessary to ensure confident and consistent use.

Resistance to change was also highlighted. Team members may be hesitant to adopt the framework if it is seen as adding more steps or complexity to the existing requirements process. This concern may be particularly relevant in teams with established practices or where time and resources are limited. In such cases, resistance could be reduced by integrating the framework into existing tools and workflows. Some respondents pointed out that tight project timelines might hinder full application of the framework unless supported by automation or tool integration. Without such support, teams may revert to traditional review methods to save time. Another consideration is the evolving nature of language. Respondents noted that the framework's rule sets might require periodic updates to remain effective as language use and terminology change over time. Similarly, projects that involve multiple languages or specialized domains may require the framework to be extended to accommodate specific linguistic or contextual needs.

Finally, adoption may be slow if requirements engineers are unfamiliar with the framework or perceive it as too complex. In such cases, they may prefer to stick with familiar, traditional review processes. In summary, while respondents recognized the value of the framework, successful implementation depends on addressing challenges related to training, tool support, user acceptance, language adaptability, and project constraints.

Table 13: Summary of Respondents' feedback on the challenges in SADF implementation

Key Insight	**Respondent Feedback**
Challenges	Out of the total respondents, 11 indicated that they do not foresee any challenges in implementing the framework in a real-world software project.
Training	4 of the respondents stated that training may be required to raise awareness and understanding of using the framework.
Require Update	2 of the respondents agrees that the need extension if the project involves multiple language.

In response to Figure 24, respondents suggested the addition of components that are currently missing from the framework. One respondent recommended including a requirement checklist to ensure that all necessary criteria are systematically considered during the planning and development phases. This addition would enhance the framework's structure by promoting thoroughness and reducing the risk of overlooking critical elements. Another respondent proposed incorporating stakeholder involvement as a core component.

Q18 - Based on your experience, are there any components important or relevant for detecting ambiguity that are missing in this framework?

Data	Responses
no	3
Provide a clear, step-by-step guided pathway that helps novices navigate the framework without feeling overwhelmed, much like "wizards" or interactive checklists which break tasks into manageable chunks and progressively reveal advanced options only when needed. Ensure the interface follows accessibility principles, with clear navigation, visual cues, consistent layouts, and responsive designs so users of varied abilities and devices can use the framework effectively	1
linking ambiguous requirements to their sources and change logs, to better support continuous improvement	1
Incorporating training materials or onboarding guides for new users to effectively apply the framework.	1
Adding a feedback or review loop, where ambiguity findings are tracked and addressed across multiple iterations or stakeholders	1
no all good. but why pronouns in ambiguity repository got two examples?	1
add one more component : Requirements Quality Checklists where this component help to detect ambiguity by using formal checklists that include ambiguity checks to ensures consistency and thoroughness during manual inspections or reviews.	1
put one more component : Stakeholder Involvement and Validation why? because engaging diverse stakeholders during elicitation and	1

Figure 24: Propose of Any Additional Components

This emphasizes the importance of engaging relevant parties such as end users, clients, or regulators throughout the process to ensure that the framework remains aligned with real-world needs and expectations. Overall, these suggestions highlight the need to make the framework more comprehensive, methodical, and user centered.

Table 14: Summary of Respondents' feedback on Additional Components

Key Insight	**Respondent Feedback**
No additional	4 respondents indicated that no additional components are needed for detecting ambiguity in the framework
Stakeholder involvement	1 respondent proposed incorporating stakeholder involvement as a core component
Requirement Checklist	1 respondent recommended including a requirement checklist to ensure that all necessary criteria are systematically considered during the planning and development phase.

The initial research question sought to identify the key components that contribute to the detection of syntactic ambiguities within SRS. The survey results demonstrate that the proposed framework's core components collectively play a crucial role in effectively identifying syntactic ambiguities in SRS documents. These components were derived from literature analysis, expert input, and user validation, and they collectively form the foundation for the proposed detection framework. RE experts concurred that clearly defined requirement attributes are fundamental, as they ensure each requirement is articulated in a manner that permits only one interpretation, thereby minimizing ambiguity. The writing guidelines incorporated in the framework, which emphasize the use of precise language, the avoidance of vague terms, and explicit definition of acronyms and specialized terminology, received strong endorsement. These guidelines promote clarity and consistency, which are essential in mitigating the risk of misinterpretation. Moreover, the inclusion of standardized requirement patterns, such as EARS, IEEE 830-1998/ISO 29148,

RUPP, and Volere, was identified as particularly beneficial. Such patterns provide structured templates that guide the formulation of requirements, reducing linguistic variability that often leads to ambiguity. Likewise, the framework's ambiguity rules and the repository of ambiguous terms including pronouns, modal verbs, quantifiers, positioning words, and subjective expressions were highlighted as important tools. This repository functions as an effective checklist, enabling practitioners to proactively identify and replace ambiguous language during requirements drafting and review. Participants also suggested that extending the framework to encompass semantic and pragmatic ambiguities, integrating automated detection tools, and developing onboarding materials would further enhance its applicability across diverse project contexts.

To answer research question 2, the framework for detecting syntactic ambiguity in Software Requirements Specifications (SRS) was developed through a multi-phase process aimed at creating a structured and comprehensive method to identify ambiguities commonly arising during requirement elicitation and documentation. Grounded in an extensive literature review, the development began by analyzing existing ambiguity detection approaches and identifying gaps in current frameworks, with a focus on integrating both linguistic and engineering perspectives. This informed the design of six key components: requirement quality attributes, ambiguity knowledge incorporating writing guidelines, standardized requirement, linguistic-based ambiguity rules, a glossary of ambiguous terms, and mechanisms for detecting latent ambiguities through meta-cognitive strategies and automated support. The framework's architecture was iteratively refined based on expert validation via structured surveys, ensuring practical applicability and robustness. Visual diagrams were also created to illustrate component relationships and detection workflows. Ultimately, this process resulted in a comprehensive, user-centered framework that supports effective detection of syntactic ambiguities not only during the early phases of RE but also in other phases.

The third research question addressed the evaluation methodology of the framework. To this end, expert validation approach was employed, utilizing quantitative Likert-scale assessments with additional open-ended questions. This approach facilitated a comprehensive appraisal of the

framework's clarity, relevance, coverage, usability, and effectiveness. Majority of respondents 88% strongly agreed that the framework is clear and comprehensible, an important factor for practical adoption. Regarding relevance and scope, approximately 72% of experts strongly agreed that the framework adequately addresses the common syntactic ambiguities encountered in real-world SRS documentation. The framework's applicability in practice was well received, with 76% agreeing that it can be effectively integrated into requirements elicitation and documentation activities. Respondents further acknowledged the framework's enhanced capability to detect ambiguities frequently overlooked during manual reviews, thus contributing significantly to the improvement of requirements quality assurance. In addition, experts concurred that the framework provides a systematic and more efficient alternative to existing ambiguity detection methods, with potential to reduce the time and effort required for writing and reviewing requirements. The quantitative data feedback contributed to a nuanced evaluation, confirming the framework's strengths and identifying areas for future enhancement, including automation integration and expanded instructional resources. This evaluation methodology, grounded in expert judgment supported by structured feedback mechanisms, proved effective in validating the framework within the domain of requirements engineering.

To strengthen the validity of the findings, triangulation was applied by comparing the results from different survey sections and by cross-referencing participant profiles with their responses. The combination of statistical data and thematic insights enabled a comprehensive understanding of how the framework was received by practitioners and researchers. Based on the findings of this phase, recommendations for improvement were identified and used to further refine the framework. The results of the data analysis not only validated the relevance of the proposed approach but also highlighted practical considerations for its implementation in real-world SRS development environments.

The feedback revealed the following core elements contributing to effective syntactic ambiguity detection:

1. Requirement Patterns: Experts acknowledged the relevance of structured requirement patterns such as RUPP (Rational Unified Process Patterns), EARS (Easy Approach to Requirements Syntax), and the Volere template. These patterns helped in standardizing the structure of requirements, making ambiguities easier to identify.
2. Ambiguity Detector: A curated list of ambiguous keywords and syntactic constructs was deemed useful. Experts pointed out that such lists helped flag common sources of misunderstanding, such as vague quantifiers (e.g., "some," "many"), subjective terms (e.g., "user-friendly"), and implicit conditions.
3. Writing Skills and Documentation Practices: The inclusion of best practices for writing clear and testable requirements was appreciated by experts.

When asked about the framework's usability, majority of experts responded very positively. They reported that the framework is clear, understandable, and practically applicable during requirements elicitation and documentation. Most experts agreed that it effectively supports ambiguity detection by covering the common syntactic ambiguity types typically encountered in real-world SRS. Furthermore, respondents indicated that the framework improves upon current ambiguity detection practices by providing a more systematic and comprehensive approach, which also helps reduce the time and effort involved in writing and reviewing requirements. These findings suggest that the framework is user-friendly, enhances the quality of requirements documentation, and can integrate well with existing workflows, though expert feedback also highlighted the need for additional supportive resources such as examples, automated tools integration, and training materials to further improve its usability in practice. Requirement Expert also suggested future enhancements such as automating parts of the ambiguity detection process and incorporating domain-specific examples to make the framework more adaptable across industries. Using thematic grouping, several recurring themes emerged from expert feedback such as follows:

1. Need for Guidance: There was a strong demand for actionable, example-based, and clear step-by-step guidance within the framework.

2. Hybrid Human-Automated: Experts acknowledged that while human judgement is helpful, automated tools are still needed in order to make the framework easy to use

In conclusion, this chapter presented the findings of this study. The results demonstrate that the proposed framework is effective in supporting RE practitioners in the detection and mitigation of syntactic ambiguities within SRS documents. Expert validation highlighted the practicality and relevance of the framework, and analysis confirmed its contribution to structured and unambiguous requirement writing. This chapter also reaffirmed that addressing ambiguity is a multifaceted problem that requires both guidance and practitioner insight achieved by the proposed framework.

5.3 DISCUSSION

While the proposed framework offers a structured and practical approach for detecting syntactic ambiguity in SRS, several limitations were identified during its development and evaluation. Firstly, the framework primarily focuses on syntactic ambiguity, which is only one type of ambiguity found in natural language requirements. Other ambiguity types, such as semantic, lexical, and pragmatic, are not addressed in depth. As a result, the framework does not provide a complete solution to all forms of requirement ambiguity.

Secondly, the framework relies on manual input and expert interpretation in several components, such as applying writing guidelines, interpreting ambiguous terms, and validating the presence of latent ambiguities. Although this enhances flexibility and adaptability, it also introduces subjectivity and variability in how the framework is applied. Unlike some automated tools that integrate with requirement management platforms, this framework has yet to be implemented as a tool-supported solution. Furthermore, its effectiveness may depend on the level of experience and linguistic awareness of the requirements engineer. Therefore, future

enhancements could address this by embedding the framework into a software tool or plugin that supports semi-automated ambiguity detection.

When compared to existing approaches such as the Software Requirement Ambiguity Avoidance Framework (SRAAF) proposed by Gupta and Deraman (2019), the proposed framework offers a more practical and flexible approach focused on the detection and refinement of syntactic ambiguities within software requirements. Unlike SRAAF, which emphasizes preventing ambiguity upfront through formal language constraints, structured templates, and automated verification tools, the proposed framework combines rule-based detection, ambiguity term repositories, writing guidelines, and expert validation to support requirements engineers throughout the elicitation and documentation phases. The core components of the proposed framework, including requirement attributes, writing guidelines, requirement patterns, ambiguity rules, an ambiguity terms repository, and latent ambiguity detection, work together to provide a comprehensive and systematic approach to identifying and clarifying ambiguous statements. While SRAAF provides strong consistency and rigor by restricting the language used in requirement specifications, it can be less adaptable to the natural variability of language and practical workflows. In contrast, the proposed framework prioritizes usability and adaptability, allowing for human judgment to interpret and clarify ambiguous statements, though it still depends somewhat on manual effort and subjective evaluation. Furthermore, whereas SRAAF aims to address a broad spectrum of ambiguities by controlling requirement expression, the proposed framework currently focuses primarily on syntactic ambiguity but offers greater flexibility in handling real-world scenarios through its modular components. Overall, these frameworks complement each other. SRAAF delivers formal, automation-driven prevention of ambiguity, while the proposed framework facilitates ongoing detection and iterative clarification within practical requirements engineering processes.

In conclusion, while the framework demonstrates potential in guiding requirements engineers toward clearer, more unambiguous specifications, its current limitations such as limited

automation, manual dependency, and scope restriction highlight opportunities for future improvement and integration with broader ambiguity management strategies.

5.4 THREATS TO VALIDITY

Like any empirical study, this research is subject to certain threats to validity that must be acknowledged to ensure transparency. One concern is constructing validity, as survey questions and explanatory materials may have influenced respondents toward positive agreement, particularly in the absence of an alternative framework for comparison, meaning their judgements reflect perceived effectiveness rather than comparative benchmarking. Internal validity is also limited since respondents' judgments were based on their understanding of the framework through notes and a video demonstration rather than direct use in practice which introduces the possibility of misinterpretation or overestimation of effectiveness. In terms of external validity, the sample size and respondent background may not fully represent the diversity of requirements engineering practitioners across industries, which restricts generalizability. Finally, conclusion validity is constrained by the reliance on descriptive statistics (frequency and percentages), which provide useful insights but do not establish causal relationships. These limitations highlight the need for further validation through larger samples, comparative benchmarking against existing approaches, and longitudinal case studies in future work to strengthen confidence in the framework's effectiveness in real world practice.

CHAPTER SIX

CONCLUSION AND RECOMMENDATIONS

6.1 INTRODUCTION

This chapter summarizes the key findings and outcomes of this research on the development of a syntactic ambiguity detection framework for SRS. The chapter highlights how the proposed framework integrates multiple elements to support RE practitioners in writing clearer and higher-quality requirements. Furthermore, this chapter outlines practical recommendations for future work and potential enhancements to the framework based on the insights gathered throughout the study.

6.2 CONCLUSION

The research has successfully defined and developed a syntactic ambiguity detection framework that addresses a critical challenge in the RE that is to support identifying and mitigating ambiguity in natural language SRS documents. By combining requirement patterns, ambiguity detection rules, and writing guidelines, the framework provides a comprehensive and systematic approach to improving requirement quality.

The integration of requirement patterns such as RUPP, EARS, and Volere templates enables the framework to recognize common syntactic structures that often result in ambiguity, thus providing a structured method for analysis. The ambiguity detector effectively utilizes linguistic rules and a curated repository of ambiguous terms to flag potential syntactic ambiguities, including latent ambiguities that may not be immediately apparent. Incorporating requirement documentation best practices and writing guidelines empowers practitioners to proactively avoid ambiguity during requirement drafting.

Validation through surveys responded by experts confirmed the framework's relevance and usefulness in practical settings, highlighting its potential to improve SRS quality and reduce misunderstandings in software development. The balanced approach combining automated detection with human expert input enhances both precision and recall in ambiguity identification, fostering better communication and collaboration among stakeholders. Overall, the framework bridges the gap between automated ambiguity detection and human expert judgment, offering a balanced and scalable solution suitable for integration into existing RE workflows.

This research has achieved several important outcomes. The framework was developed based on both theoretical foundations and practical input from experienced RE professionals. The combination of rule-based analysis, ambiguity patterns, writing guidelines, and tacit knowledge provides a layered detection mechanism that supports both novice and experienced practitioners. The modular design allows the framework to be used flexibly at different phase of the requirements lifecycle. Moreover, the inclusion of ambiguity terms repository and writing best practices contributes to capacity building in writing better-quality SRS.

Despite its strengths, several limitations are acknowledged. The validation was based on a limited number of respondents. Nonetheless, all respondents are experts in the RE field and are practicing in writing requirements. Additionally, the framework currently focuses on syntactic ambiguity and does not fully address semantic or pragmatic ambiguity. Automation is also limited at this stage, requiring manual use of the framework. However, the foundation laid by this research presents opportunities for significant advancement and real-world impact.

6.3 FUTURE PLAN AND RECOMMENDATIONS

Looking ahead, the next step in this research is the implementation of a working software system based on the proposed framework. Such a tool would serve as a practical aid for requirements engineers in resolving syntactic ambiguity issues during the creation of SRS documents. The

implementation would ideally integrate with existing RE workflows and requirement management platforms to ensure accessibility and usability in real-world settings.

To guide future improvements, several recommendations are proposed. Firstly, expanding the dataset and validation scope is crucial. Future research should test the framework on larger and more diverse sets of SRS documents, particularly from industrial projects across various domains. This will help evaluate the robustness and generalizability of the framework. Secondly, a user-friendly software tool or plugin should be developed to operationalize the framework. This tool should ideally support real-time feedback, highlight ambiguous terms within requirements, and provide recommendations for revision. Seamless integration with popular RE tools (e.g., IBM DOORS, Jira, or ReqIF Studio) would facilitate adoption among practitioners. In addition, incorporating collaborative review features would strengthen ambiguity detection through combined human and automated assessments. Allowing multiple stakeholders to review and comment on requirement statements can help identify ambiguities more effectively while promoting shared understanding. Another important recommendation is the provision of training and educational materials. Comprehensive documentation, tutorials, workshops, or even online learning modules could be developed to support RE practitioners, particularly those new to the field, in learning how to write clearer and more precise requirements. These materials can also serve to promote the framework's usage and encourage its inclusion in academic and industrial RE training programs. In addition, survey respondents suggested that the framework could be strengthened by including more real-world examples and case studies for each ambiguity component. Developing "before-and-after" scenarios would illustrate how ambiguous requirements can be transformed into clearer specifications, thereby showcasing the framework's effectiveness in practice. This enhancement will be considered in future work to improve usability and practitioner relevance.

In summary, this research presents a novel and effective syntactic ambiguity detection framework that aids RE practitioners in producing quality SRS. By combining linguistic pattern recognition, ambiguity rules, writing best practices, and expert insights, the framework addresses

a significant source of error in software development. While the initial results are promising, further refinement, validation, and tool development are essential to realize its full potential in real-world applications. The recommendations provided offer a clear roadmap for future enhancements that will contribute to advancing the state-of-the-art in requirements quality assurance. Ultimately, the framework aspires to foster better communication, reduce ambiguity-related errors, and support the successful delivery of high-quality software systems.

REFERENCES

Antoniou, C., & Bassiliades, N. (2024). A tool for requirements engineering using ontologies and boilerplates. Automated Software Engineering, 31(5).

Anuar, U., Ahmad, S., & Emran, N. A. (2015). A simplified systematic literature review: Improving software requirements specification quality with boilerplates. In Software Engineering Conference (MySEC), 2015 9th Malaysian (IEEE).

Apshvalka, D., Donina, D., & Kirikova, M. (2009). Understanding the problems of requirements elicitation process: A human perspective. In B. Ch., K. Conboy, M. Lang, G. Wojtkowski, & W. Wojtkowski (Eds.), Information systems development: Challenges in practice, theory and education (Vol. 1, pp. 211–223). Springer.

Araújo, M., Araújo, J., Oliveira, R., Romao, L., & Kalinowski, M. (2025). Domain knowledge in requirements engineering: A systematic mapping study. arXiv:2506.20754. https://arxiv.org/pdf/2506.20754

Arora, C., Wang, F., Tantithamthavorn, C., Huang, K., & Aleti, A. (2018). Requirements-driven automated software testing: A systematic review. https://arxiv.org/pdf/2502.18694

Arora, C., Sabetzadeh, M., Briand, L., & Zimmer, F. (2013). RUBRIC: A flexible tool for automated checking of conformance to requirement boilerplates. In Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013) (pp. 599–602). ACM. https://doi.org/10.1145/2491411.2494591

Arora, C., Sabetzadeh, M., Briand, L., & Zimmer, F. (2013, November). Automated checking of conformance to requirement boilerplates via text chunking: An industrial case study. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (pp. 36–46).

Arora, C., Sabetzadeh, M., Briand, L., & Zimmer, F. (2014). Requirement boilerplates: Transition from manually enforced to automatically verifiable natural language patterns. In 2014 IEEE 4th International Workshop on Requirements Patterns (RePa) (pp. 1–8). IEEE.

Arora, C., Sabetzadeh, M., Briand, L., & Zimmer, F. (2015). Requirements smell: A static analysis technique for detecting problematic patterns in requirements specifications. In Proceedings of the 37th International Conference on Software Engineering (pp. 36–46). IEEE Computer Society.

Ashfaq, F., Bajwa, I. S., & Khan, M. (2021). An intelligent analytics approach to minimize complexity in ambiguous software requirements. Scientific Programming, 2021, Article ID 6616564. https://doi.org/10.1155/2021/6616564

Avison, D., & Fitzgerald, G. (2006). Information systems development: Methodologies, techniques and tools (4th ed.). McGraw-Hill Education.

Ayyaz, M., & Ramzan, F. (2024). New requirement elicitation techniques, processes, and frameworks: Innovation to requirement engineering. *Journal of Software Engineering Practice,* *5*(2), 24–35. https://jsep.info/index.php/jsep/article/download/24/15

Bajceta, A., Leon, M., Afzal, W., Lindberg, P., & Bohlin, M. (2022). Using NLP tools to detect ambiguities in system requirements – A comparison study. In Proceedings of the 5th Workshop on Natural Language Processing for Requirements Engineering @ REFSQ, Birmingham, UK. CEUR Workshop Proceedings, Vol. 3122. Retrieved from https://ceur-ws.org/Vol-3122/NLP4RE-paper-3.pdf

Bass, L., Clements, P., & Kazman, R. (2012). Software architecture in practice (3rd ed.). Addison-Wesley.

Bashir, S., Ferrari, A., Khan, M. A., Strandberg, P. E., Haider, Z., Saadatmand, M., & Bohlin, M. (2025). Requirements ambiguity detection and explanation with LLMs: An industrial study. In ICSME 2025 – Industry Track.

Beg, A., O'Donoghue, D., & Monahan, R. (2025). Formalising software requirements with large language models. arXiv preprint arXiv:2506.14627. https://arxiv.org/pdf/2506.14627

Berry, D. M., Kamsties, E., & Krieger, M. M. (2003). From contract drafting to software specification: Linguistic sources of ambiguity. Technical Report, University of Waterloo.

Berry, D. M., & Kamsties, E. (2003). Ambiguity in requirements specification. In Perspectives on software requirements: Principles and techniques (pp. 191–194). Springer.

Berry, D. M., Kamsties, E., & Kamsties, M. (1994). Ambiguity in requirements specifications. IEEE Software, 11(2), 98–109.

Benfell, G. (2020). The role of tacit knowledge in requirements engineering. Journal of Systems and Software, 165, 110564.

Boehm, B., & Basili, V. R. (2001). Software defect reduction top 10 list. Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili, pp. 426–432. Springer. https://doi.org/10.1007/978-3-642-45120-9_25 [ieeexplore.ieee.org]

Bourque, P., & Fairley, R. E. (Eds.). (2014). Guide to the software engineering body of knowledge (SWEBOK) (Version 3.0). IEEE Computer Society.

Ceccato, M., Kiyavitskaya, N., Zeni, N., Mich, L., & Berry, D. M. (2004). Ambiguity identification and measurement in natural language texts. University of Trento.

Chantree, F. J., de Roeck, A., Nuseibeh, B., & Willis, A. (2006). Identifying nocuous ambiguity in natural language requirements. The Open University, UK.

Chattopadhyay, S., Nelson, N., Au, A., Morales, N., Sanchez, C., Pandita, R., & Sarma, A. (2022). Cognitive biases in software development. Communications of the ACM. Retrieved from https://cacm.acm.org/research/cognitive-biases-in-software-development/

Chazette, L., Brunotte, W., & Speith, T. (2022). Explainable software systems: From requirements analysis to system evaluation. Requirements Engineering, 27(4),

457–487. https://doi.org/10.1007/s00766-022-00393-5

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). Model checking. Cambridge. https://books.google.com/books/about/Model_checking.html?id=Nmc4wEaLXFEC

Cleland-Huang, J., Gotel, O., Hayes, J. H., Zisman, A., & Egyed, A. (2006). Software traceability: Trends and future directions. In Proceedings of the Future of Software Engineering (FOSE 2007) (pp. 162–176). IEEE. https://doi.org/10.1109/FOSE.2007.19

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. Computer, 34(11), 131–133.

Cohen, L., Manion, L., & Morrison, K. (2018). *Research methods in education* (8th ed.). Routledge.

Coughlan, D., & Macredie, R. D. (2014). Effective communication in requirements elicitation: A comparison of methodologies. Requirements Engineering, 7(2), 47–60.

Da Silva, S., Gupta, R., & Monzani, D. (2023). Editorial: Highlights in psychology: cognitive bias. Frontiers in Psychology, 14, Article 1242809. https://doi.org/10.3389/fpsyg.2023.1242809

Dalpiaz, F., van der Schalk, I., & Lucassen, G. (2018). Pinpointing ambiguity and incompleteness in requirement engineering via information visualization and NLP. In Proceedings of the RESFQ 2018 Workshop. Utrecht University.

Damian, D., Zowghi, D., & Neill, C. (2020). Improving requirements communication: The challenge of natural language and structured specifications. Information and Software Technology, 127, 106361.

Dampney, C. N. G., Busch, P., & Richards, D. (2002). The meaning of tacit knowledge. Australasian Journal of Information Systems, 10(1), 3–13. https://doi.org/10.3127/ajis.v10i1.438

Dar, H. S., Imtiaz, S., & Lali, M. I. (2022). Reducing requirements ambiguity via gamification: Comparison with traditional techniques. Computational

Intelligence and Neuroscience, 2022, Article ID 3183411. https://doi.org/10.1155/2022/3183411

Davidson, E. J., Sanders, G. L., & Land, F. (2003). Identity construction in technology user narratives: A research note. Information Systems Research, 14(3), 352–370.

Davis, A. M., Overmyer, S. P., Jordan, K. D., Caruso, J. D., Dandashi, F., Dinh, A., … Kincaid, G. (1993). Identifying and measuring quality in a software requirements specification. In Software Metrics Symposium, 1993. Proceedings., First International (pp. 141–152). IEEE.

Davis, J. A., Clark, M. A., Cofer, D. D., Fifarek, A., Hinchman, J., Hoffman, J. A., Hulbert, B. W., Miller, S. P., & Wagner, L. G. (2013). Study on the barriers to the industrial adoption of formal methods. In C. Pecheur & M. Dierkes (Eds.), FMICS 2013: Formal Methods for Industrial Critical Systems (Vol. 8187, pp. 63–77).

Elghariani, K., Kama, N., Mohd Azmi, N. F., & Abu Bakar, N. A. (2019). Implicit thinking knowledge injection framework for Agile requirements engineering. International Journal of Advanced Computer Science and Applications, 9(11), 163-170

Emam, K. E., & Koru, A. G. (2008). A replicated survey of IT software project failures. IEEE Software, 25(5), 84–90.

Fabbrini, F., Fusani, M., Gnesi, S., & Lami, G. (2001). The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. In Proceedings of ICSSEA.

Feiman, R., & Snedeker, J. (2016). The logic in language: How all quantifiers are alike, but each quantifier is different. Cognitive Psychology, 87, 29–52.

Fantechi, A., Gnesi, S., & Semini, L. (2024). Exploring LLMs' Ability to Detect Variability in Requirements. In Requirements Engineering: Foundation for Software Quality (REFSQ 2024), Lecture Notes in Computer Science, vol 14588, pp. 178–188. Springer. https://doi.org/10.1007/978-3-031-57327-9_11

Femmer, H., Vogelsang, A., & Philippsen, M. (2017). Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources. In Requirements Engineering: Foundation for Software Quality (REFSQ 2010), Lecture Notes in Computer Science, vol 6182, pp. 218–232. Springer. https://doi.org/10.1007/978-3-642-14192-8_20

Ferrari, A., Spoletini, P., & Gnesi, S. (2016). Ambiguity and tacit knowledge in requirements elicitation interviews. Requirements Engineering, 21(3), 333–355.

Ferrari, A., Lipari, G., Gnesi, S., & Spagnolo, G. O. (2014). Pragmatic ambiguity detection in natural language requirements. In 2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE) (pp. 1–8). IEEE. https://doi.org/10.1109/AIRE.2014.6894849

Fischer, W., & Bauer, B. (2010). Domain dependent semantic requirement engineering. In CAiSE'10 Workshop DE@CAiSE'10, Hammamet, Tunisia, pp. 6–17. https://ceur-ws.org/Vol-602/DE_CAiSE10_paper2_Fischer.pdf

Friedrich, F., Wagner, S., & Pohl, K. (2013). Model-based detection of ambiguities in natural language requirements specifications. Requirements Engineering, 18(3), 233–251.

Fuchs, N. E., & Schwitter, R. (1996). Attempto controlled English (ACE).

Funk, M., Frattini, D., & Vogelsang, A. (2017). Towards the detection of ambiguity in natural language requirements. In 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW) (pp. 393–399).

Gavran, I., Štemberger, M. I., & Orehek, E. (2021). Natural language processing in requirements engineering: A systematic literature review. Information and Software Technology, 135, 106577.

Gemino, A., & Parker, D. (2011). Visual support for use case modeling: An experiment to determine the effectiveness of use case diagrams. In Theoretical and Practical Advances in Information Systems Development: Emerging Trends and Approaches (pp. 1–19). IGI Global.

Gervasi, V., & Zowghi, D. (2005). Reasoning about inconsistencies in natural language requirements. In A. Sutcliffe & A. Finkelstein (Eds.), Requirements Engineering: A Good Practice Guide (pp. 185–206). Springer.

Gervasi, V., Ferrari, A., Zowghi, D., & Spoletini, P. (2019). Ambiguity in requirements engineering: Towards a unifying framework. In M. H. ter Beek, A. Fantechi, & A. Gnesi (Eds.), From software engineering to formal methods and tools, and back (pp. 191–210). Springer.

Gervasi, V., & Vogelsang, A. (Eds.). (2022). Requirements Engineering: Foundation for Software Quality (Vol. 13216). Springer. https://doi.org/10.1007/978-3-030-30985-5_12

Gervasi, V., Ferrari, A., & Tolomei, G. (2019). Ambiguity detection in natural language requirements. Requirements Engineering, 24(3), 289–307.

Gleich, B., Creighton, O., & Kofler, L. (2010). Ambiguity detection: Towards a tool explaining ambiguity sources. In Requirements Engineering: Foundation for Software Quality: 16th International Working Conference, REFSQ 2010 (pp. 218–232). Springer.

Goldin, L., & Berry, D. M. (1994). AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Engineering, 1(4), 375–412.

Gorschek, T., Garre, P., Larsson, S., & Wohlin, C. (2006). A model for technology transfer in practice. IEEE Software, 23(6), 88–95. https://doi.org/10.1109/MS.2006.147 [ieeexplore.ieee.org]

Gourlay, S. (2006). Towards conceptual clarity for 'tacit knowledge': A review of empirical studies. Knowledge Management Research & Practice, 4(1), 60–69.

Grbich, C. (2013). *Qualitative data analysis: An introduction* (2nd ed.). SAGE Publications.

Grünbacher, P., & Briggs, R. O. (2001). Surfacing tacit knowledge in requirements negotiation: Experiences using EasyWinWin. In 34th Annual Hawaii International Conference on System Sciences (pp. 1–10). IEEE.

Großer, K., Ahmadian, A. S., Rukavitsyna, M., Ramadan, Q., & Jürjens, J.

(2024). Benchmarking requirement template systems: comparing appropriateness, usability, and expressiveness. Requirements Engineering, 29, 481–522.

Gupta, A., Sharma, S., Negi, V., & Kush, A. (2019). Challenges in requirement engineering process: An empirical study. International Journal of System Assurance Engineering and Management, 10(3), 505–518.

Gupta, A. K., & Deraman, A. (2019). A framework for software requirement ambiguity avoidance. International Journal of Electrical and Computer Engineering (IJECE), 9(6), 5436–5445.

Hall, A. (1990). Seven myths of formal methods. IEEE Software, 7(5), 11–19.

Händel, M., Naujoks-Schober, N., & Dresel, M. (2023). Metacognitive monitoring via strategies and judgments: Different phases, different objects. Zeitschrift für Entwicklungspsychologie und Pädagogische Psychologie, 55(2–3), 67–76.

Haron, H., & Ghani, A. A. A. (2015). A survey on ambiguity awareness towards Malay system requirement specification (SRS) among industrial IT practitioners. Procedia Computer Science, 72, 261–268.

Hayman Oo, K., Nordin, A., Ritahani Ismail, A., & Sulaiman, S. (2018). An analysis of ambiguity detection techniques for software requirements specification (SRS). International Journal of Engineering & Technology, 7(2.29), 501.

Hall, A. (1990). Using formal methods to improve software quality. IEEE Computer, 23(5), 11–13.

Hoffer, J. A., George, J. F., & Valacich, J. S. (2004). Modern systems analysis and design (4th ed.). Prentice Hall.

Hughes, B., Cotterell, M., & Mall, R. (2007). Software project management (5th ed.). McGraw-Hill Education.

Hembree, J. (2021). Challenges in detecting pragmatic ambiguity in NLP systems. Journal of Computational Linguistics, 47(2), 134–150.

Hull, E., Jackson, K., & Dick, J. (2011). *Requirements engineering* (3rd ed., pp. 85–88). Springer.

Huisman, M., Gurov, D., & Malkis, A. (2020). Formal methods: From academia to industrial

practice – A travel guide.

Hussain, W., Ahmad, N., Khan, R. A., & Akbar, M. (2023). A systematic literature review on requirements engineering and its practices. Journal of Systems and Software, 196, 111534.

Hussain, A., Ahmed, H., Khamaj, A., & Nawi, M. N. M. (2021). A model of consequences of ambiguous requirements. Journal of Southwest Jiaotong University, 56(6), 599–609. https://doi.org/10.35741/issn.0258-2724.56.6.52

IEEE. (1998). IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998). Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/IEEESTD.1998.88286

INCOSE. (2023). INCOSE guide for writing requirements (4th ed.). International Council on Systems Engineering.

ISO/IEC/IEEE (2023). *Systems and Software Engineering — System Life Cycle Processes* (ISO/IEC/IEEE Standard 15288:2023).

ISO/IEC/IEEE. (2018). Systems and software engineering — Life cycle processes — Requirements engineering (ISO/IEC/IEEE 29148:2018). International Organization for Standardization.

Jia, H., Morris, R., Ye, H., Sarro, F., & Mechtaev, S. (2025). Automated repair of ambiguous natural language requirements. arXiv preprint arXiv:2505.07270.

Jurafsky, D., & Martin, J. H. (2021). Speech and Language Processing (3rd ed., draft). Stanford University.

Kahneman, D. (2011). Thinking, fast and slow. Farrar, Straus and Giroux. https://psycnet.apa.org/record/2011-26535-000 [psycnet.apa.org]

Kamsties, E. (2001). Surfacing ambiguity in natural language requirements. In Perspectives on requirements engineering (pp. 44–48). Kluwer Academic Publishers.

Kamsties, E., Berry, D. M., & Paech, B. (2001). Detecting ambiguities in requirements documents using inspections. In J. C. S. Leite & J. Doorn (Eds.), Perspectives on Software Requirements (pp. 7–44). Springer. https://doi.org/10.1007/978-

1-4615-0465-8_2

Kamsties, E. (2007). Understanding ambiguity in requirements engineering. In A. Aurum & C. Wohlin (Eds.), Engineering and managing software requirements (pp. 245–266).

Kato, T., Tsuda, K., & Masuda, S. (2022). A method of ambiguity detection in requirement specifications by using a knowledge dictionary. Procedia Computer Science, 207, 1482–1489. https://doi.org/10.1016/j.procs.2022.09.205

Keil, M., Cule, P. E., Lyytinen, K., & Schmidt, R. C. (1998). A framework for identifying software project risks. Communications of the ACM, 41(11), 76–83.

Kiyavitskaya, N., Zeni, N., Mich, L., & Berry, D. M. (2008). Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements Engineering, 13(3), 207–239.

Kof, L. (2005). Natural language processing for requirements engineering: Applicability to ambiguous requirements. Requirements Engineering, 10(4), 249–262.

Kotonya, G., & Sommerville, I. (1998). *Requirements engineering: Processes and techniques*. Chichester, UK: John Wiley & Sons.

Kuhn, T. (2014). A survey and classification of controlled natural languages. Computational Linguistics, 40(1), 121–170. https://doi.org/10.1162/COLI_a_00168

Kucharska, W., & Erickson, S. G. (2023). Tacit knowledge acquisition & sharing, and its influence on innovations: A Polish/US cross-country study. Journal of Innovation & Knowledge.

Kücherer, C. (2018). Domain-specific adaptation of requirements engineering methods [Doctoral dissertation, Heidelberg University].

Khezri, R. (2017). Automated detection of syntactic ambiguity using shallow parsing and web data (Master's thesis, University of Gothenburg).

Kwizera, I. L. (2025). Overcoming the ambiguity requirement using generative AI (Master's thesis, Mälardalen University). DiVA Portal. https://www.divaportal.org/smash/get/diva2:1931301/FULLTEXT01.pdf

Li, Y., Ma, Y., & Ma, L. (2005). Problems and counter measures in requirements elicitation. In International Conference on Computer Science and Software Engineering.

Li, Z., Zhang, M., & Chen, W. (2014). Ambiguity-aware ensemble training for semi-supervised dependency parsing. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Vol. 1, pp. 457–467). Association for Computational Linguistics. https://doi.org/10.3115/v1/P14-1043

Liu, A., Wu, Z., Michael, J., Suhr, A., West, P., Koller, A., Swayamdipta, S., Smith, N., & Choi, Y. (2023). We're afraid language models aren't modeling ambiguity. In H. Bouamor, J. Pino, & K. Bali (Eds.), Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (pp. 790–807). Association for Computational Linguistics.

Lombriser, P., Dalpiaz, F., Lucassen, G., & Brinkkemper, S. (2016). Gamified requirements engineering: Model and experimentation. In Requirements Engineering: Foundation for Software Quality (pp. 171–187). Springer. https://doi.org/10.1007/978-3-319-30282-9_12

Luisa, M., Mariangela, F., & Pierluigi, N. I. (2004). Market research for requirements analysis using linguistic tools. Requirements Engineering, 9(1), 40–56.

M. H. Osman and M. F. Zaharin, "Ambiguous Software Requirement Specification Detection: An Automated Approach," in Proc. Int. Workshop on Requirement Engineering and Testing (RET 2018), Gothenburg, Sweden, June 2018, pp. 1–8. doi: 10.475/123_4

Macaulay, L. (1996). Requirements for requirements engineering techniques. In Proceedings of the Second International Conference on Requirements

Engineering (pp. 157–164).

Maiden, N., & Rugg, G. (1996). ACRE: Selecting methods for requirements acquisition. Software Engineering Journal, 11(3), 183–192.

Mavin, A., Rupp, C., & Friedrich, J. (2009). EARS: Easy Approach to Requirements Syntax.

Mavin, A., Wilkinson, P., Harwood, A., & Novak, M. (2017). Easy Approach to Requirements Syntax (EARS): An open-source approach to improving requirements quality. Information and Software Technology, 92, 164–185.

Mehrparvar, B., & Pezzelle, S. (2024). Detecting and translating language ambiguity with multilingual LLMs. In J. Sälevä & A. Owodunni (Eds.), Proceedings of the Fourth Workshop on Multilingual Representation Learning (MRL 2024) (pp. 310–323). Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.mrl-1.26

Mich, L., Franch, M., & Novi Inverardi, P. (2004). *Market research for requirements analysis using linguistic tools*. *Requirements Engineering*, 9(1), 40–56. Springer

Microsoft Corporation. (2012). Microsoft manual of style (4th ed.). Microsoft Press.

Mohamed, M., Basir, N., & Abdul Salam, S. (2015). Facilitating tacit knowledge acquisition within requirements engineering. Journal of Theoretical and Applied Information Technology, 82(2), 308–315.

Mohamed, K. A., Din, J., & Baharom, S. (2022). A tool to detect pragmatic ambiguity with possible interpretations suggestion in software requirement specifications. International Journal of Synergy in Engineering and Technology, 3(2), 52–60. https://www.researchgate.net/publication/369753958

Mohedas, I., Daly, S. R., Loweth, R. P., Huynh, L., Cravens, G. L., & Sienko, K. H. (2022). The use of recommended interviewing practices by novice engineering designers to elicit information during requirements development. Design Science, 8, e16. https://doi.org/10.1017/dsj.2022.4

Mourya, S., & Singh, V. B. (2025). Code smell detection by exploiting parameter tuning techniques for sustainable software evolution. International

Journal of System Assurance Engineering and Management. https://doi.org/10.1007/s13198-025-02951-w

Nartker, M., Firestone, C., Egeth, H., & Phillips, I. (2025). Sensitivity to visual feature in inattentional blindness. eLife, 13, RP100337.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: A roadmap. In Proceedings of the Conference on The Future of Software Engineering (pp. 35–46). ACM.

Osama, S., & Aref, M. (2018). Detecting and resolving ambiguity approach in requirement specification: Implementation, results and evaluation. International Journal of Intelligent Computing and Information Sciences, 18(1), 27–36. https://doi.org/10.21608/ijicis.2018.15909

Osama, M. (2020). Contextual limitations in pragmatic ambiguity detection. International Journal of Language and AI, 12(3), 88–10

Palomares Bonache, C. (2016). Definition and use of software requirement patterns in requirements engineering (Doctoral dissertation). Universitat Politècnica de Catalunya.

Pham, B. (2020). Parts of Speech Tagging: Rule-Based. Harrisburg University of Science and Technology.

Polanyi, M. (1966). *The tacit dimension* (p. 4). Routledge & Kegan Paul.

Polpinij, J. (2009). An ontology-based text processing approach for simplifying ambiguity of requirement specifications. In 2009 IEEE Asia-Pacific Services Computing Conference.

Pohl, K. (2010). Requirements engineering: Fundamentals, principles, and techniques. Springer.

Pohl, K., & Rupp, C. (2011). Requirements engineering fundamentals: A study guide for the Certified Professional for Requirements Engineering exam – Foundation Level (IREB compliant). Rocky Nook. (p. 36)

Pandey, D., Suman, U., & Ramani, A. K. (2010). An effective requirement engineering process model for software development and requirements management. In 2010 International Conference on Advances in Recent Technologies in Communication and Computing (pp. 287–291). IEEE.

Penmetsa, S., & Lingampalli, N. (2024). An empirical analysis of the usage of requirements

attributes in requirements engineering research and practice. In Proceedings of the 16th International Conference on Computational Collective Intelligence (ICCCI 2024) (pp. 29–40). Springer.

Pereira dos Reis, J., Brito e Abreu, F., Carneiro, G. F., & Anslow, C. (2022). Code smells detection and visualization: A systematic literature review. Archives of Computational Methods in Engineering, 29, 47–94. https://doi.org/10.1007/s11831-021-09566-x

Rasheed, A., Zafar, B., Shehryar, T., Aslam, N. A., Sajid, M., Ali, N., Dar, S. H., & Khalid, S. (2021). Requirement engineering challenges in agile software development. Mathematical Problems in Engineering, 2021, Article ID 6696695, 1–18

Raj, A., Rahim, M. A. B. U., Hussain, S., & Zia, I. (2025). Enhancing software requirements quality: Ambiguity detection and resolution using large language models. In Computational Science and Computational Intelligence (CSCI 2024) (pp. 340–355). Springer.

Riaz, M. Q., Butt, W. H., & Rehman, S. (2019). Automatic detection of ambiguous software requirements: An insight. In 5th International Conference on Information Management.

Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., & Ceccato, M. (2010). The role of requirement patterns in model-driven software development. In 2010 IEEE 18th International Requirements Engineering Conference (pp. 8–17). IEEE.

Ribeiro, C., & Berry, D. M. (2020). The prevalence and severity of persistent ambiguity in software requirements specifications:

Robertson, S., & Robertson, J. (2012). Mastering the requirements process (3rd ed.).

Addison-Wesley. Rupp, C., et al. (2014). Requirement templates for unambiguous documentation.

S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, “TAPHSIR: Towards AnaPHoric Ambiguity Detection and ReSolution In Requirements,” in *Proc. 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE)*, Singapore, 2022, pp. 1–

13.

Sabriye, A. O. J. A., & Wan Zainon, W. M. N. W. (2017). A framework for detecting ambiguity in software requirement specification. In 2017 8th International Conference on Information Technology (ICIT) (pp. 155–160). IEEE. https://doi.org/10.1109/ICITECH.2017.8080002

Sabriye, A. O. J. A., & Zainon, W. M. N. W. (2018). An approach for detecting syntax and syntactic ambiguity in software requirement specification. Journal of Theoretical & Applied Information Technology, 96(8).

Sami, M. A., Rasheed, Z., Waseem, M., Zhang, Z., Herda, T., & Abrahamsson, P. (2024). Prioritizing software requirements using large language models. arXiv preprint. https://arxiv.org/abs/2405.01564

Sawyer, P. (2009). Requirements engineering. Springer.

Serna, E. M., & Serna, A. A. (2023). A model for documenting requirements elicitation. Ingeniería, 28(2), Article e19411.

Shah, T., & Patel, S. (2014). A review of requirement engineering issues and challenges in various software development methods. International Journal of Computer Applications, 99, 36–45.

Sharma, R., Gupta, A., & Srivastava, M. (2020). Techniques for mitigating syntactic ambiguities in software requirements specification. International Journal of Software Engineering and Its Applications.

Shi, Z., Castellucci, G., Filice, S., Kuzi, S., Kravi, E., Agichtein, E., Rokhlenko, O., & Malmasi, S. (2025). Ambiguity detection and uncertainty calibration for question answering with large language models. In T. Cao et al. (Eds.), Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025) (pp. 41–55). Association for Computational Linguistics. Singh, T., Patidar, V.,

Singh, M., & Rocha, Á. (2025). Schema understandability: A comprehensive empirical study of requirements metrics. Information, 16(2), 155.

Sinpang, J. S., Sulaiman, S., & Idris, N. (2017). Detecting ambiguity in requirements

analysis using Mamdani fuzzy inference. UTeM Open Journal System. Universiti Teknikal Malaysia Melaka.

Snijders, R., Dalpiaz, F., & Brinkkemper, S. (2018). REfine: A gamified platform for participatory requirements engineering. In *CrowdRE 2015*. Bournemouth University.

Sommerville, I. (2016). Software engineering (10th ed., pp. 105–106, 108). Pearson Education Limited.

Sommerville, I. (2004). Software engineering (7th ed., pp. 9–11). Addison-Wesley.

Sommerville, I. (2011). Software engineering (9th ed., pp. 82–117, 122–24). Addison-Wesley.

Sommerville, I., & Sawyer, P. (1997). Requirements engineering: A good practice guide (pp. 122–124). John Wiley & Sons.

Sternberg, R. J., Wagner, R. K., Williams, W. M., & Horvath, J. A. (2000). Testing common sense.American Psychologist, 55(4), 355–366.

Tripathi, V., & Goyal, A. K. (2014). Agile requirement engineer: Roles and responsibilities.

Umar, M. A., & Lano, K. (2024). Advances in automated support for requirements engineering: A systematic literature review. Requirements Engineering, 29, 177–207

Vallejo, D., Escalona, M. J., & Mejías, M. (2020). Using requirement templates to reduce ambiguity in natural language specifications. Journal of Systems and Software, 170, 110798. https://doi.org/10.1016/j.jss.2020.110798

Verner, J., Sampson, J., & Cerpa, N. (2008). What factors lead to software project failure? In 2008 2nd International Conference on Research Challenges in Information Science (pp. 71–80). IEEE.

Volere (2016). Volere Requirements Specification Template (Edition 18). Atlantic Systems Guild.

Wiegers, K. E., & Beatty, J. (2013). Software requirements (3rd ed., pp. 21–22). Microsoft Press.

Yadav, M., Jain, A., & Mishra, R. (2018). Dependency parsing for ambiguity detection in SRS. In Proceedings of the IEEE International Conference on Software Engineering.

Zhao, W., & Kamalrudin, M. (2018). Towards effective automated requirements ambiguity detection. In Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC) (pp. 298–305). IEEE.

Zowghi, D., & Coulin, C. (2005). Requirements elicitation: A survey of techniques, approaches, and tools. In A. Aurum & C. Wohlin (Eds.), Engineering and managing software requirements (pp. 19–46). Springer.

Zowghi, D., & Gervasi, V. (2003). On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software Technology, 45(14), 993–1009.

Zowghi, D., & Paryani, S. (2003). Teaching requirements engineering through role-playing: Lessons learnt. In Proceedings of the 11th IEEE International Requirements Engineering Conference (pp. 233–241). IEEE.

APPENDIX I: SURVEY QUESTIONNAIRES

Survey on the Syntactic Ambiguity Detection Framework in Software Requirements Engineering

Dear Participants,

I am a Master's student in Computer Science and Information Technology at the International Islamic University Malaysia, conducting a research study on a Syntactic Ambiguity Detection Framework for Software Requirements Specifications (SRS). This framework is designed to help Requirement Engineers to detect syntactic ambiguities in SRS documents. The goal of this survey is to gather insights from professionals involved in requirements engineering or software development who have experience working with software specifications in the industry. Your feedback will help assess the relevance, completeness, and usability of the proposed framework. Participation is voluntary, and all responses will be kept confidential and used strictly for academic purposes. The survey will take approximately 15–20 minutes to complete. Thank you for your time and valuable contribution!

Background of Study

Ambiguities in Software Requirements Specifications (SRS) are a common challenge in software development. When the structure or wording of requirements is unclear, it can lead to misunderstandings between stakeholders, developers, and testers resulting in project delays, defects, or increased costs. This study focuses on syntactic ambiguity, a type of ambiguity that arises from sentence structure. It occurs when a sentence can be interpreted in more than one way due to its grammar or word arrangement. Example:

"The system shall notify the user with an email and a text message."– It is unclear

statements, supporting clearer communication and better software quality. Before proceeding, please watch the video below for an overview of the proposed framework.

Understanding the Framework

Section 1 : Participant Background

Q1 - What is your role in the organization? *

- ○ Requirement Engineer
- ○ Software Engineer
- ○ Business/System Analyst
- ○ QA/Test Engineer
- ○ Lecturer/Educator
- ○ Other

Q2 - Do you have formal training on Requirement Engineering?

- [ ] University-level course(s)
- [ ] I am IREB CPRE Certified
- [ ] I am CCBA Certified
- [ ] I have attended courses/training/workshop/seminar related to Requirement Engineering
- [ ] No, I do not have formal training in Requirements Engineering
- [ ] No, but I have gained knowledge through practical experience
- [ ] No, I am not familiar

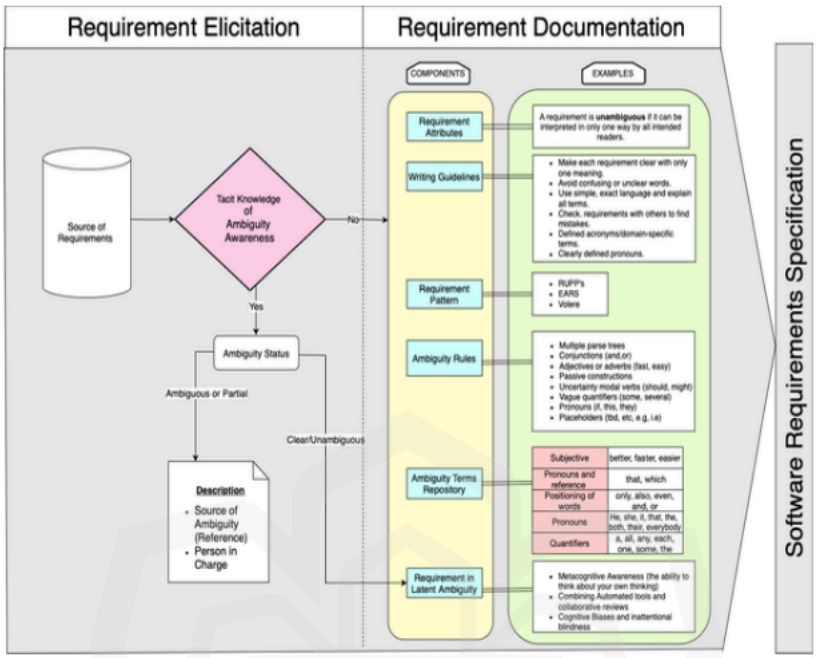
Q3 - Tick any requirement patterns (boilerplates or structures) that you are familiar with.

- [ ] EARS
- [ ] RUPP'S
- [ ] Volere Requirement Shell
- [ ] IEEE 830-1998/ ISO 29148:2018
- [ ] Planguage Requirement Shell
- [ ] User Story Template

Q4 - How many years of experience do you have in software requirements engineering?

- ( ) Less than 1 year
- ( ) 1-3 years
- ( ) 4-6 years
- ( ) More than 6 years

Section 2 : Framework Evaluation



Syntactic Ambiguity Detection Framework in SRS

Instructions: Please review the above framework and indicate your level of agreement with the following statements. (Scale: 1 Strongly Disagree. 5 = Strongly Agree)

Q5 - The proposed framework is clear and understandable.

- ○ Strongly Disagree
- ○ Disagree
- ○ Neutral / Neither Agree nor Disagree
- ○ Agree
- ○ Strongly Agree

Q6 - The framework components are relevant to detecting functional

- Strongly Disagree
- Disagree
- Neutral / Neither Agree nor Disagree
- Agree
- Strongly Agree

Q7 - The framework covers the general range of syntactic ambiguity types typically encountered in real-world software requirements.

- Strongly Disagree
- Disagree
- Neutral / Neither Agree nor Disagree
- Agree
- Strongly Agree

Q8 - The framework is easy to understand and can be practically applied during the elicitation and documentation phases.

- Strongly Disagree
- Disagree
- Neutral / Neither Agree nor Disagree
- Agree
- Strongly Agree

Q9 - The framework has the potential to reduce the risk of misinterpretation in software requirements.

- Strongly Disagree
- Disagree
- Neutral / Neither Agree nor Disagree
- Agree
- Strongly Agree

Q10 - The framework includes relevant components for identifying and resolving ambiguous terms.

- Strongly Disagree
- Disagree
- Neutral / Neither Agree nor Disagree
- Agree
- Strongly Agree

Q11 - The framework effectively identifies ambiguities in software requirements that are commonly missed during manual reviews processes.

- Strongly Disagree
- Disagree
- Neutral / Neither Agree nor Disagree
- Agree
- Strongly Agree

Q12 - The framework provides sufficient examples or guidance for practical use.

- Strongly Disagree
- Disagree
- Neutral / Neither Agree nor Disagree
- Agree
- Strongly Agree

Q13 - Compared to your current ambiguity detection practices (e.g., manual review or existing techniques) the framework offers an improvement.

- Strongly Disagree
- Disagree
- Neutral / Neither Agree nor Disagree
- Agree
- Strongly Agree

Q14 - The framework helps reduce the time and/or effort involved in writing or reviewing software requirements.

- ○ Strongly Disagree
- ○ Disagree
- ○ Neutral / Neither Agree nor Disagree
- ○ Agree
- ○ Strongly Agree

Q15 - What improvements or additions would you recommend to enhance the framework's usability, coverage or effectiveness? Please specify any improvements or additions you would recommend to enhance the framework. *

Q16 - Have you used any existing techniques or tools for ambiguity detection, including manual methods? (Yes/No)? If yes, please explain how this framework compares to your previous experience?

Q17 - Do you foresee any challenges in implementing this framework in a real-world software project? If yes, please describe them.

Q18 - Based on your experience, are there any components important or relevant for detecting ambiguity that are missing in this framework?

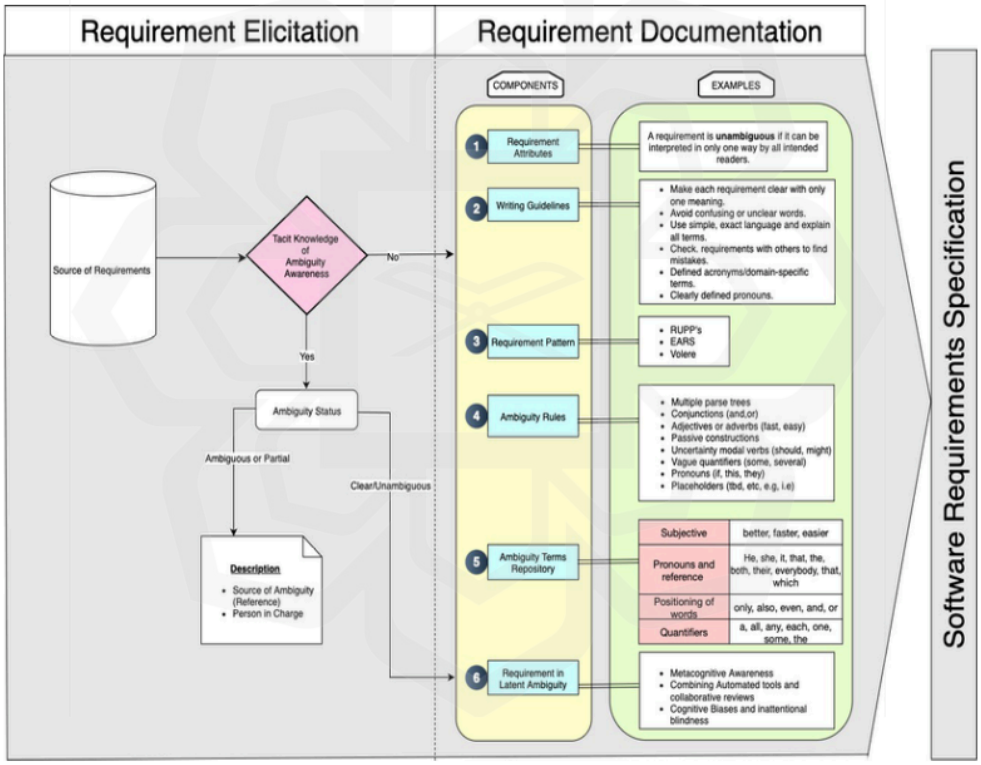
Save Submit

APPENDIX II : FRAMEWORK GUIDELINE

Guideline for Applying SADF in Software Requirements Specifications

1. **Purpose**
   This guideline helps Requirements Engineers detect and manage syntactic ambiguity in Software Requirements Specifications (SRS), starting from Requirement Elicitation through Requirement Documentation.

2. **Framework Overview**
   Phase 1: Requirement Elicitation
   Objective: Identify ambiguity awareness early during stakeholder interactions.
   Phase 2: Requirement Documentation
   Objective: Provide structured support to reduce ambiguity in written requirements.



3. **Steps/Guideline:**

Step 1:
Gather requirements from stakeholders via interviews, documents, or discussions.

Step 2:
Assess whether the Requirements Engineer (RE) demonstrates awareness of ambiguity risks during elicitation, including tacit (hidden) knowledge that may influence interpretation:
If ambiguity awareness is present, move to step 3.
If ambiguity awareness is not present, move to step 6.

Step 3:
RE classify the ambiguity status of the requirement:
If the classification is Ambiguous or Partially, move to step 4.
If the classification is Clear or Unambiguous, move to step 5.

Step 4:
RE will prepare the document indicating Source of ambiguity, Relevant references, Responsible person for resolution. Sample of the document as below:

Requirement ID	**Source of Ambiguity**	**Relevant Reference**	**Responsible person for resolution**
RQ-103 "The system should allow users to quickly access reports."	1) Term **"quickly"** is subjective and lacks measurable criteria. 2) No clear definition of acceptable response time.	1) IEEE Std 830-1998: Software Requirements Specification Guidelines. 2) SADF Ambiguity Terms Repository: Avoid vague adverbs like "quickly," "efficiently," "easily."	Name: Syara Role: Lead Requirement Engineer. Action: Replace "quickly" with a measurable performance metric (e.g., within 2 seconds)

Step 5:
RE will perform a latent ambiguity check which is under component 6 before inclusion in the final SRS.

Handling latent Ambiguity (Detection Strategies):

Definition	**Detection Strategies:**	**Details Guideline**
Ambiguity that emerges later during design or development.	**Strategy 1: Metacognitive Awareness:** Re-read from user's perspective, question assumptions.	**What it means:** Metacognition is "thinking about your thinking." In ambiguity detection, it means the RE actively reflects on how they interpret requirements and questions their assumptions. **How to apply:** Re-read requirements from the user's perspective: Ask, "If I were the stakeholder, would this sentence make sense without extra context?" **Question assumptions:** Identify any unstated conditions you might be assuming. For example, if a requirement says, "The system should respond quickly," ask, "What does 'quickly' mean? Is it 1 second or 10 seconds?" **Self-check:** Pause and ask, "Could someone else interpret this differently?" **Why it matters:** Tacit knowledge often hides ambiguity because RE assume shared understanding that isn't documented.
	Strategy 2: Combining Automated Tools with Collaborative Review: Use NLP-based tools to scan for hidden issues and get Peer or expert feedback to catch overlooked ambiguities.	**What it means:** Use Natural Language Processing (NLP)-based tools or requirement analysis software to scan for ambiguous terms and patterns. **Examples of tools:** • **Ambiguity checkers:** Detect vague words like *"fast," "easy," "efficient."* • **Grammar and syntax analyzers:** Highlight complex sentence structures that may cause misinterpretation.

Step 5:
RE will perform a latent ambiguity check which is under component 6 before inclusion in the final SRS.

Handling latent Ambiguity (Detection Strategies):

Definition	**Detection Strategies:**	**Details Guideline**
Ambiguity that emerges later during design or development.	**Strategy 1: Metacognitive Awareness:** Re-read from user's perspective, question assumptions.	**What it means:** Metacognition is "thinking about your thinking." In ambiguity detection, it means the RE actively reflects on how they interpret requirements and questions their assumptions. **How to apply:** Re-read requirements from the user's perspective: Ask, "If I were the stakeholder, would this sentence make sense without extra context?" **Question assumptions:** Identify any unstated conditions you might be assuming. For example, if a requirement says, "The system should respond quickly," ask, "What does 'quickly' mean? Is it 1 second or 10 seconds?" **Self-check:** Pause and ask, "Could someone else interpret this differently?" **Why it matters:** Tacit knowledge often hides ambiguity because RE assume shared understanding that isn't documented.
	Strategy 2: Combining Automated Tools with Collaborative Review: Use NLP-based tools to scan for hidden issues and get Peer or expert feedback to catch overlooked ambiguities.	**What it means:** Use Natural Language Processing (NLP)-based tools or requirement analysis software to scan for ambiguous terms and patterns. **Examples of tools:** • **Ambiguity checkers:** Detect vague words like *"fast," "easy," "efficient."* • **Grammar and syntax analyzers:** Highlight complex sentence structures that may cause misinterpretation.

		• **Requirement quality tools:** Some tools integrate with requirement management systems to flag unclear statements. **Collaborative Review** **Why it matters:** Manual review can miss hidden ambiguities. Automated tools provide consistency and speed. **What it means:** Bring in peers, domain experts, or stakeholders to review requirements together. **How to apply:** • **Peer review sessions:** Ask colleagues to interpret the requirement independently. • **Expert validation:** Domain experts can spot technical ambiguities that REs might overlook. • **Stakeholder feedback:** Confirm that the requirement matches their intent. **Why it matters:** Different perspectives reveal ambiguities that one person might miss due to familiarity or bias.
	Strategy 3: **Bias Awareness:** Avoid cognitive biases and inattentional blindness.	**What it means:** Recognize cognitive biases that affect interpretation and decision-making. **Common biases in requirements work:** • **Confirmation bias:** Seeing what you expect instead of what's written. • **Anchoring bias:** Relying too heavily on initial information. • **Inattentional blindness:** Missing details because you're focused on something else. **How to apply:** • **Slow down:** Don't rush through requirements. • **Challenge your first impression:** Ask, "Could this mean something else?"

		• **Use checklists:** They help counter bias by forcing systematic review. **Why it matters:** Bias can make REs overlook ambiguity even when it's obvious.

Step 6:
Since ambiguity awareness is not present during elicitation, proceed to the Requirement Documentation phase (Phase 2) for detailed ambiguity detection in each component (component 1-5).

After Component 1-5 has been performed, move to step 6 as every requirement must pass a latent ambiguity check before inclusion in the final SRS.

If clear → mark as safe for documentation.

Guideline for Component 1-5 as below:

Component 1: Requirement Attributes
RE must know the requirement attributes especially unambiguity to ensure each requirement is unambiguous meaning it can only be interpreted in one way by all intended readers.

Component 2: Writing Guidelines
Follow best practices for clarity and precision.

1) **Make each requirement clear with only one meaning.**
   Every requirement should be written so that all readers interpret it the same way.
   Example: Instead of "The system should respond quickly," write "The system should respond within 2 seconds."
2) **Avoid confusing or unclear words.**
   Words like fast, easy, efficient, user-friendly are subjective and lead to ambiguity. Replace them with measurable or well-defined terms.
   Example: Replace "easy to use" with "requires no more than 3 steps to complete a transaction."
3) **Use simple, exact language and explain all terms.**
   Avoid jargon unless necessary and define technical terms clearly.
   Example: If you use API, explain what it means for the context.
4) **Check requirements with others to find mistakes.**
   Conduct peer reviews or stakeholder validation sessions. Different perspectives help catch unclear or ambiguous statements.
5) **Define acronyms and domain-specific terms.**
   Always provide the full form of acronyms at first use.

Example: "Customer Relationship Management (CRM)" before using CRM alone. Clearly define pronouns

6) **Avoid pronouns like it, they, this without clear reference.**
   Example: Instead of "It should be fast," write "The payment process should complete within 2 seconds."

Component 3: Requirement Patterns
Use templates like RUPPS or EARS for consistency.

RUPPS Template
Provides reusable sentence structures for common requirement types.
Example: *"The system shall [action] when [condition]."*
EARS Template
Uses simple keywords to structure requirements:
Ubiquitous: Applies to all situations (*"The system shall…"*)
Event-driven: Triggered by an event (*"When [event], the system shall…"*)
State-driven: Based on system state (*"While [state], the system shall…"*)
Example: *"When the user clicks 'Submit,' the system shall validate the input."*

Component 4: Ambiguity Rules
Avoid patterns and structures that cause or introduce ambiguity in requirements. (e.g., vague pronouns, optional phrases).

1. **Multiple Parse Trees**
   A sentence that can be grammatically parsed in more than one way creates ambiguity.
   Example: *"Display the report for managers and employees."*
   Does it mean one report for both, or separate reports?
2. **Conjunctions (and/or)**
   Using *and/or* is unclear because it's not obvious whether both conditions apply or just one.
   Example: *"The system shall support PDF and/or Excel export."*
   Should it support both formats or just one?
3. **Adjectives or Adverbs (fast, easy)**
   Subjective words lead to different interpretations.
   Example: *"The system should respond quickly."*
   Replace with measurable criteria: *"within 2 seconds."*
4. **Passive Constructions**
   Passive voice hides the actor, making responsibility unclear.
   Example: *"Data shall be validated."*
   Who validates it? The system or the user?
5. **Uncertainty Modal Verbs (should, might)**
   Words like *should, might, may* imply optionality or uncertainty.
   Example: *"The system should allow data export."*
   Replace with *"shall"* for mandatory requirements.
6. **Vague Quantifiers (some, several)**
   These terms lack precision.

Example: *"Some users will have access to advanced features."*
Specify exact roles or numbers.

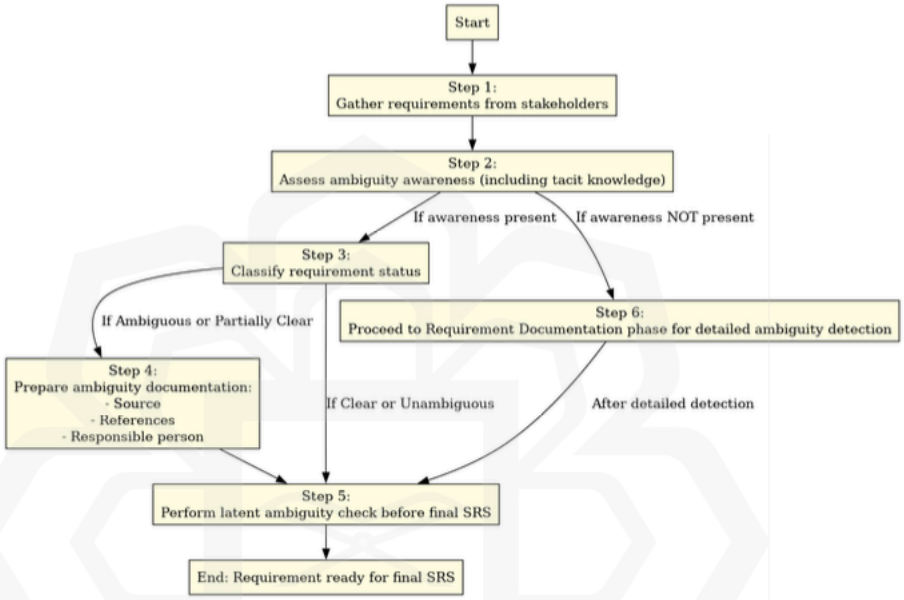
7. **Pronouns (it, this, they)**
   Pronouns without clear reference create confusion.
   Example: *"It should be fast."*
   Replace with explicit subject: *"The payment process should complete within 2 seconds."*
8. **Placeholders (TBD, etc., e.g., i.e.)**
   Placeholders indicate incomplete information and lead to ambiguity.
   Example: *"The system shall support multiple formats (e.g., PDF, Excel, etc.)."*
   Define all formats explicitly.

Component 5: Ambiguity Terms Repository
This component provides a reference vocabulary list of known ambiguous terms that Requirements Engineers (REs) should avoid when writing requirements.

Examples of Ambiguous Terms:
Adjectives/Adverbs: fast, easy, efficient, user-friendly
Quantifiers: some, several, many
Modal Verbs: should, might, may
Placeholders: TBD, etc., e.g., i.e.
Pronouns: it, this, they

How to Use:
Check every requirement against the repository.
Replace ambiguous terms with precise alternatives or measurable criteria.
Example:
Ambiguous: "The system should respond quickly."
Clear: "The system shall respond within 2 seconds."

4. Flowchart Representing the Ambiguity Detection Guideline in the Framework



SITI SYARA AIMAN BINTI SEH WALI	MSC	2025	IIUM